UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE STRAPAÇÃO GUEDES VIANNA

Testing Guidelines for Data Stream Processing Applications

Recife

2023

ALEXANDRE STRAPAÇÃO GUEDES VIANNA

Testing Guidelines for Data Stream Processing Applications

A Ph.D. Thesis presented to the Center for Informatics of the Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Concentration Area: Software Engineering

Advisor: Kiev Santos da Gama

Recife

2023

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Vianna, Alexandre Strapação Guedes.
Testing guidelines for data stream processing applications / Alexandre Strapação Guedes Vianna. - Recife, 2023.
249 f.: il.

Tese (Doutorado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciência da Computação, 2023.
Orientação: Kiev Santos da Gama.
Inclui referências e apêndices.

1. Processamento de fluxos de dados; 2. Teste de software; 3. Engenharia de software; 4. Testes de aplicações que processam fluxos de dados. I. Gama, Kiev Santos da. II. Título.

UFPE-Biblioteca Central

Alexandre Strapação Guedes Vianna

“Testing Guidelines for Data Stream Processing Applications”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Engenharia de Software e Linguagens de Programação.

Aprovada em: 18/12/2023.

_____

Orientador: Prof. Dr. Kiev Santos da Gama

BANCA EXAMINADORA

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática/UFPE

_____

Prof. Dr. Sérgio Castelo Branco Soares
Centro de Informática /UFPE

_____

Prof. Dr. Breno Alexandro Ferreira de Miranda
Centro de Informática /UFPE

_____

Profa. Dra. Thais Vasconcelos Batista
Departamento de Informática e Matemática Aplicada/UFRN

_____

Prof. Dr. Paulo Sergio Lopes de Souza
Instituto de Ciências Matemáticas e de Computação/USP

ACKNOWLEDGEMENTS

Um doutorado não se faz sozinho; ele depende da construção coletiva do conhecimento de uma comunidade de pesquisadores que, ao longo do tempo, pavimentaram o caminho que percorri. Esta é uma conquista da sociedade como um todo. Por isso, agradeço a todos que contribuíram de alguma forma, direta ou indiretamente.

Agradeço especialmente à minha esposa e companheira de vida, Lívia, que com amor esteve ao meu lado, me apoiando e incentivando ao longo desta jornada. Foram cinco anos em que ela escutou pacientemente meus dilemas e compartilhou das conquistas, como a aprovação de artigos, a qualificação e a defesa.

Minha gratidão aos meus pais, Pedro e Cristina, que sempre estiveram presentes em toda a minha trajetória de vida e estudos, oferecendo apoio, ensinamentos e exemplos de vida. Esse alicerce foi essencial durante o doutorado. Agradeço também às minhas irmãs, Catarina e Carolina, aos meus cunhados, Helder e Leandro, e à minha cunhada Vitória, por todo o apoio e incentivo. Aos meus sogros, Yvan e Tereza, sou grato pelo apoio constante e pela acolhida ao longo dessa jornada. E às minhas sobrinhas, Marina, Helena, Violeta e Aurora, agradeço pelos momentos de descontração, nos quais pude deixar de lado as preocupações.

Agradeço ao meu orientador, Kiev Gama, por seu papel fundamental ao me guiar por uma trajetória repleta de desafios. Ele foi sempre solícito, e, nas inúmeras reuniões que realizamos, me ajudou a encontrar soluções, sugeriu melhorias e me apresentou a uma rede de colaboradores de pesquisa. Além da orientação técnica e acadêmica, descobri em Kiev um ser humano excepcional, que me apoiou e acreditou em mim nos momentos mais difíceis. Suas palavras de incentivo e nossas conversas foram essenciais para que eu seguisse em frente. Obrigado!

Aos meus colegas de pesquisa, Carlos Zimmerle, Fernando Kenji, Waldemar Neto e João Neto, que foram parceiros incansáveis nos trabalhosos experimentos que conduzi. Em especial, agradeço a Carlos, por seu apoio contínuo em diversas etapas do trabalho, e a Kenji, pelo incentivo e suporte em vários momentos.

Sou grato também aos colegas do IFPE Campus Igarassu, que, além de oferecerem incentivo e apoio, possibilitaram o meu afastamento por dois anos, essencial para que eu pudesse me dedicar plenamente ao doutorado.

Agradeço ainda aos meus amigos de vida, Hugo, Luis, Amaro, Saulo, Magno, Lucas, João Paulo, Rodolfo, Eiji, Dario e Uirá, por todo o apoio e amizade.

Aos membros das bancas de qualificação e defesa – Nelson Rosa, Breno Miranda, Sergio Soares, Thais Batista e Paulo Souza – agradeço pela cuidadosa análise do meu trabalho e pelas valiosas contribuições, direcionamentos e sugestões que o ajudaram a se consolidar. Agradeço também a todos os docentes e funcionários do Cin/UFPE que, de alguma forma, contribuíram para este trabalho, fornecendo serviços e infraestrutura essenciais. Sou grato ainda ao INCT INES (Instituto Nacional para Engenharia de Software - CNPq/465614/2014-0), pelo apoio na pesquisa e na participação em eventos.

"Se vi mais longe, foi por estar de pé sobre os ombros de gigantes" (NEWTON, 1675).

RESUMO

A abordagem de Processamento de Fluxos de Dados (PFD) foca no processamento em tempo real, aplicando técnicas para captura de dados e subsequente processamento de resultados sem armazenamento prévio. Essa abordagem ganhou relevância na indústria devido ao crescimento da quantidade de dados gerados por diversas fontes. O PFD é valioso por extrair informações que são úteis em curtos períodos após a geração dos dados, aplicando-se em áreas como detecção de fraudes, comportamento anômalo em sistemas de computadores e monitoramento industrial. Com a crescente adoção em diversos setores, testar aplicações de PFD torna-se relevante ao mesmo tempo que apresenta desafios devido a fatores como requisitos de performance, temporalidade das mensagens, paralelismo de processamento, volume e variabilidade de dados, complexidade da infraestrutura e não-determinismo.

Este trabalho visa desenvolver e avaliar diretrizes para testes de aplicações DSP, abordando aspectos relevantes para a indústria e colaborando com profissionais na identificação de práticas atuais. A metodologia inclui três etapas: investigação, proposição e avaliação. A investigação envolveu estudos empíricos com praticantes do PFD, incluindo um estudo exploratório com questionários e entrevistas para validar a relevância do tema, identificar desafios e mapear práticas. Seguiu-se uma revisão de literatura cinza, analisando 154 documentos para revelar desafios, objetivos, técnicas, estratégias e ferramentas de testes no contexto industrial de PFD.

A etapa de proposição consistiu no desenvolvimento de diretrizes de testes fundamentadas nas informações coletadas na fase de investigação. A avaliação das diretrizes envolveu grupos focais e uma pesquisa com profissionais, visando entender percepções, benefícios, fraquezas, melhorias e aplicabilidade das diretrizes no contexto industrial. Os resultados indicaram uma percepção positiva das diretrizes, com sugestões de melhorias incorporadas na versão final.

Em resumo, esta tese investigou um tema emergente na indústria por meio de metodologias adequadas e colaboração de praticantes, contribuindo para diminuir a lacuna entre o conhecimento acadêmico e industrial sobre testes de aplicações de PFD. As diretrizes desenvolvidas foram avaliadas e disponibilizadas online, representando uma contribuição tangível à comunidade de PFD.

Palavras-chaves: processamento de fluxos de dados. teste de software. engenharia de software. testes de aplicações que processam fluxos de dados.

ABSTRACT

The Data Stream Processing (DSP) approach focuses on real-time data processing, employing data capture techniques and processing on-the-fly results (without prior storage). This approach has gained significance in the software industry due to the growth in the data volume generated by various sources. DSP is valuable for extracting useful information shortly after data generation and is typically used in areas such as fraud detection, anomalous user behaviour monitoring in computer systems, and industrial equipment monitoring. With its increasing adoption across various sectors, testing DSP applications becomes relevant while presenting challenges due to factors like performance requirements, message temporality, processing parallelism, data volume and variability, infrastructure complexity, and non-determinism.

This work aims to develop and evaluate guidelines for testing DSP applications, addressing aspects relevant to the industry and collaborating with professionals in identifying current practices. The methodology encompasses three main phases: investigation, proposition, and evaluation. The investigation involved empirical studies with DSP practitioners, including an exploratory study with questionnaires and interviews to validate the topic's relevance, understand practical aspects, map challenges, and identify topics for deeper exploration in subsequent studies. This was followed by a Grey Literature Review (GLR), analyzing 154 documents to identify challenges, testing objectives, techniques, and tools in the industrial context of DSP.

The proposition phase consisted of developing the testing guidelines for DSP applications, grounded in the insights collected during the investigation phase. The final phase was the evaluation of the proposed guidelines, involving focus groups and a survey with industry professionals to assess perceptions, benefits, weaknesses, areas for improvement, and the applicability of the guidelines in the industrial context. The results indicated a positive reception of the guidelines, with suggestions for improvements incorporated into the final version.

In summary, this doctoral thesis investigated an emerging topic in the industry employing appropriate methodologies and practitioner collaboration. The resulting publications contribute to bridging the gap between academic and industrial knowledge regarding DSP application testing. The developed guidelines were evaluated and made available online, representing a tangible contribution to the DSP community.

Keywords: data stream processing. software testing. software engineering. data stream processing applications testing.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

This chapter introduces the work at hand. Section 1.1 contextualises the problem's theme, challenges and research motivation for the problem. Section 1.2 presents the general objective as well as the specific ones. Section 1.3 then lists the central research questions and develops derived research questions. Finally, Section 1.4 describes the document's structure.

1.1 CONTEXT AND MOTIVATION

Over the last few years, there has been a considerable increase in data generated by computer systems such as Internet of Things (IoT) devices (MORALES et al., 2016), smartphones, and user interaction in the real world and websites (HASAN; ORGUN et al., 2018; FIGUEIRAS et al., 2018; FAWCETT; PROVOST, 1999). In parallel, streaming data – not media (video/audio) streaming – is also becoming important for social media platforms such as Facebook, Twitter, and LinkedIn (PÄÄKKÖNEN, 2016). These companies were not prepared to deal with the large volume of data generated by the interactions of millions of users on social networks and, due to the lack of tools, companies have developed their own stream processing tools: Facebook's Puma, Swift, and Stylus; Twitter's Storm and Heron; and LinkedIn's Kafka (PANDEY et al., 2019). All these systems produce a high volume of events, increasing the data processing demand. Approaches in the context of Big Data were proposed to deal with this issue (Philip Chen; ZHANG, 2014; KAISLER et al., 2013). Conventional data processing strategies, such as batch processing, store all data before processing it (BABCOCK et al., 2002). This method adds latency and may not handle large volumes of data as fast as some applications require. Nowadays, low latency is critical in some time-sensitive application contexts like fraud detection systems (ASSUNCAO; VEITH et al., 2018). In these circumstances, Data Stream Processing (DSP) has emerged as a branch of Big Data to handle real-time and data-intensive applications (LIU; IFTIKHAR; XIE, 2014).

The DSP approach focuses on real-time data processing. It applies specific techniques for capturing and processing relevant data for on-the-fly results, i.e., without requiring storage (STONEBRAKER; ÇETINTEMEL; ZDONIK, 2005). This approach allows the processing of the data immediately after its generation. The stream processing strategy is recommended when there is a need to continuously query data flows to detect patterns within a short time inter-

val (GOLAB; ÖZSU, 2003). Many business sectors exploit DSP benefits, such as the financial market, banking system, e-commerce, marketing, social networks, communication industry, infrastructure monitoring, and others (GAROFALAKIS; GEHRKE; RASTOGI, 2016). The solutions and tools around DSP are advancing quickly due to the business demands and the continuous need to provide solutions in that context.

Like in any other software, testing to find bugs in features and verify compliance with non-functional requirements plays an essential role in the quality assurance of DSP. Besides, DSP is significantly involved in critical applications in several organizations' businesses, where the impacts of failures can be catastrophic or cost many resources. For example, a failure to detect fraud in credit card transactions can result in a considerable loss in just a few minutes (KHINE; KHIN, 2020). In other situations, such as a smart grid for monitoring power infrastructures, DSP focuses on predicting failures that can cause a collapse of essential systems that can affect different sectors of society (CHOI et al., 2018).

However, testing such kind of software is not a simple task due to factors hindering testing such as message temporality, parallelism, data volume, data variability, message speed, etc (PUNN et al., 2019). There are many difficulties related to distributed processing infrastructure, in which data from different sources are processed in geographically dispersed clusters (CHEN; PAIK; LI, 2016). In addition, data arrives at high speed and the applications that process it must do so under very strict constraints. Therefore, data streams pose several challenges for application testing.

Compared to conventional databases, DSP is a field that has been maturing over the last decades. There are still no standards or semantics that publish consensus across the community. Consequently, a DSP test infrastructure must be extensible and well-prepared for the new challenges in this field (RAIZMAN et al., 2010). Designing a test infrastructure covering all these factors is difficult since it is challenging to simulate configurations where errors manifest (RAIZMAN et al., 2010). Developing an efficient and effective streaming test system is complex as it requires people who are skilled and capable of understanding the stream data to be processed, considering the technical nuances of stream processing infrastructure.

Nevertheless, the relevance of testing DSP in the industry can be identified in many informal sources such as tools discussion lists (FOUNDATION, 2022), question & answer sites (Stack Overflow, 2017; Stack Exchange, 2017), tools documentations (Apache Flink, 2021; Amazon Kinesis, 2017), lecture themes at technical events (MALASKA, 2019; WIESMAN, 2018; GAMOV, 2020), professional social networks (WAEHNER, 2022), blog posts (ALADEV, 2021; OSWILL, 2019;

KHARE, 2020), open-source software repositories (Mocked Streams, 2016; AUTHORJAPPS, 2019; LEOPARDI, 2017; KARAU, 2016), podcast (Software Engineering Daily, 2020), and other web-published materials. Hence, it is noticeable that the community has been promoting collective advances in specific issues, such as developing strategies, good practices, open-source tools, and libraries for testing DSP applications. Still, the industry know-how in DSP application testing is fragmented into several online documents, such as websites, blogs, forums, and software repositories.

Although widely adopted in the industry, there is limited formal literature on testing DSP applications. The existing formal literature brings contributions addressing particular issues of testing DSP applications by proposing strategies, techniques, and tools for specific cases. However, there is a lack of academic work broadly addressing the subject and relating the different aspects of the whole context. Therefore, due to advances in practical industry DSP issues that are driven by market demand there is a gap between industry knowledge and academic knowledge about testing DSP applications. Also, there are no consolidated materials, such as roadmaps, guidelines, or best practices compilation, guiding practitioners in DSP application testing.

In this sense, this doctoral work aims to research through collaboration with the DSP application academic and industry community. Although there are many challenges in conducting software research involving industry-academia collaborations, it promotes many benefits (WOHLIN, 2013). For researchers, collaboration allows them to access real-world problems and datasets, and to obtain feedback from experienced practitioners in developing robust systems (RODRÍGUEZ; KUVAJA; OIVO, 2014). As for industry, collaboration supports technological improvement and innovation in the sector, also promoting industrial relevance in academic research (GAROUSI et al., 2017).

1.2 OBJECTIVE

Recognizing the significance of DSP in today's landscape—especially since this technology is valuable to the business operations of many companies—and considering the pivotal role of testing within the DSP context for enhancing software quality, combined with the observed lack of comprehensive guidelines for testing DSP applications, our guiding research hypothesis is:

Hypothesis: *"It is feasible to formulate valuable guidelines for the industry grounded on diverse pieces of knowledge and experiences from practitioners, which have been systematically collected, selected, and synthesized through appropriate research methods."*

Therefore, our research aims to advance the state-of-the-art in DSP application testing by addressing these issues, guiding us to our main objective:

Main Objective: *"To develop and evaluate testing guidelines for DSP application by studying industry-relevant aspects and collaborating with practitioners to identify current industry practices."*

To achieve this primary objective, we established the following specific objectives:

- Explore the field of testing in the context of DSP through studies involving practitioners to identify relevant practical aspects.
- To investigate in depth the domain of industry practices regarding the testing challenges, testing purposes, testing techniques, strategies for obtaining testing datasets and the tools employed in order to build a base of relevant information about testing practices in the field.

1.3 RESEARCH QUESTIONS

To achieve the objective, this thesis aims to answer the following research questions:

RQ1: How do practitioners deal with DSP application testing?

With this central question, we investigated the following derived questions:

RQ1.1 What are the challenges to DSP application testing?

RQ1.2 What are the testing purposes?

RQ1.3 What are the approaches and specific types of tests performed?

RQ1.4 What are the strategies adopted by practitioners to obtain testing data?

RQ1.5 What are the tools, and under what circumstances are they used in DSP application testing?

RQ2: What are the practitioners' perceptions about the proposed guidelines?

With this question, we aim to collect the impressions of experienced practitioners regarding the proposed guidelines. The choice of evaluation with practitioners was because they are the target users of the proposed guidelines and have the knowledge and practical experience to provide feedback aligned with the industry reality. Specifically, we seek to evaluate practitioners' impressions regarding practical applicability, expected benefits, positive aspects, and potential adverse effects while also gathering suggestions for enhancement.

1.4 CONTRIBUTIONS

During the course of the investigation, this thesis made three main contributions.

- **Exploratory Study on Practitioners' Impressions of Testing in the DSP Context:** We carried out the first exploratory study on this topic with practitioners who work directly with DSP in the industry. The study included 12 semi-structured interviews and gathered 101 survey responses. Our findings highlighted the significance of this topic for the industry and recorded practitioners' perceptions about testing DSP applications. Furthermore, the study offered valuable insights and data to guide future research in this domain. Detailed results and contributions from this study can be found in Chapter 4.
- **Study regarding the state of practice in testing DSP applications:** At the same time, we conducted a comprehensive Grey Literature Review (GLR) on the state of practice in testing DSP applications. This was the first review of its kind on this topic, resulting in the mapping and synthesis of practical knowledge and experience regarding testing DSP applications. Our methodology led to the selection of 154 sources, helping us identify the primary challenges of testing, main test objectives, specific testing approaches, and strategies for obtaining test data. Additionally, we reported 50 tools utilized in different testing activities. This study gathers extensive information from an industrial perspective, guiding future research in the domain. The complete study is detailed in Chapter 5.

- **Testing Guidelines for DSP Applications:** The primary contribution of this thesis is the development (including the validation with practitioners) of guidelines designed to support professionals in planning DSP application testing. These guidelines take into account the unique characteristics of DSP applications. They were formulated based on previous primary study findings and further refined with practitioners' feedback. To make it easily accessible to developers, the guidelines have been made available in website format, representing a tangible research contribution to the community.

1.5 DOCUMENT STRUCTURE

The remainder of this document is structured as follows:

- **Chapter 2** exposes the fundamentals of DSP, the central concept of this thesis. It also explains the standard technologies infrastructure, data stream analysis approaches, and state-of-the-art data stream testing.
- **Chapter 3** details the research methods, describing methodological aspects of the investigation, guidelines development process and evaluation phases.
- **Chapter 4** presents the results of the exploratory study based on interviews, questionnaires, and grounded theory. It shows demographic data, analysis, results synthesis, discussion, and threats to validity.
- **Chapter 5** presents the results of the GLR. It describes the GLR method application, demographic data and a concise summary of the results along with the discussion.
- **Chapter 6** presents the testing guidelines for DSP applications, the main contribution of this doctoral thesis.
- **Chapter 7** presents the conclusions and contributions of this thesis and directions for future research.

2 FUNDAMENTAL CONCEPTS

This chapter presents the fundamentals of DSP, the central concept of this thesis. The chapter begins with Section 2.1, which presents the data stream definition and a brief history. Section 2.2 presents the consolidated DSP infrastructure. Section 2.3 describes data stream analysis approaches. Finally, state-of-the-art in data stream tests is presented in Section 2.5.

2.1 DATA STREAM

The first studies about DSP were published in the 60s (STEPHENS, 1997). They presented some concepts about dataflow systems and the evolution of Database Management Systems (DBMSs). In the 70s, the Lucid language was developed with the objective of processing dataflows (precursor of the data stream) (WADGE; ASHCROFT, 1985). The volume of data traffic increased with the expansion of personal computers and the Internet in the 1980s. The data volume peaked in the 21st century with smartphones and IoT devices. Unfortunately, traditional data management systems did not offer real-time support for this data volume (ZHAO et al., 2017). In the current century, another challenge is the speed of value changes in data streams since it has to be processed continuously in real-time. Researchers are taking advantage of parallel processing and distributed computing studies, and have proposed techniques to deal with these challenges. Nowadays, the main techniques integrating the base of DSP are Map-reduce (CONDIE et al., 2010), continuous queries (BABU; WIDOM, 2001) and temporal data models (KRÄMER; SEEGER, 2004). The evolution of concepts and techniques associated with DSP leads us to the following definition.

> **Definition:** A data stream can be understood as a sequence of unbounded and never-ending (potentially) generated data, in which the order of arrival cannot be controlled (GOLAB; ÖZSU, 2003). Due to the massive volume and speed of data, it is not convenient to store it and its computation must be done in real-time without prior storage by queries running continuously against data streams.

In summary, a Data Stream Management System (DSMS) processes a continuous and infinite data stream immediately upon arrival without prior storage. In contrast, traditional Database Management Systems (DBMS) aggregate data to represent the final state of an

operation. Once the results are aggregated, they are stored in the database and subsequently queried. Figure 1 illustrates the two approaches. To clarify the distinction, we examine the approaches within an e-commerce context. Online shoppers frequently add and remove items from their carts during a shopping session. Using data aggregation, the DBMS-based method captures only the cart's final state. The DSMS approach continuously keeps processing the shopping cart events stream (addition or remotion of products), evaluating all cart operations in real-time. The DSMS approach allows for in-depth event analysis, such as the identification of users' shopping behaviour patterns.

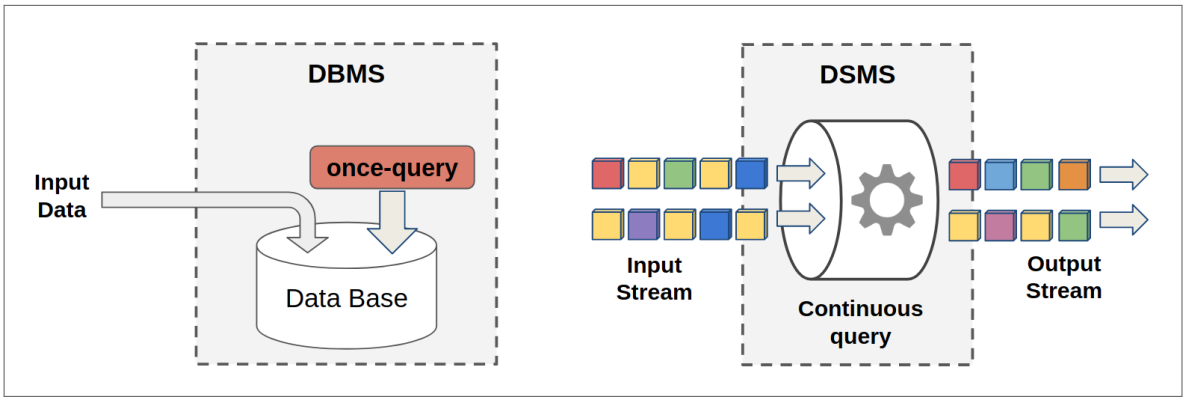


Figure 1 – Traditional DBMS processing versus DSMS processing.

The software industry has successfully adopted solutions using this approach for applications with high data volume requiring real-time answers. For example, anti-fraud systems, real-time dashboards, clickstream for user behaviour analysis (HANAMANTHRAO; THEJASWINI, 2017), analysis of user engagement in marketing campaigns, user sentiment on social networks (HASAN; ORGUN et al., 2018), geofences (virtual fences based on location sensors), monitoring IoT data in smart cities (TÖNJES et al., 2014), investment robots in the stock market and equipment failure detection. In these situations, the result of any analysis would be worthless if delivered late. DSP is constantly evolving, and its importance has become remarkable in recent years. There are many challenges around data streams (MEHMOOD; ANEES, 2020), such as the autonomous management of DSP, fault tolerance (VIANELLO et al., 2018), resource elasticity (ASSUNCAO; VEITH et al., 2018), data stream mining (KREMPL et al., 2014), evolving data streams (BIFET et al., 2009), and integration with machine learning algorithms (KRAWCZYK, 2016; NOKLEBY; RAJA; BAJWA, 2020).

2.2 DATA STREAM INFRASTRUCTURE

In recent years, many technologies have emerged to compose the DSP infrastructure (KOLAJO; DARAMOLA; ADEBIYI, 2019; FRAGKOULIS et al., 2023). Analysing architectures proposed and discussed in the literature (TANTALAKI; SOURAVLAS; ROUMELIOTIS, 2020; NAMIOT, 2015; KIRAN et al., 2015), we have identified five essential categories of components: producer, ingestion, stream analytics, storage and delivery. Figure 2 presents the traditional organization of this infrastructure.

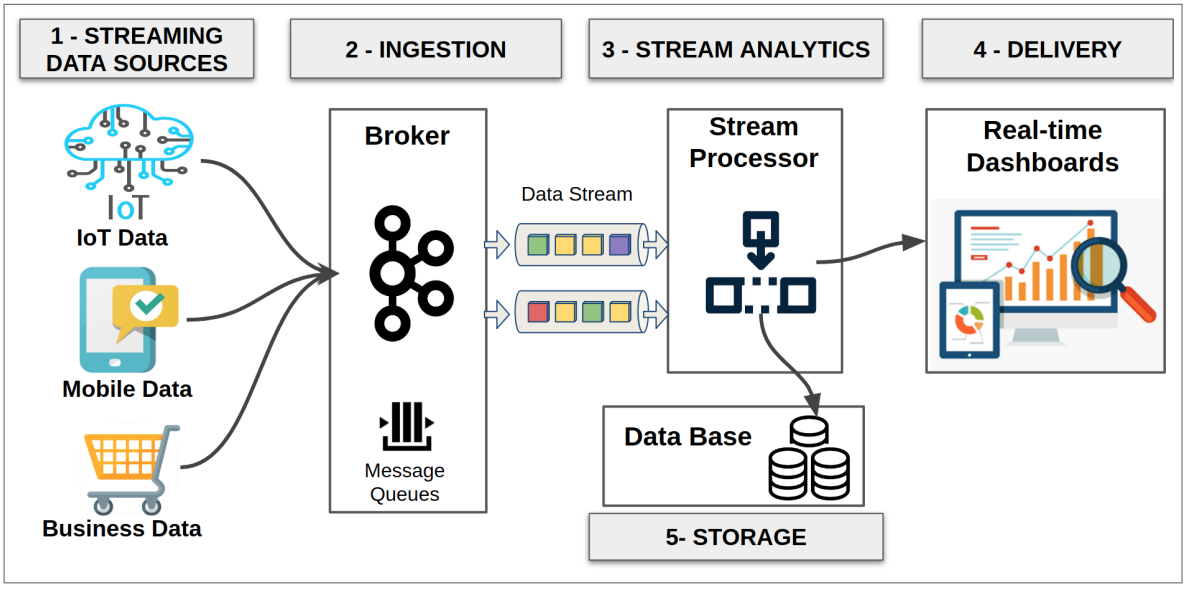


Figure 2 – DSP Infrastructure.

Bellow, a list of the infrastructure components is presented, along with a brief description and examples of tools adopted in each case.

1. **Streaming Data Sources:** Data producers send data to message brokers. In general, they can be sensors or equipment from the IoT universe, smartphone applications, social networks, business data or thousands of other data sources on the Internet.

2. **Ingestion:** Some message brokers (e.g., Kafka) are systems focused on stream processing, and they work similarly to the traditional publish-subscribe messaging system by managing queues of producers and consumers (MEEHAN et al., 2017). However, instead of standardized protocols, such as AMQP, MQTT and STOMP, message brokers for stream processing adopt a custom protocol on top of TCP/IP for communication between applications and the clusters. Moreover, they are more robust tools designed for

high performance. Message brokers for stream processing are distributed systems with parallelization support, redundancy, and fault tolerance features (ISAH; ZULKERNINE, 2018). Furthermore, these tools are also concerned with specific data streams aspects, such as the semantics of message processing: at-least-once, at-most-once, and exactly-once. Some examples of popular ingestion tools are Apache Kafka, Amazon Kinesis, Flume and Apache NiFi.

3. **Stream Analytics:** Stream processors are broker consumers who perform data stream analysis processing logic. They apply several specific data stream analysis techniques, such as CEP, windowing, summary, machine learning, and others (described in Section 2.3) (SAHAL; BRESLIN; ALI, 2020). The analysis can generate alerts sent to applications and aggregate data sent to real-time dashboards or databases. The stream processor output can also result in a new data stream sent to the broker as input. Source examples of stream processor tools are Apache Flink, Spark Streaming, Kafka Streams, and Kinesis Data Streams.

4. **Delivery:** Finally, at this point, the stream processing results reach their final goal by feeding dashboards and Business Intelligence tools to managers (GÜRCAN; BERIGEL, 2018).

5. **Storage:** Aggregated data from stream processing can be stored in databases. Several database systems are suitable for handling large volumes of data, making it possible to employ big data and data mining techniques in further processing (WANG et al., 2020). Some examples of storage tools for this context are S3 Dynamic DB, Azure Cosmos DB, Cassandra, Elastic Search and HDFS.

2.3 DATA STREAM ANALYSIS APPROACHES

A data stream analysis involves extracting relevant data and detecting patterns with very short delays (i.e., low latency). However, analysing stream data can be quite challenging because of the massive and endless amount of data that continuously arrives. Researchers are taking advantage of algorithms and distributed processing advances (CHERNIACK et al., 2003), proposing techniques to address the challenges of processing massive volumes of data efficiently to meet real-time requirements. Filters may include a sequence of operations-based

transformations, such as selections, aggregations, joins, and operators defined by relational algebra (CUGOLA; MARGARA, 2012a). Well-known techniques involve windowing, Complex Event Processing (CEP), machine learning, filters, randomization, sampling, synopsis, sketches, and summaries.

2.3.1 Windowing

Windowing is a stream processing technique where elements are grouped according to timestamps or quantity. Windowing is also known as micro-batches and is considered near real-time processing. Depending on the window size, the delay window increases. Latency typically ranges from a few milliseconds to a few seconds (VEITH; ASSUNCAO; LEFEVRE, 2021; QIN; EICHELBERGER; SCHMID, 2019). Several windowing strategies are applicable for different purposes, such as tumbling, hopping, sliding, session and snapshot (AKIDAU et al., 2015; CARBONE et al., 2016; GOLAB; OZSU, 2022). Below, we describe two well-known windowing approaches: sliding Windows and tumbling windows.

2.3.1.1 Sliding Window

Sliding windows continuously slide through data streams and are delimited by time intervals or the number of elements. The sliding window allows window overlapping between executions so that elements can belong to more than one sliding window and be processed many times in consecutive executions. Figure 3 illustrates the execution of a sum function in a sliding time window with a five-second size, while Figure 4 shows a five-element sliding window.

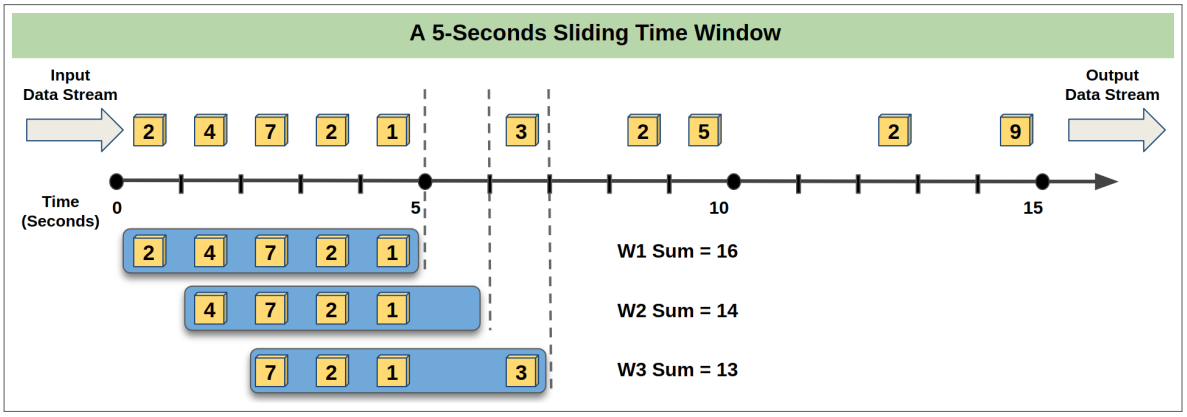


Figure 3 – A 5-Seconds Sliding Time Window.

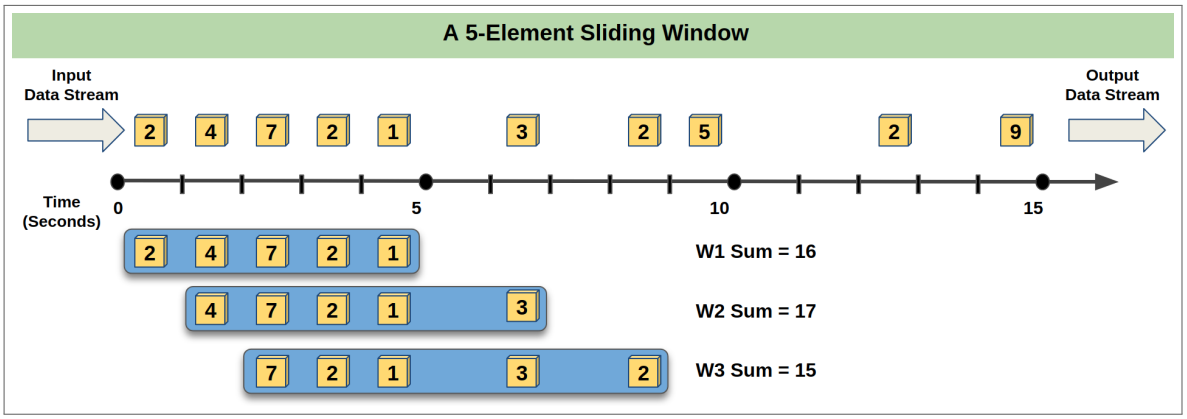


Figure 4 – A 5-Elements Sliding Window.

2.3.1.2 *Tumbling Window*

A tumbling window segments a data stream into distinct and non-overlapping segments. The critical aspects of a tumbling window are that they repeat but do not overlap, so an event cannot belong to more than one tumbling window. A five-second tumbling time window execution is illustrated in Figure 5, and Figure 6 shows a five-element tumbling window.

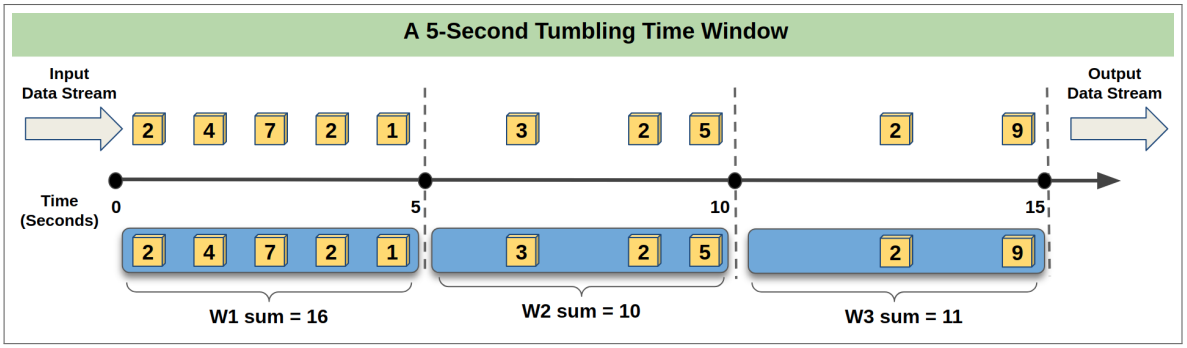


Figure 5 – A 5-Seconds Tumbling Time Window.

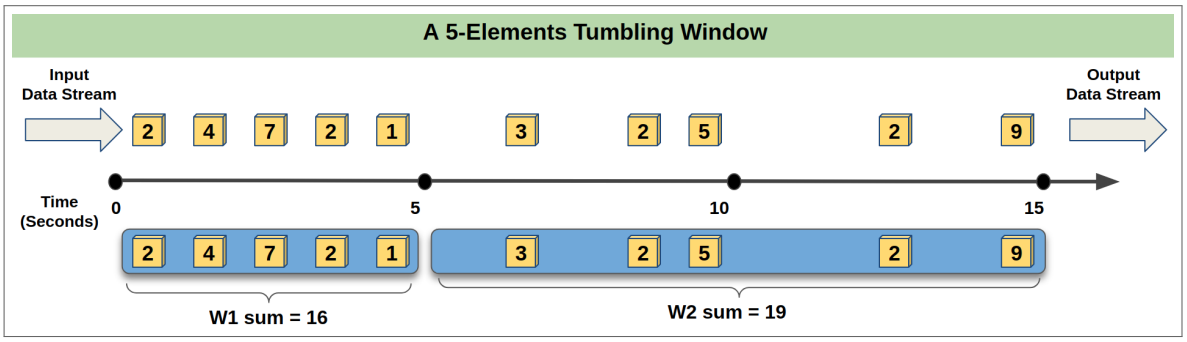


Figure 6 – A 5-Elements Tumbling Window.

2.3.2 Watermarks

Watermarks are crucial for managing event time and addressing out-of-order events in data streams. They act as a timestamp indicating the completion of all data events with a timestamp earlier than or equal to the watermark's time. This mechanism helps overcome challenges like network latencies and varying processing speeds, which cause unordered event arrival. Watermarks link processing time (when events are processed) and event time (when events actually occur). They provide a lower-bound timestamp for all unreceived events, ensuring that future records will have event times greater than the watermark. This allows DSP systems to accurately perform time-sensitive operations, such as windowed aggregations and joins, by informing when a time window is complete and ready for processing (AKIDAU et al., 2021). They are often used with windowing techniques to determine time window boundaries accurately. They are essential in large-scale, real-time DSP systems for handling event timing despite the unordered nature of event arrival.

2.3.3 Complex Event Processing

CEP is usually associated with identifying complex complex patterns based on various events from different sources and temporal relationships (BUCHMANN; KOLDEHOFE, 2009). This technique processes sequential primitive input events against a pattern to detect and report composite events, as illustrated in Figure 7. Employing CEP facilitates the aggregation of diverse information, analysis of cause-and-effect relationships between events, and provides insights into critical business issues. These capabilities enable proactive action-taking.

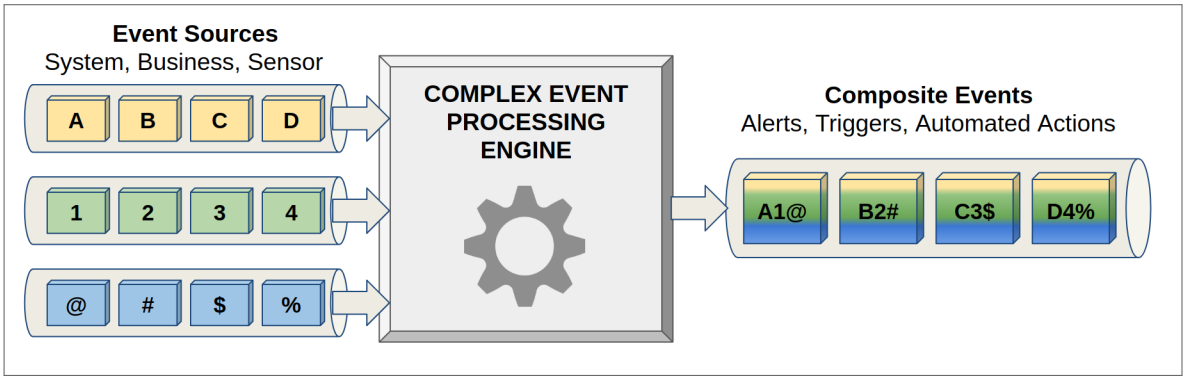


Figure 7 – Complex Event Processing Diagram.

Despite the event's complexity, queries are built with high-level languages inspired by

relational algebra (SQL-like) (LANGHI; TOMMASINI; VALLE, 2020). Even the construction of queries can be easily adapted to web-based and graphical tools (ZIMMERLE; GAMA, 2018), making it easier for non-developer managers to construct custom searches. CEP is normally adopted for scenarios in which there is a high volume of events occurring. Main application fields for CEP include real-time marketing, predictive maintenance, stock market trading, sensor networks and business activity monitoring.

2.3.4 Learning from Data Streams

Traditional machine learning algorithms have been adapted for pattern detection in data stream contexts, such as anomaly detection, trajectory prediction, failure prediction, outliers and detection of abnormal activities (VALSAMIS et al., 2017). It is a very challenging task (NOKLEBY; RAJA; BAJWA, 2020; VRIES; SOMEREN, 2012) and costly to employ machine learning in a data stream, mainly due to the high-speed requirements, limited computational resources (BU et al., 2009), time constraints and the non-stationary dynamic environment. Often, it is necessary to sacrifice accuracy in order to meet the application's basic requirements, which probabilistic models can solve. Another significant challenge is the concept of drift, where the underlying distribution of the data changes over time. This drift can be gradual or abrupt, affecting the model's performance as previously learned patterns become obsolete.

There are fundamental questions in learning from stream data, such as continuously maintaining learning models that evolve over time, learning and forgetting deviating concepts, and detecting changes. Traditional systems learn models inaccurately when they mistakenly assume that the underlying concept is stationary. The concept drift is a remarkable challenge for learning data streams. In the literature, many works address methodologies and strategies to deal with this issue and mitigate the problem (KHAMASSI et al., 2018; SAYED-MOUCHAWEH, 2016; DEMŠAR; BOSNIĆ, 2018).

2.3.5 Summarizing Techniques

The massive volume and velocity of data present significant challenges in delivering accurate results for streaming data. As the required level of accuracy increases, the demand for computing resources correspondingly escalates. In many cases, approximate responses are satisfactory when they are highly reliable. In this context, it is possible to use summary tech-

niques derived from statistical science to deal with memory limitations. The algorithms' design must also be based on computational theory techniques to implement time and space-efficient solutions. Following, we present a list of some summary techniques:

- **Sampling:** is the process of statistically selecting input stream elements that should be analyzed. Sampling has often been applied as a data reduction technique to produce approximate responses to queries about data streams. Sampling is essential in developing techniques for clustering data streams (IKONOMOVSKA; LOSKOVSKA; GJORGJEVIK, 2007).
- **Aggregation:** represents the elements in an aggregated value using some statistical measure, such as average, variance, or the sum (BABU; WIDOM, 2001).
- **Frequency:** Knowing the frequency of an element over time is desirable for many applications. For example, in digital marketing, we want to calculate how many times an advertisement has been shown throughout the day.

2.4 SOFTWARE TESTING

In this section, we address software testing and include observations in the context of DSP, and highlight the need to adapt traditional testing paradigms. Building on insights from Felderer et al. (FELDERER; RUSSO; AUER, 2019), we highlight the distinct challenges posed by DSP systems, such as their dynamic data flows, real-time processing demands, and the complexities of ensuring data integrity and system robustness. Below we detail three relevant issues: Test Oracle, Test Adequacy Criteria and Test Selection Criteria. We seek to provide a nuanced understanding of how software testing strategies should evolve to meet characteristics of DSP applications.

2.4.1 Test Oracle

Test Oracles in traditional software testing serve as a benchmark for evaluating the accuracy of a program's output. However, the inherent characteristics of DSP – continuous and unlimited data streams, real-time processing, and dynamic data schemas – present unique challenges that require a re-evaluation and adaptation of traditional methodologies for building test oracles.

Barr et al. (BARR et al., 2014) highlight the inherent difficulties in accurately specifying expected results in environments characterized by high variability and non-determinism, such

as DSP. They advocate the adoption of more flexible and adaptive test oracle designs that can accommodate the unpredictable nature of non-determinism systems, such considerations are applicable to the DSP context. Weyuker (WEYUKER, 1982) previously introduced the notion of pseudo-oracles, which may be particularly relevant in DSP testing, providing a means of approximating expected results when an exact comparison is impractical.

Vinay Arora's et al. (ARORA; BHATIA; SINGH, 2016) provides a comprehensive review of techniques that could be highly relevant for constructing Test Oracles in DSP. The article classifies testing approaches for concurrent programs into eight categories: reachability testing, structural testing, model-based testing, mutation-based testing, slicing-based testing, formal methods, random testing, and search-based testing. These approaches are particularly pertinent to DSP due to similarities in handling non-deterministic behaviors, synchronization issues, and ensuring correctness under dynamic operational conditions.

For instance, model-based testing can aid in dynamically adjusting Test Oracles to DSP system specifications or data schema changes. Reachability testing and formal methods could support the validation of streaming data processes against a range of acceptable outcomes. Mutation testing and slicing-based testing can enhance the robustness of Test Oracles by simulating potential anomalies or variations in data streams, thus ensuring sensitivity to unexpected changes. These methodologies from concurrent program testing could inform the development of flexible, dynamic, and robust Test Oracles for DSP, addressing the unique challenges posed by real-time, data-intensive systems. The adoption of these approaches can ensure the effective and reliable testing of DSP systems, enhancing their dependability in critical applications.

Incorporating stateful considerations into test oracles, as suggested by Carzaniga et al. (CARZANIGA et al., 2014), addresses the temporal aspects of DSP applications where the correctness of an output may depend on previous states or sequences of events. This emphasizes the need for test oracles that can not only validate individual outputs, but also the integrity of processing logic over time.

Adapting test oracles to DSP involves adopting these methodologies – dynamic, probabilistic and stateful validation – to ensure effective testing of DSP systems. These adaptations preserve the core purpose of test oracles, but reflect the unique requirements and challenges of DSP environments.

2.4.2 Test Adequacy Criteria

In the domain of software testing, test adequacy criteria serve as a reference to determine when the testing phase of a software system is considered sufficiently complete, attesting to a level of reliability and correctness of the system (CHEN et al., 2020; KAPUR; SHRIVASTAVA; SINGH, 2017). Traditional criteria include code coverage – encompassing declaration, branch, and path coverage – as well as fault detection rates. The choice of criteria depends on the system's context, and the system's criticality must be considered to establish the desired level of reliability (LV; YIN; CAI, 2014; SANTOS et al., 2017).

Traditionally, test adequacy criteria have focused on static aspects of software, such as code structure or functional requirements. Methods like structural testing (HOSSAIN et al., 2023), functional testing (SEGURA; BENAVIDES; CORTÉS, 2008), model-based testing (ANDREWS et al., 2003) and fault-based (RUTHERFORD; CARZANIGA; WOLF, 2008) testing have been extensively applied. These leverage criteria, such as code coverage, ensure that all program elements are executed at least once during testing and that various input scenarios are explored to uncover potential errors (ZHU; HALL; MAY, 1997). Such methodologies have established a baseline for assessing test completeness, guiding testers in identifying unexplored parts of the application and informing decisions on when additional testing is no longer productive.

However, DSP systems introduce dynamic and temporal dimensions that necessitate reevaluating these traditional criteria. DSP systems are characterized by continuous data ingestion, real-time processing requirements, scalability, state management, and dynamic data patterns, all influencing the decision of when to halt testing activities. This context requires methodologies to assess the system's endurance over lengthy periods without performance or accuracy degradation (HIRZEL et al., 2014; CARBONE et al., 2015). For instance, the system's ability to adapt to changing data patterns and scale based on incoming data volume necessitates dynamic testing criteria to evaluate the system's performance and accuracy under varying conditions (TANTALAKI et al., 2020; CARDELLINI et al., 2022). Additionally, state management in DSP—whether stateful or stateless—adds complexity in ensuring correctness and consistency across distributed components, especially in failure recovery scenarios (LIU et al., 2020; XU et al., 2020).

Consequently, adapting suitability criteria to the DSP context involves incorporating metrics considering data variability, processing latency, and system scalability. This includes data coverage metrics that measure the diversity of data items processed and the completeness

of scenarios covered, including handling data biases and anomalies. Latency measurements should be employed to verify that the system meets real-time processing requirements under varying load conditions. Test adequacy criteria for DSP must also consider the system's state behaviour and its ability to maintain correctness over time, assessing its resilience to changes in data patterns and its capability to recover or adapt without degrading system performance. Techniques like mutation testing, which simulates potential variations in the data stream, can provide insights into system robustness (NETO et al., 2022).

Determining when to stop testing in the DSP context requires adapting conventional fitness metrics approaches. DSP's dynamic, continuous processing, real-time nature, combined with scalability and state management challenges, necessitates a set of criteria focused on the system's ability to handle these characteristics and maintain operational integrity. Continuous testing becomes an integral part of the DSP system development and maintenance flow, employing a combination of static analysis to ensure code quality and dynamic monitoring to evaluate system behaviour under real-world data conditions.

2.4.3 Test Selection Criteria

Test case selection aims to identify which test cases to execute to minimize the use of testing resources while ensuring the most comprehensive evaluation possible of the system under test (HERZIG et al., 2015). Typical test selection criteria include code and path coverage, requirements coverage, fault detection potential, and test execution cost. Furthermore, in the context of regression testing, these criteria expand to consider the connections between architectural modules and the mapping of features that may have been affected by changes (BISWAS et al., 2011). Additionally, the criteria encompass risk assessment and the criticality of features to the business, thus directing testing efforts towards components that significantly influence system functionality and user satisfaction (KHANDAI; ACHARYA; MOHAPATRA, 2012; SRIKANTH; HETTIARACHCHI; DO, 2016).

However, in the context of large, complex, and distributed systems, such as the DSPs discussed in this thesis, the challenge of deciding which set of test cases to perform in each situation is magnified due to the huge number and variety of possible test cases. The survey of Sadri-Moshkenani et al. raises test case selection and prioritization approaches in the context of cyber-physical systems, which can also be considered for the DSP context (SADRI-MOSHKENANI; BRADLEY; ROTHERMEL, 2022). Also, in this context, testing objectives are varied

and include functional correctness, performance and scalability, fault tolerance and recovery, and data quality and integrity. The prioritization of these objectives is a crucial part of the test case selection strategy, tailored to the specific context of each application. In the literature, we find efforts to improve the selection of test cases, such as the proposal by Olsthoorn and Panichella based on multi-objective evolutionary algorithms (OLSTHOORN; PANICHELLA, 2021). Therefore, a thorough analysis of risk and impact on the business is essential in test case selection, as data streams often play a critical role in companies' core operations.

Performance parameters in DSP applications are notably sensitive to changes, whether in the application code or the configurations of the stream processing structure, an issue that becomes more prominent in complex applications involving multiple data stream processing steps. This complexity and sensitivity to changes underscore the importance of change traceability and a thorough understanding of data flow in DSP systems (ZVARA et al., 2019). Data undergoes several transformative processing stages, and changes at any stage in the processing pipeline can trigger cascading effects across components (DIVÁN; REYNOSO, 2020). Thus, a comprehensive mapping of data flows becomes indispensable for effective test case selection, facilitating targeted and efficient testing. It also ensures system modifications do not inadvertently compromise its integrity or performance.

Furthermore, economic considerations regarding the cost of running tests play a significant role in the context of DSP systems. The infrastructure for stream processing can be quite costly, and testing expenses can escalate, especially in the cloud, due to hardware allocation for real-time processing (SOUZA et al., 2020). These costs become especially pronounced during system-level performance tests that manage large cloud data volumes and involve paid third-party services. Therefore, understanding and managing the economic aspects of test selection is crucial.

2.5 STATE-OF-THE-ART IN DATA STREAM TESTING

Although there are no literature reviews or exploratory studies on testing practices in the DSP context, existing works focus on individual tools and approaches and some studies on testing and data stream software quality. In the following section, several research works and their contributions to the field have been explored.

Kallas et al. (KALLAS et al., 2020) proposed DiffStream, a differential testing library for Apache Flink, implemented in Java. It can be used with common testing frameworks, such as

JUnit, or as a stand-alone tool for Flink programs. The tool focuses on differential output testing for distributed DSP systems, verifying whether two implementations produce equivalent output streams in response to the same input stream. DiffStream serves various purposes, including checking for bugs in MapReduce programs, supporting the development of applications that are hard to parallelize, and monitoring DSP applications with minimal performance impact. The effectiveness and usability of the proposed test framework were evaluated by conducting four case studies: *Taxi Distance*, where it detected logic errors in calculating taxi routes using GPS data; *Topic Count*, which identified concurrency issues in counting topics in text streams; *Real-World MapReduce Programs*, revealing functional mismatches in MapReduce transformations adapted for streaming; and *Performance for Online Monitoring*, demonstrating minimal performance overhead, making it suitable for real-time application monitoring. These studies underscore DiffStream's versatility in uncovering bugs and concurrency problems, as well as its applicability in performance evaluation for distributed stream processing systems, confirming its utility across various testing scenarios. DiffStream is available on GitHub as an open-source tool[1].

Other works propose to apply property-based testing techniques to DSP application testing. This approach focuses on generating test cases based on properties defined by a set of Boolean expressions. Property-based testing checks if a system under test abides by a property. A significant benefit of applying this approach in the DSP context is greater coverage of test cases. Espinosa et al. (ESPINOSA et al., 2019) introduced FlinkCheck, the property-based testing tool for Apache Flink. FlinkCheck provides a bounded temporal logic for generating function inputs and stating properties. This tool randomly generates a specified number of finite input stream prefixes with the required structure and evaluates the Flink runtime's output streams. In this context, Riesco and Rodríguez-Horatlá (RIESCO; RODRÍGUEZ-HORTALÁ, 2019) proposed a combination of temporal logic and property-based testing to DSP applications testing. An Application Programming Interface (API) for testing Spark Streaming programs was made available, and it allows the writing of tests in a Scala functional language with the ScalaCheck library.

The work by (ZVARA et al., 2017) presented a holistic tracing framework for batch and streaming systems, which may be used to debug DSP applications by tracing individual input records. Thus, issues caused by outliers become traceable in complex topologies. They built a prototype implementation for an open-source data processing engine, Apache Spark.

[1] https://github.com/fniksic/diffstream

The evaluation indicated that the tool helps to identify a variety of common problems in real-time (e.g., bottlenecks, irregular characteristics of incoming data, anomalies propagated unexpectedly through the system). It can also assist developers improve system performance by detecting inefficiencies in the computing topology and reducing latency. This approach is valuable to prevent developers from spending countless hours identifying these problems manually, but it only works for problems in real-time running environments.

Restream (SCHLEIER-SMITH; KROGEN; HELLERSTEIN, 2016) is a tool that can replay streams from historical event logs. The tool is designed for accelerated replay and parallel processing, preserving sequential semantics. ReStream was compared against multiple implementations built on Apache Spark and outperformed them all, surpassing the single-threaded implementation and exceeding the performance of Spark. Although ReStream simulates streams precisely in a controlled environment, there is a lack of features to simulate fault tolerance scenarios. On the contrary, it handles fault tolerance cases by avoiding them during the simulation, for instance ignoring duplicated messages and resending messages that may have been lost.

Regarding the replay of historical data, there is the work by Gu et al. (GU et al., 2018), which also proposes a specific tool, Penguin, for data stream replay. However, Penguin was mainly designed for analytical purposes in scenarios where some portions of the extensive historical data need to be processed in a specified replay order (e.g., ordered by the timestamps). A notable feature of this tool is that it comes with specific operators to configure replay characteristics, for example, replay speed, message order, cache usage, message selection filters, merge messages, and map operation to perform a function on each record to generate a new message. These features may be helpful to test environments as they may be able to modify replay characteristics and approximate the simulation of the production environment or to specific test scenarios desired.

The work by Xu et al. (XU et al., 2013) discusses the validation and detection of abnormality in data streams by an analytical model, which can be defined as a function of sensor measurements. They implemented two general strategies to validate streams from industrial equipment: model-and-validate and learn-and-validate. The experiment shows that both approaches apply to real industrial data from IoT devices. However, these approaches may involve scalability challenges. Also, Pizonka et al. (PIZONKA; KEHRER; WEIDLICH, 2018) stated that data quality plays a pivotal role in many IoT applications, which demands continuous monitoring and validation of streaming data. They proposed VortoFlow, a tool that enables users to capture validity requirements regarding value ranges as part of an information model and

express it with a Domain-Specific Language. They demonstrated the general feasibility and applicability of VortoFlow using the case of a weatherboard.

Other works like (FEILER, 2010) also explored model-based validation of stream data in the context of IoT, and indicate data quality as one of the root causes bugs in IoT DSP. Multiple components are involved in handling the data stream, and they all can affect it by inserting noise and data distortion. They pointed out that the most recurring sources of problems are time sensitivity, data format, out-of-range values limits, concurrency, latency, and missing elements.

Due to the amount of information that flows through a data stream, data test generation is a relevant topic for testing DSP applications. In some situations, real historical data is unavailable, and the synthetic data generation can be the only way to have test data. It is necessary to generate many events with specific structures and values to test these systems' functionalities. The proposed tool by Gutiérrez-Madroñal et al. (GUTIÉRREZ-MADROÑAL; MEDINA-BULO; DOMÍNGUEZ-JIMÉNEZ, 2018), IoT–TEG, uses high-level languages, such as XML, for describing events patterns and rules for event test generation. It has controlled randomization to generate values within a set between a minimum and maximum value. Although this approach simplifies data generation by producing random data within defined ranges, generating data in patterns similar to real events would be interesting. For example, generating latitude and longitude data from a vehicle's Global Positioning System (GPS) in a logical sequence should represent automobile-compatible behaviour.

In this sense, a better data generation mechanism can be achieved through data preprocessing, statistical models and machine learning techniques. Iglesias et al. (IGLESIAS et al., 2020) introduced MDCStream, which provides a highly flexible mechanism to manipulate data distribution of real data stream to generate new synthetic data. The tool allows the configuration of several data characteristics, such as the range of dimensions, statistical distributions, randomisable seed, data dependencies and correlations, etc. Similarly, Komorniczak and Ksieniewicz (KOMORNICZAK; KSIENIEWICZ, 2022) have employed a one-dimensional interpolation technique to transform real-world datasets into customisable synthetic data streams.

Generating accurate test data streams is especially challenging in the presence of concept drift, which refers to slight updates to data concepts over time. Additionally, imbalances in the class distribution can further complicate the generation of representative test data. Regarding imbalanced data streams, techniques like the Synthetic Minority Oversampling Technique (SMOTE) come into play, as evidenced by works like Krawczyk et al. (KRAWCZYK, 2016) and

Bernardo and Bella (BERNARDO; VALLE, 2021). These works aim to generate synthetic data stream samples to address difficulties from imbalanced data streams. The potential of deep learning techniques in data augmentation is addressed by Iglesias et al. (IGLESIAS et al., 2023), who have systematically reviewed generative models such as Variational Autoencoder (VAE) and Generative Adversarial Network (GAN). These models can augment data by adding noise and permutations or crafting new synthetic datasets. Further endorsing the capacity of GANs, Zhang et al. (ZHANG et al., 2018) and Li et al. (LI et al., 2022) utilise GAN-based algorithms to create synthetic time series data, indicating the potential of such machine learning techniques in enhancing data samples for testing in DSP applications. Finally, Table 1 encapsulates each work's primary focus and main features to provide an overview of the discussed literature.

Work	Tool or Approach	Main Focus	Notes
(KALLAS et al., 2020)	DiffStream	Differential output testing for distributed DSP systems	Applications monitoring with minimal impact
(ESPINOSA et al., 2019), (RIESCO; RODRÍGUEZ-HORTALÁ, 2019)	Property-based Testing Tools	Property-based testing with temporal logic	Testing in Scala functional language
(ZVARA et al., 2017)	Tracing Framework	Debug DSP applications by tracing individual input records	Focuses on real-time problems
(SCHLEIER-SMITH; KROGEN; HELLERSTEIN, 2016)	ReStream	Replay streams from historical event logs	Lacks fault tolerance simulation
(GU et al., 2018)	Penguin	Replay stream data for analytical purposes	Allows modification of replay characteristics
(XU et al., 2013)	Validation Model	Validation and detection of abnormality in data streams	Scalability challenges
(PIZONKA; KEHRER; WEIDLICH, 2018)	VortoFlow	Capture validity requirements regarding value ranges	Domain-Specific Language
(FEILER, 2010)	Model-Based Validation	Explore data quality issues in IoT data streams	Identified root causes of system problems
(GUTIÉRREZ-MADROÑAL; MEDINA-BULO; DOMÍNGUEZ-JIMÉNEZ, 2018)	IoT–TEG	Data test generation for data stream software	Random data generation within ranges
(IGLESIAS et al., 2020)	MDCStream	Manipulate data distribution to generate synthetic data	Flexible configuration of data characteristics
(KOMORNICZAK; KSIENIEWICZ, 2022)	Interpolation Technique	Synthesise data streams using static datasets	Customizable with user-defined parameters
(KRAWCZYK, 2016), (BERNARDO; VALLE, 2021)	SMOTE	Generate synthetic data stream samples	Address imbalanced data streams
(IGLESIAS et al., 2023)	Deep Learning Generative Models	Increase data amount with augmentation techniques	Reviews VAE and GAN algorithms
(ZHANG et al., 2018), (LI et al., 2022)	GAN-based Algorithms	Creation of synthetic time series data	Synthetic data based on salient features distribution

Table 1 – Summary of Works on Data Stream Testing

2.6 SUMMARY

This chapter established the foundational concepts necessary for understanding DSP applications. Characterized by their high volume and velocity, DSP applications provide real-time

processing through continuous queries without prior storage, distinguishing them from traditional batch processing. The chapter also elucidated various DSP approaches highlighting each approach's trade-offs and use cases.

It introduced concepts like event time, processing time, and watermarks to address time-sensitive data challenges. Additionally, the chapter discussed the typical architecture of DSP systems, encompassing the roles of broker, stream processor, consumers and producers. It underscored the importance of fault tolerance, scalability, and state management in DSP applications.

The chapter acknowledged the unique challenges presented by the real-time architectural complexities, necessitating distinct strategies for testing. This encompassed ensuring data quality, managing stateful operations, and handling out-of-order events.

Finally, the chapter explored existing research in the field, specifically addressing test-related aspects. Seventeen works were briefly discussed and summarized in a table. It is worth noting that there was no comprehensive study covering DSP testing.

3 RESEARCH METHODS

This chapter presents the methods adopted to conduct this research and thus achieve the proposed objective. Figure 8 summarizes the methodology used to develop and evaluate the proposed testing guidelines for DSP applications. The process starts with recognizing the lack of formal DSP testing literature, the gap between academic and industry knowledge, and the scarcity of consolidated documents guiding the testing of DSP applications. Four sequential steps conduct the methodology: (1) An Exploratory Study gathering insights via 12 interviews and 101 surveys to confirm the topic's relevance for industry and practitioners' perspectives on the subject; (2) A GLR examining 154 grey sources to discern current testing practices, challenges, goals, data strategies, and adopted tools; (3) The development of an initial guidelines version (Guidelines V1) predicated on GLR findings and formal literature; and finally (4) The guidelines were evaluated through a 12-participant focus group and a 22-response survey, culminating in the refined Guidelines V2.

In the remainder of this chapter, we detail the methodology steps. Empirical studies from the investigation phase are carefully described in Section 3.1 (Qualitative Study) and Section 3.2 (GLR). Subsequently, the guidelines development methodology is presented in Section 3.3. Finally, Section 3.4 describes the evaluation process for the guidelines.

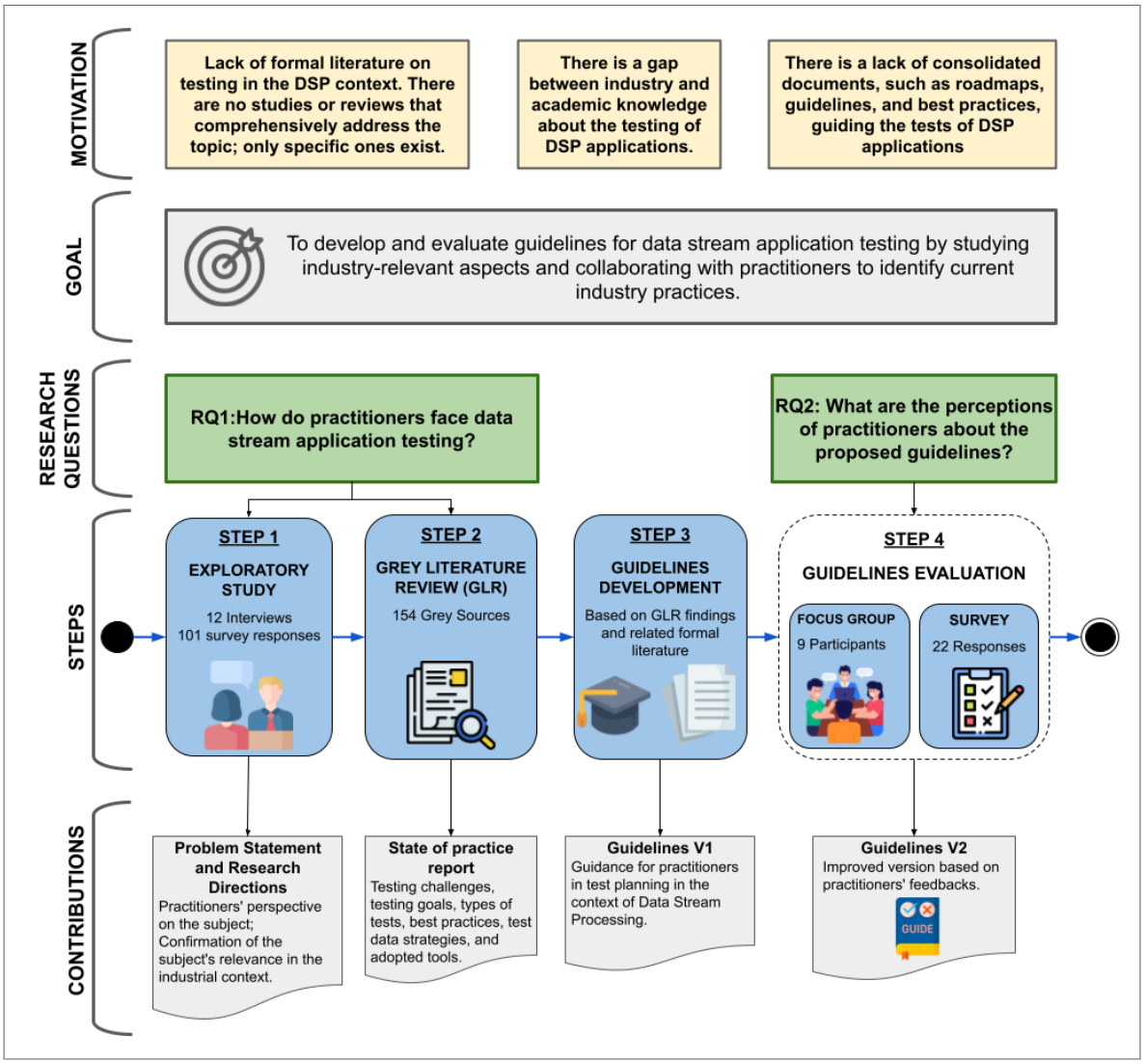


Figure 8 – Overview of Research Methods.

3.1 EXPLORATORY STUDY

Our objective at this step is to carry out an exploratory study in order to improve the understanding of the problem, map the state of practice, and identify the challenges related to DSP application testing. For this purpose, we administered a questionnaire and conducted interviews. The use of multiple data collection methods, such as questionnaires and interviews, allows data triangulation in order to support the validity of the results. While surveys are valuable for extensively capturing quantitative data, semi-structured interviews add depth, context, and qualitative data to the research (GHAZI et al., 2018). By complementing surveys with interviews, we aim for a comprehensive and multi-faceted understanding of the study's topic.

We chose grounded theory as the research method due to its capacity to construct theories

from empirical data. According to Glaser (CHARMAZ; BELGRAVE, 2007), grounded theory is recommended when a researcher asks: "What is going on here?". This questioning is aligned with the explorative nature of our study. Grounded theory was introduced in 1967 with the publication of Glaser and Strauss (GLASER; STRAUSS, 2017). The method later evolved into two versions with separate terminology and processes as Glaser and Strauss developed different perspectives (GOULDING, 1998). These two versions are known as the Glaserian and Straussian grounded theory methods.

Moreover, some modifications and variations have been proposed to the method, coming in at least seven different versions (DENZIN; BRYANT; CHARMAZ, 2007). We decided to follow classical Glaser's method since we wanted to approach the field with no research questions but rather a general interest in it. Also, we wanted to let the concepts and categories emerge from the data. We carried out our research using existing guidelines on conducting a Grounded Theory research (STOL; RALPH; FITZGERALD, 2016).

3.1.1 Questionnaire

Aiming at a broader audience and quantitative data, we administered a questionnaire with general statements about data streams and specific questions on DSP application testing (Appendix A). The statements were of four types: (1) closed-ended single choice questions; (2) closed-ended multiple choice questions; (3) questions about data scored on a 5-point Likert scale response format from strongly disagree to strongly agree; and (4) open-ended questions which allowed respondents to provide more in-depth and nuanced answers. The questionnaire development received support from two other experienced researchers who contributed by reviewing the questionnaire and offering suggestions for enhancements. One of these contributors is a master's student with two years of research experience in the field, while the other is a PhD professor with over ten years of research expertise in the area. In order to conclude the questionnaire preparation, a pilot application study was carried out with a researcher specialized in the DSP field. The pilot aimed to verify the question's relevance and significance, validating the questionnaire while addressing potential ambiguities or biases to enhance its reliability in capturing genuine aspects of the field.

Over nearly four months (from November 2018 to February 2019), we strategically disseminated the questionnaire among technical communities associated with Data Stream Processing (DSP), as well as directly to professionals who listed DSP experience on their curriculum on

LinkedIn. This targeted approach ensured our outreach was focused on individuals with at least one year of experience working in the DSP field, aligning with our requirement for participation. From discussion lists to social media forums, we aimed to engage a knowledgeable and experienced audience relevant to our study. The decision to stop responses at 100 was made as we began receiving fewer replies over time, with each new response adding minimal relevant content. This pattern suggested that additional data collection might only reiterate insights already acquired (JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008). Initially, we received 105 responses, but after a thorough review, we excluded four responses that failed to meet our criteria for serious and thoughtful participation, characterized by excessive one-word answers, nonsensical replies, and uniform responses to Likert-scale items. Consequently, our dataset was finalized with 101 valid responses, emphasizing the specific nature of our respondent pool where participation was voluntary and relevant DSP experience was a prerequisite.

3.1.2 Interview

Collecting data via interviews provides flexibility, enabling the interviewer to investigate intriguing or unexpected responses further, uncovering nuances linked to the participants' context. Additionally, it allows seeking for clarifications or justifications.

The sample universe comprises software developers with one year of experience working with DSP applications from any nationality as long as they communicate in Portuguese or English. A year in the field allows professionals to comprehend the main characteristics and nuances of the DSP context, going beyond foundational knowledge to encompass advanced scenarios—increasing the richness and applicability of our findings. While we mandated a minimum of one year of experience, we did not set a higher threshold to avoid excessively narrowing our sample pool.

Our semi-structured interview guide consists of five sections, which can be found in Appendix B. In Section B.1, we started the interview by introducing ourselves and providing a brief overview of the study, ensuring interviewees understand their anonymity and the consent terms. Section B.2 covers the interviewee's profile and background, gathering details about their education, general IT experiences, specific DSP experience and professional career. In Section B.3, we aim to discover their hands-on experience with DSP application development, highlighting projects that involve DSP and the strategies their organizations employ to handle DSP, including tools, mechanisms, processes, and more. Section B.4 addresses questions

about DSP application testing practices, exploring faced challenges in this context. Section B.5 focuses on testing tools and their features. To conclude, we allowed the interviewees to ask questions and encouraged them to share general comments or add pertinent information not covered in the interview.

For recruitment, we used the university's contact network, encompassing professors and alumni, and also conducted manual searches on professional social networking platforms, resulting in 12 participants. Each interviewee signed a consent form granting permission for the recording of their interviews and the anonymized publication of the research results. Interviews were conducted from December 2018 to February 2019 via Skype, lasting between 41 and 54 minutes, with an average duration of 48 minutes.

3.1.3 Data Analysis

The primary data analysis technique employed was coding (SALDAÑA, 2015). The adopted guidelines organize the procedures into three steps: Open Coding Data Collection, Selective Coding Data Analysis, and Theoretical Coding.

In the first step, we systematically read and coded all interview transcripts to identify initial categories. Following the open coding method (GLASER, 1992), we began without any preconceived codes. Every relevant sentence in each transcript generated a new code. This approach was necessary as, at this preliminary phase, it was unclear which data would prove pertinent.

During the second step, we clustered the initial codes by comparing new incidents with previous codes. As the coding progressed, some clusters emerged, each becoming a category (SALDAÑA, 2015). A fundamental objective here was to identify the core category, facilitating integration with the other categories. All relationships between categories must be based on this core category to imply a consolidated grounded theory (GLASER, 1992). We employed a constant comparative method and contrasted codes/categories within the same interview. Subsequently, comparisons were made between one interview and others, delimiting coding to only those categories related to one—or occasionally multiple—core categories to establish a succinct theory. This step concludes when theoretical saturation is achieved when no new codes or categories emerge from the recent participants (LUZ; PINTO; BONIFÁCIO, 2018).

Simultaneous to the second step, we matched our findings (categories and relationships) with responses to the open-ended questions from the questionnaire. Only passages that confirm

or contrast with interview findings were coded. This data triangulation aims to minimize research bias, since using only one data source is a threat, considering that people's reports in an interview are not always consistent with reality. A classic instance of this is the tendency of individuals to underestimate the time needed to accomplish tasks (KRUGER; EVANS, 2004).

In the last step, post-saturation, a theory is formulated to explain categories and their relationships. An essential task in this step is to reintegrate the proposed theory with the existing literature on the data stream. This step establishes conceptual relations between discovered codes and categories with insights and theories in the academic literature. Ultimately, we proposed a cohesive and homogeneous theory, emphasizing the roles of categories as enablers and results.

3.2 GREY LITERATURE REVIEW

The GLR is suitable for exploring complex topics that lack consolidated formal literature and also for investigating emerging issues, especially those currently evolving in the industrial context (GAROUSI; FELDERER; MäNTYLä, 2019). Kamei et al. (KAMEI et al., 2020) discovered that Brazilian software engineering researchers mainly use grey literature to understand new topics and seek answers to practical and technical questions. The practitioner community also shows high interest in this topic, as demonstrated by a previously exploratory study (VIANNA; FERREIRA; GAMA, 2019). Given these observations, the GLR is a suitable approach to continue our study with a deeper investigation of the issues that emerged during the exploratory study. Therefore, we conducted a GLR to select, analyze, and synthesize the relevant knowledge and pertinent expertise developed by practitioners working in the context of DSP application testing. We are especially interested in answering questions derived from the central research question, RQ1.

RQ1.1 What are the challenges to DSP application testing?

RQ1.2 What are the testing purposes?

RQ1.3 What are the approaches and specific types of tests performed?

RQ1.4 What are the strategies adopted by practitioners to obtain testing data?

RQ1.5 What are the tools, and under what circumstances are they used in DSP application testing?

We elaborated our GLR protocol based on the guidelines of Garousi et al. (GAROUSI; FELDERER; MäNTYLä, 2019). Figure 9 presents an overview of our GLR process, which has four main phases: 1) review planning, 2) search process, 3) source selection and 4) data extraction and data synthesis. In the following sections, we describe the protocol phases. For replication purposes, the data used in this study is available online at: <https://doi.org/10.6084/m9.figshare.22259539>

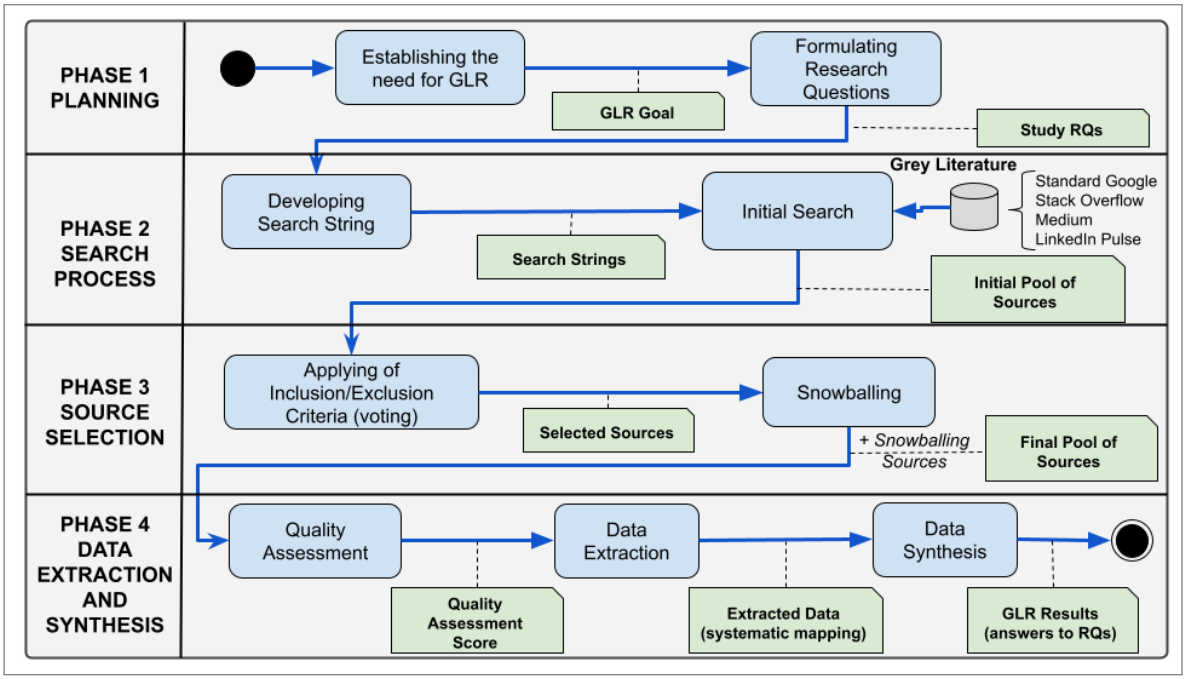


Figure 9 – GLR Protocol

3.2.1 Search process

The development of search strings is the first activity of the search process. We started including terms related to the research questions and then refined the strings by performing several tests and tuning in order to obtain more relevant results. Ultimately, we reached five search strings, each associated with an RQ. Table 2 shows the search strings.

We employed searches using the Google search engine, targeted searches on IT-specialized websites and supplemented our findings with a snowballing step. This strategy is well described in the guidelines provided by Garousi et al. (GAROUSI; FELDERER; MäNTYLä, 2019). The adoption of similar strategies has been experienced and discussed by other GLR works, such as (GODIN et al., 2015), (MAHOOD; EERD; IRVIN, 2014), and (ADAMS et al., 2016). Several reviews, including (TOM; AURUM; VIDGEN, 2013), (GAROUSI; MÄNTYLÄ, 2016), (KULESOVS,

Table 2 – Search Strings.

Associated RQ	Search String
RQ1	(challenges OR difficulties) AND (“data stream” OR datastream) AND (test OR testing)
RQ2	(purposes OR objective OR goal) AND (“data stream” OR datastream) AND (test OR testing)
RQ3	(“testing approach” OR “testing strategy” OR “unit test” OR “integration test” OR “system test” OR “acceptance test” OR “functional test” OR “load test” OR “performance test”) AND (“data stream” OR datastream) AND (test OR testing)
RQ4	(“testing data” OR “test data”) AND (“data stream” OR datastream) AND (test OR testing)
RQ5	(“testing framework” OR “testing tool” OR “testing library”) AND (“data stream” OR datastream) AND (test OR testing)

2015) and (MÄNTYLÄ; SMOLANDER, 2016), mainly used Google Search Engine for grey literature searches.

Although Google can identify key relevant sources, it often returns much irrelevant content due to its ranking system, which considers various factors that may not align with our research aims (GOOGLE, 2021; FU; LIN; TSAI, 2006). In line with recommendations by Kamei et al. (KAMEI et al., 2021), we expanded our search to target IT-specialised websites. Even though regular Google searches turn up results on these sites, only some of their pages are ranked for the final pool of results, making targeted searches on these sites necessary. We discovered that conducting standardised searches across multiple specialised websites was challenging, as they lack uniform logical operators for constructing search strings. As a result, we employed Google's advanced search engine to target specialised website URLs.

Therefore, we selected three popular IT-specialized websites among software engineering practitioners:

(1) Medium [1]: A platform where professionals share articles about experiences and technical challenges.

(2) Stack Overflow [2]: A popular Q&A website for IT professionals, hosting discussions on both practical and theoretical programming topics.

[1] <www.medium.com>
[2] <www.stackoverflow.com/questions>

(3) LinkedIn Pulse [3]: A blog platform within LinkedIn where professionals and organizations publish articles about their technical projects.

To streamline the search process, we employed two Python scripts: one for conducting searches on the Google search engine (Appendix G) and another for converting search results, specifically web pages, into PDF format for simplified review and archival purposes (Appendix H). The methodology involves initially executing the five predefined search strings via Google's standard configuration. Subsequently, these search strings are rerun on Google, but with the searches confined to each of the designated IT-specialized websites. The source codes are accessible in a public online repository[4].

An important note on using Google: while it may indicate finding millions of results, its relevance-based ranking system often provides only between 100 to 300 (GOOGLE, 2021). This limited output is what Google's algorithm considers most relevant. As a result, similar search queries can yield different outcomes due to this ranking system. Given this dynamic, we opted for five distinct search strings one per research question rather than a single complex string with multiple keywords.

3.2.2 Source Selection

This phase aimed to select the relevant sources that should proceed to the extraction and analysis phase. In this study, the term "source" represents a grey literature document, equivalent to a "study" in a systematic literature review. The selection begins by applying the inclusion and exclusion criteria. These criteria were initially developed based on Garousi's guidelines and subsequently refined through discussions among the researchers collaborating in the study (authors of the paper related to this chapter). Both the inclusion and exclusion criteria consist of questions that can be answered exclusively with "Yes" or "No". For a source to be included, it must be answered "Yes" for all inclusion criteria and "No" for all exclusion criteria. In this GLR, the exclusion criteria are the opposite of the inclusion criteria. Therefore, we present only the inclusion criteria in Table 3 along with justifications.

[3] <www.linkedin.com/pulse>

[4] Scripts for the search process can be found at <https://github.com/AlexandreSGV/glr_scripts>

Table 3 – List of inclusion criteria.

#	Criteria	Justification
IC1	**The source is essentially textual (may contain figures), and the text is in English and can be fully accessed.**	Choosing English as the primary language ensures a broader reach, as it is one of the dominant languages in the IT and academic world. Some content may be partially available for free, requiring payment to fully access it.
IC2	**The source is not duplicated or republished.**	Duplication or republishing can introduce bias. To identify similar textual content, we adopted the tool WCopyfind [5].
IC3	**The source is not an academic paper.**	As this is a GLR, it focuses on non-academic sources to capture industry knowledge. However, academic articles found should be saved for later discussions.
IC4	**The originality is not in doubt.**	Doubts about a source's originality risk introducing misinformation. We search for text excerpts on Google and check the author's other publications. The publication platform's feedback mechanisms (e.g., comments, likes) help estimate originality.
IC5	**The source was published between January 2010 and December 2020 (included).**	We focused on 2010-2020, marked by the evolution of modern stream processing and the emergence of major DSP systems (CARBONE et al., 2020).
IC6	**The source is related to DSP applications testing.**	The primary focus is on testing in the DSP context, so sources must relate to this domain.
IC7	**The source addresses some of the issues related to DSP application testing: challenges, tools, processes, approaches, best practices, guidelines, practical examples, tutorials, experience reports, and opinions.**	This criterion ensures that the review encloses content related to the research questions, including comprehensive insights ranging from theoretical discussions to hands-on experiences.

3.2.3 Data extraction and synthesis

3.2.3.1 Quality Assessment

The sources are evaluated through a quality assessment checklist based on the AACODS framework (TYNDALL, 2010), designed to enable evaluation and critical appraisal of such grey literature. It evaluates the source considering the following aspects: Authority, Accuracy, Coverage, Objectivity, Date, and Significance. The checklist's development process had the collaboration of a researcher specialized in GLR, and then it was submitted to a pilot study involving the other two collaborators. The checklist's application is blindly paired and, in cases of disagreement, a third evaluator participates in the decision. Table 4 presents the final

checklist.

As proposed in Garousi's guidelines, we included the source's impact aspect, which is composed of metrics about interactions with the source, such as the number of views, shares, and likes. We also collected specific metrics for software repositories, like the number of watchers, stars, and forks. Additionally, we also considered the source classification as a quality factor. GLR sources can be classified according to the outlet control, from the 1st tier with the most credibility to the 3rd tier with the least credibility (ADAMS; SMART; HUFF, 2017).

The quality assessment's final output is a numeric score between 0 and 1, summarizing the quality grade for each source. In order to calculate the quality score, each Yes/No response was scaled to values 1 and 0, while responses with numeric values were normalized to values between 0 and 1. However, we realized that, due to the significant differences between the obtained values, the simple normalization of values between 0 and 1 would make most values zero after rounding. It was necessary to carry out a distribution transformation to smooth out the significant discrepancies between the values. Therefore, we conducted a sensitivity analysis using different distribution functions and analyzed the goodness-of-fit of each distribution to our original data. The exponential distribution best fits because it smoothed high values while retaining the underlying distribution's characteristics. After transforming and normalizing the numerical values, we separately calculated the average score for each aspect of Table 4 and obtained the overall average.

The results of the quality assessment were used to prioritize sources, giving special attention to those with the highest scores. Such sources undergo a detailed and careful analysis, as they are more likely to be relevant. We also weigh the quality score when corroborating our findings. However, we did not use quality assessment scores to exclude any sources. This decision was made primarily to ensure we did not discard pertinent data. Even studies with lower-quality scores can contribute valuable insights or perspectives that might be absent in higher-scoring studies (YANG et al., 2021). Furthermore, we acknowledged the potential for selection bias despite safeguards like paired quality reviews, standardized reviewer training, and preliminary pilot studies.

3.2.3.2 Data Extraction

Data extraction is based on systematic mapping, presented in Table 5. It was prepared to structure the content extracted from the selected sources, and the results were then synthesized

Table 4 – Quality assessment checklist.

Aspect	Question	
Authority of the producer	▪ Is the organisation/author reputable? (in a general way) , E.g., the Software Engineering Institute, large IT companies, and well-known organizations or individuals in the field. ▪ Produced/published other work (grey or formal) in the field? (Data Stream)	
Accuracy	▪ Does the source have a clearly stated aim? ▪ How many references/links? Total ▪ Supported by authoritative, documented references or credible sources? (Grey literature or formal references)	
Coverage	▪ Does the source clearly addresses a specific RQ?	
Objectivity	▪ Is the statement a subjective opinion? ▪ Are the conclusions free of bias? Is there no vested interest? , E.g., a tool comparison by authors who are working for a particular tool vendor.	
Date	▪ Year ▪ Does the item have a clearly stated date related to content? ▪ How many contemporary references? Check the bibliography for contemporary material. (up to 5 years from source year)	
Significance	▪ Does it add context? (background) ▪ Does it enrich or add something unique to the research?	
Impact (Quantify interactions)	General websites: ▪ Views ▪ Likes, claps or upvotes ▪ Comments ▪ Backlinks (Distinct Refering Domains)	GitHub ▪ Watch ▪ Star ▪ Fork
Outlet type	▪ 1st tier Grey Literature (measure = 1): High outlet control/ High credibility: Bachelor/Master/Ph.D. thesis, Software/Tool Documentation, White Paper, Blog Post (in company website) ▪ 2nd tier Grey Literature (measure = 0.5): Moderate outlet control/ Moderate credibility: Blog Post (technical blog platform), Q&A Websites, Slide presentation, Tool Repository, Wiki Articles ▪ 3rd tier Grey Literature (measure = 0): Low outlet control/ Low credibility: Email Discussion List, Blog Post (individual), Tweets	

and reported. The systematic mapping relates the RQs (Column 1) with the corresponding attribute/aspect (Column 2) and a set of categories (Column 3). Column 4 indicates whether the category selection is Multiple (M) or Single (S) for the corresponding attribute/aspect. Developing the predefined set of categories was based on our past experiences with the qualitative study presented in Section 3.1. As the initial set of categories can restrict the findings, new categories that emerge from the sources through the conduction of open and axial coding will be included in the systematic mapping (HASHIMOV, 2015).

Table 5 – Systematic map.

RQ	Attribute/Aspect	Categories	(M)ultiple/ (S)ingle
-	Contribution type	Guidelines, Method (technique), Tool, Tutorial, Best Practice, Experience Reports, Philosophical & Opinion	S
RQ1.1	Issues and challenges	Data Privacy, Performance, Costs of Tests, Message Schema, Contracts Checking, Fault Tolerance, Exactly-Once Semantics, Concept Drift, Test Automation, Lack of Tools, Lack of Specialised People or Know-how, Latency	M
RQ1.2	Testing approaches	Load/Stress Test, Unit Test, Integration Tests, System Tests, Generative Test, Property-based Test, Regression Tests, Data Stream Simulation, End-to-End Tests	M
RQ1.3	Tools	Testing Frameworks, Data Generators, Simulation Tools, Data Replay Tools, Mocking tools, Contract Testing Tool	M
RQ1.4	Test purposes and goals	Correctness of Results, Bug detection, Performance, Fault Tolerance	M
RQ1.5	Test data	Historical Data, Synthetic Data, Sampling Production Data, Custom Data Set	M

The extraction process is carried out with peer review. The extraction tool adopted is a Google Docs Spreadsheet that performs the systematic mapping from Table 5. The researchers carefully analyze the source, marking the categories related to it. For each marked category, the researcher must incorporate an annotation with the text copied from the original source representing the category. Annotations are traceability links between the extracted data and primary sources. Additionally, the researcher may include comments on the source that would be useful in the synthesis phase. After the extraction, a systematic comparison of the pair's results is performed. Disagreement cases should be resolved by discussion.

3.2.3.3 Data Synthesis

The synthesis approach is based on qualitative methods with both quantitative and qualitative data analysis techniques. We adopted a qualitative analysis based on open, axial, and selective coding (SALDAÑA, 2015). The first step is open coding, in which textual data is associated with codes; then, axial coding explores the relationships among codes; and finally, there is selective coding, where codes are grouped into core categories. This coding process is guided by the pre-defined categories in the systematic mapping. After collecting the data, we will be able to structure the categories better and synthesize the evidence found in the coding with a focus on answering the research questions.

On the other hand, in the quantitative analysis, we evaluate the most frequent codes related to research questions in order to get more findings, such as the most cited test tools, the most recurring problems, the most discussed challenges, and common test purposes. Particularly, question/answer websites, like StackOverflow, enable quantitative analysis of the popularity of tools, problems, and recurring questions that software engineers face in DSP application testing. Finally, we perform a quantitative analysis of demographic data, such as the source authors' profile, the temporal distribution of the selected sources, the type of document, the most cited references and others.

3.3 GUIDELINES DEVELOPMENT

This thesis proposed guidelines for testing DSP applications, representing the main outcome of this study. These guidelines were developed primarily based on thoroughly synthesising relevant information from two preceding studies: the exploratory study and the GLR. These

studies provided an extensive data source on testing practices specific to the DSP context, serving as a substantial foundation for creating a comprehensive testing guide for DSP applications.

In formulating these guidelines, we integrated recommendations from related academic works predominantly identified during the GLR. We also looked at studies regarding the testing field that engaged directly with practitioners as primary sources, bridging knowledge between industrial applications and academic research. Integrating academic insights with practical findings ensures that the guidelines are scientifically robust and grounded in real-world practices.

To design our document, we analysed existing guidelines from academic literature and practical models used in the industry, such as: The Twelve-Factor App [6], OWASP Testing Guide [7], Unit Testing Guidelines [8], The Principles of OOD [9], Guidelines for Software Development [10], Atlassian REST API design guidelines version [11], and Zalando RESTful API and Event Guidelines [12]. This review aided us in developing a format for our guidelines that is structured, user-friendly, and accessible, catering to the needs of both practitioners and researchers. Our goal was to create a document that is not only informative but also easily comprehensible and applicable in practical settings. The guidelines underwent evaluation through focus groups and surveys (Section 3.4). Feedback from these evaluations was employed in refining and enhancing the document, verifying its perceived relevance and applicability.

Finally, with the intent of contributing back to the industry, which contributed to this research, and to ensure broad dissemination among professionals, we prepared a web format version of the guidelines (<http://datastreamtesting.space>) targeted at practitioners. Additionally, we created a condensed one-page version in cheat sheet format (Appendix J), available as an image or PDF. This concise version outlines the guidelines to their most essential elements, making them suitable for easy web publishing and social media sharing. This approach aligns with the findings of Florea and Stray's study (FLOREA; STRAY, 2020), which underscored the preference of software testing practitioners for learning from informal online sources.

[6] <12factor.net/>
[7] <owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf>
[8] <geosoft.no/unittesting.html>
[9] <http://butunclebob.com/FitNesse.UserGuide>
[10] <www.cyber.gov.au/resources-business-and-government/essential-cyber-security/ism/cyber-security-guidelines/guidelines-software-development>
[11] <developer.atlassian.com/server/framework/atlassian-sdk/atlassian-rest-api-design-guidelines-version-1/>
[12] <opensource.zalando.com/restful-api-guidelines/>

3.4 EVALUATION

The proposed guidelines must be evaluated to check their benefits and overall quality. Flawed guidelines could exacerbate the challenges of DSP application testing (JEDLITSCHKA; PFAHL, 2005). Our evaluation addresses the research question RQ2, which seeks to understand practitioners' perceptions of the proposed guidelines regarding applicability, strengths, and potential adverse effects. An essential aspect of this evaluation is to verify that the guidelines document is clear, self-explanatory, and easily comprehensible.

To this end, we proposed an evaluation from the perspective of experienced practitioners in the DSP field. Our methodology is inspired by reading first the document and then the theoretical evaluation of guidelines discussed in (KITCHENHAM et al., 2008). We employed a triangulated approach to mitigate potential biases inherent in single-method feedback collection, integrating insights from both focus group discussions and survey responses from distinct participant groups (THURMOND, 2001). Triangulation enhances data reliability and validity by cross-checking information from multiple sources, thus reducing the likelihood of skewed results. Ammenwerth et al. study indicate how triangulation can support both the validation and completeness of results (AMMENWERTH; ILLER; MANSMANN, 2003).

The focus group method gathers qualitative data and insights from a selected group of individuals on a specific topic (SEAMAN, 1999; DILSHAD; LATIF, 2013). We consider this method appropriate to evaluate our proposed test guidelines, as it enables participants to provide comprehensive insights through interactive discussions. The collaborative nature of focus groups can lead to the generation of new ideas and solutions that may not emerge in individual interviews or surveys. The structure of our focus group study is based on the proposal by Morgan et al. (MORGAN; KRUEGER; KING, 1998), complemented by guidelines and best practices from Nyumba et al. (NYUMBA et al., 2018) and Breen (BREEN, 2006).

Surveys, as an empirical method, are frequently used in software engineering to collect data from a broader population (GHAZI et al., 2018). This method is suitable for evaluating the proposed guidelines, allowing participants to share their impressions, opinions, criticisms and experiences. These insights provide a valuable complement to the qualitative feedback obtained from focus groups. We guided our survey design by the literature review on guidelines for conducting surveys, which emphasised the importance of careful planning, executing, and analysing of the data (MOLLÉRI; PETERSEN; MENDES, 2016).

Figure 10 illustrates the four main stages of our evaluation process, which include both

focus group and survey methods: (1) planning, (2) research design, (3) data collection, and (4) data analysis and synthesis. The activities represented in yellow apply to both components of the evaluation. The blue boxes denote steps specific to the focus group methodology, while the red boxes correspond to the survey component. This visual schema delineates the progression from initial objectives to reporting findings.

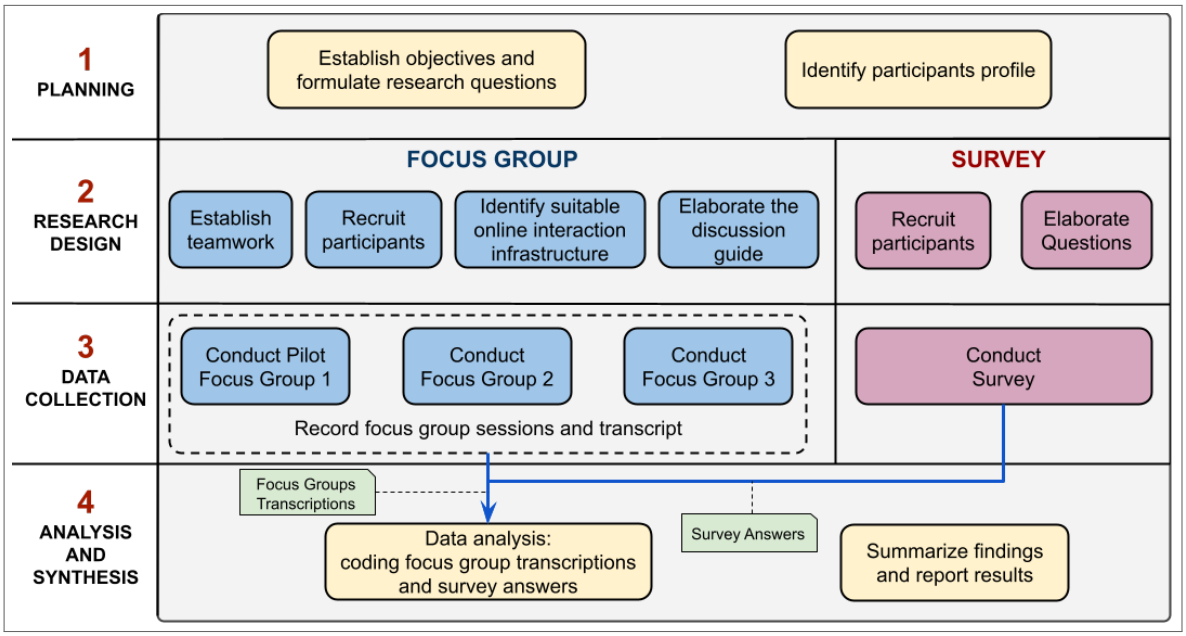


Figure 10 – Evaluation Process

3.4.1 Planning

3.4.1.1 Objective

The evaluation primarily addresses RQ2: *'What are the practitioners' perceptions of the proposed guidelines?'* Our central aim is to determine if practitioners find the proposed guidelines effective in assisting decision-making processes for testing DSP applications. Specifically, we seek to identify potential strengths, weaknesses and areas for enhancement, and assess their applicability in industrial contexts. The feedback from the focus group will be instrumental in revising these guidelines. Accordingly, we have formulated the following derived research questions to guide our inquiry:

RQ2.1 What are the perceived strengths of the guidelines?

RQ2.2 What are the identifiable weaknesses or areas for improvement in the guidelines?

RQ2.3 How do practitioners perceive the applicability of the guidelines in an industrial context?

3.4.1.2 Participants Profile

For both the focus group and the survey in our studies, participants were required to have a minimum of one year's experience working with DSP applications. This criterion aligns with the experience level of practitioners who participated in the interviews and questionnaire of the exploratory study, as detailed in Section 3.1. By ensuring a consistent professional profile among participants across all phases of research, we seek to validate the guidelines with a group that mirrors the profile of those who contributed to the initial information-gathering stage.

For the focus group, we intentionally recruited a different set of professionals for the assessment phase than those involved in the interviews of the exploratory study. This approach was adopted to gather fresh, unbiased perspectives and increase the results' validity. By diversifying the pool of evaluating practitioners, we aimed to capture a broader range of knowledge and experience, ensuring that validations were not biased by being overly tailored to the views of the initial group. This methodological choice aligns with standard research practices to reduce bias and improve the generalization of the proposed guidelines (ROMANO et al., 2021). Regarding survey participants, we did not impose this restriction as we could not guarantee control of this variable due to the recruitment strategy.

3.4.2 Research Design

Below, we bring the research design details into two sections for the focus group and survey studies, respectively.

3.4.2.1 Focus Group

Establish Teamwork: The focus group session was conducted by a collaborative team composed of a moderator-researcher and an observer. The moderator-researcher facilitates participants' interactions and discussions while adhering to a predefined discussion script. Meanwhile, the observer is responsible for documenting and taking notes on verbal exchanges and non-

verbal interactions throughout the session. They also assist the facilitator in maintaining focus on the discussion and preventing interruptions. Moreover, the observer can support the facilitator by clarifying discussion topics and managing media materials on the Whiteboard. The author of the guidelines, Alexandre Vianna, played the role of moderator-researcher, while Kiev Gama played the observer role.

Participant Recruitment: Participants were identified via profile searches on the LinkedIn social network, targeting profiles that featured terms and technologies related to DSP. For each potential participant, we meticulously examined their profile, scrutinizing their experience, certifications, listed skills, portfolio, and engagement (posts and comments) pertinent to DSP. In our initial communication, we introduced the research project and extended an invitation to participate. Subsequent message exchanges were used to further confirm the candidates' qualifications and their fit for the study's requirements.

Online Environment: Due to the geographical dispersion of professionals, the focus groups were held online rather than in person. Stewart et al. (STEWART; SHAMDASANI, 2017) highlight that online focus groups are a well-established method in research, but they require adaptations to traditional approaches. We used the Miro Online Whiteboard(REALTIMEBOARD, 2023) as our collaborative platform. This tool is designed to facilitate interactive discussions and enable real-time collaboration. It allows participants to visually communicate, share ideas, and provide feedback through several features, such as text, images, shapes, notes, and comments. The platform's compatibility across different devices ensures that participants can easily access the focus group sessions from any location with Internet connectivity. Instructions and support were provided to all participants ahead of the sessions to minimize technical difficulties and ensure smooth participation.

Discussion Guide: The discussion guide conducted the conversation during the one-hour meeting. The discussion went through each of the guidelines, encouraging participants to share their thoughts and experiences related to the guidelines. Participants were asked to address issues related to the research questions. Throughout the discussion, participants were prompted about various aspects of the guidelines, such as clarity, comprehensiveness, relevance, and applicability. The Miro Online board (REALTIMEBOARD, 2023) tool was used to expose the pertinent excerpts, tables, and figures from the guidelines document, fostering dis-

cussion on crucial points while enabling the collection of observations, ideas, and suggestions through interactive visual mechanisms. In Appendix C, the script prepared for the focus group is presented, considering the evaluation objectives to be achieved. Note that the time allocated to each section was approximate and was adapted to suit the flow of the discussion. The facilitator managed the conversation to ensure all research questions were addressed within the 60-minute timeframe.

3.4.2.2 Survey

Participant Recruitment: The recruitment process for the survey followed a similar approach taken for the focus group, with a broader outreach. We extended our search beyond LinkedIn, sharing the survey through targeted email lists and online discussion groups that are frequented by DSP professionals. Our communication strategy was more direct: we succinctly outlined the research context in our messages and invited individuals to participate. Over 500 personalized invitations were sent via LinkedIn direct messages. We also posted messages in six LinkedIn discussion groups and reached out through seven email lists dedicated to users of DSP-related tools. Below, Table 6 presents the email lists and discussion groups where the survey was disseminated.

Table 6 – Summary of Recruitment Channels for Survey

Channel Description	**e-mail address/URL**
Email Lists	
Apache Spark Users	user@spark.apache.org
Apache Samoa Developers	dev@samoa.incubator.apache.org
Apache Storm Users	user@storm.apache.org
MOA Community	moa-users@googlegroups.com
Apache Samza Developers	dev@samza.apache.org
Apache Kafka Users	users@kafka.apache.org
Apache Flink Users	user@flink.apache.org
LinkedIn Discussion Groups	
CEP & Algo Trading	<https://www.linkedin.com/groups/3892946/>
Apache Flink Group	<https://www.linkedin.com/groups/7414853/>
Streaming Analytics & CEP	<https://www.linkedin.com/groups/44123/>
KAFKA Data Streams	<https://www.linkedin.com/groups/10370853/>
AI & Data Science Community	<https://www.linkedin.com/groups/5096075/>
Stream Processing & Big Data	<https://www.linkedin.com/groups/6586282/>

Survey Questions: The survey questions are presented in Appendix D. Section 1 laid out the research context and presented the guidelines while establishing the minimum experience required for participant eligibility. Section 2 gathered demographic information in order to map the respondent's profile diversity. Sections 3 to 7 were dedicated to an in-depth assessment of each guideline, utilizing a relevance scale for participants to rate relevance and open-ended questions for qualitative feedback. This structure was designed to elicit detailed insights, including practical applications, suggestions, and critiques. The final section, Section 8, concluded the survey with an invitation for additional comments and an option for participants to receive updates on the research progress, thus fostering ongoing engagement with the professional community.

3.4.3 Data Collection

The data collection process comprised a pilot focus group session, which served as a rehearsal to refine our discussion guide, identify potential issues, and train the facilitator's skills. Participants in the pilot were exclusive to that phase and did not partake in the main focus group discussions. Following the pilot, two main focus group sessions were conducted.

All focus group sessions were audio-recorded with the explicit consent of the participants. The consent form they signed stated their understanding and agreement to the recording of the sessions. The interaction via the Miro Online Board captures real-time annotations, discussions, and collaborative interactions among participants, thus providing an additional layer of data.

The survey was disseminated to a broader participant pool via Google Forms in parallel with the focus group discussions. The recorded and transcribed focus group discussions, Miro Online Board annotations, and survey responses formed a comprehensive dataset for our study.

3.4.4 Analysis and Synthesis

This phase began with a detailed analysis of the focus group discussion transcripts, annotations from the Miro Online Board (REALTIMEBOARD, 2023), and responses to the open-ended survey questions. Open coding was meticulously applied to each qualitative data source to assign initial codes (GLASER, 1992; SALDAÑA, 2015). These initial codes were then examined for interconnections through axial coding. Clustering code in core categories facilitated the

identification of recurring themes and sentiments.

Subsequently, selective coding was undertaken, where these categories were further synthesized to form a comprehensive understanding of the practitioners' perspectives on the proposed guidelines. This synthesis was enriched by quantitative measures from the survey, such as the Likert scale responses regarding the participants' perceived relevance of the guideline items.

The culmination of this process is a report that encapsulates the practitioners' impressions of the guidelines, including strengths, weaknesses and areas for improvement, and applicability in an industrial setting. It highlights participants' consensus, discrepancies, and nuanced perspectives. The report will serve as a foundation for the refinement of the guidelines.

3.4.4.1 Threats to Validity

Given that the moderator was also the author of the guidelines, there is a potential threat to the validity of our findings. Participants might have felt inhibited or reluctant to highlight issues, errors, or inaccuracies in the guidelines due to the author's presence. This could compromise the candidness of their feedback. To counteract this potential bias, we distributed an anonymous questionnaire featuring a singular open-ended question. The intent was clarified:

> *"If, for any reason, you felt uneasy critiquing the guidelines during the focus group session, this form offers a platform for you to share opinions and criticisms anonymously, pinpointing any perceived problems, errors, or inaccuracies."*

3.5 SUMMARY

In this chapter, we presented the methodological approach adopted for the thesis:

- Exploratory Study: In this initial phase, interviews and surveys were conducted with industry practitioners to confirm the topic's relevance in the industry. The study aimed to gather insights from professionals to understand their perspectives and challenges in DSP testing.
- Grey Literature Review (GLR): This phase examined informal sources like blogs, whitepapers, and industry reports. The study aimed to identify current practices, challenges, strategies, and tools used in DSP testing.

- Development of Guidelines: The guidelines were developed based on insights from the GLR and related formal literature. These guidelines were designed to reflect the current best practices and address the challenges identified in earlier phases.
- Evaluation of Guidelines: The final phase involved evaluating the formulated guidelines through focus group discussions and surveys with industry practitioners. This step was crucial to refining the guidelines.

  Each step contributed to the overall goal of creating a valuable, practice-grounded set of guidelines for testing DSP applications.

4 EXPLORATORY STUDY

4.1 DEMOGRAPHICS

Before presenting the results from the collected data, some demographic data about the subjects is presented. Throughout the text, interviewees will be referenced as P[1-12] according to Table 11, while the questionnaire respondents will be mentioned as Q[1-101].

4.1.1 Survey Demographics

Among the 101 valid answers, 92 respondents were male and nine female. The questionnaire participants' profiles were quite varied in terms of workplace, experience, and company business sector. Table 7 shows the countries where the participants work. As it can be seen, there is a predominance of countries with large economies and consolidated software industries, such as the USA, India, Brazil, France and Germany.

Table 7 – Country where questionnaire respondents work.

Country	**Total respondents**
USA	27
India	11
Brazil	10
France	8
Germany	6
Italy, Spain, Sweden	4
Poland, Sri Lanka	3
Canada, Denmark, Netherlands, Singapore, Switzerland, Turkey	2
Australia, China, Colombia, Czech Republic, Dubai, England, Ireland, Kazakhstan, Nigeria, Portugal, Vietnam	1

Table 9 presents the company's business sectors in which the participants work. In this question it was allowed to mark more than one item. Industry, Business Intelligence and Financial Systems were the most prominent sectors, and the relationship between these fields and stream processing is evident. There was a significant number of participants involved with the research and development (R&D) of tools for stream processing. This number may be associated with the dissemination of the questionnaire in the communities of the users of

tools. An interesting aspect is that most of the respondents involved with R&D were from companies instead of academia. A possible interpretation of this result would be the emerging need for companies to invest in research and development of solutions in this context, meaning that DSP presents challenges that are not neglected.

Table 9 – Questionnaire Respondents Companies Business Sectors.

Company Sector	Quant.
Industry	46
Business Intelligence	46
Financial Systems (credit card, financial market, stock exchange)	42
Research and Development of Software and Technologies for processing data stream.	42
E-commerce	37
Digital marketing	29
Telecommunications	23
Social Networking Monitoring	19
Monitoring of equipment and infrastructures (oil pumps, oil pipelines, oil rigs, mining equipment, nuclear power plants and others)	18
Road Transport Monitoring Systems (Freight)	12
Power Distribution Systems (monitoring of transmission networks, electric stations, power plants and others)	12
Urban Transportation Systems	10
Air Traffic Control Systems	7
Maritime Traffic Control Systems	5

Table 10 shows the level of education of the participants. It is observed that 68 out of the 101 participants have master's or doctorate degrees. It is a high degree of qualification which may be associated with the profile of those who work with this technology. The involvement with the development of data stream tools and the participants' high level of training bring very relevant contributions to this work.

Table 10 – Questionnaire Respondents Education Level.

Highest Education Level	**Respondents**
Doctorate degree	20
Master's degree	48
Associate/Bachelor/Graduate degree	25
Trade/technical/vocational training	2
Some college credit, no degree	5
High school degree or equivalent	1

The graph in Figure 11 illustrates IT professional experience (upper blue bar) in years versus data stream experience time (lower red bar). It is interesting that 86 of the 101 participants are concentrated in the range of 1 to 7 years of data stream experience. This behaviour does not follow the overall IT experience in which 37 of the 101 participants have 1 to 7 years of experience. This concentration may be associated with the growth of the data stream field in recent years.

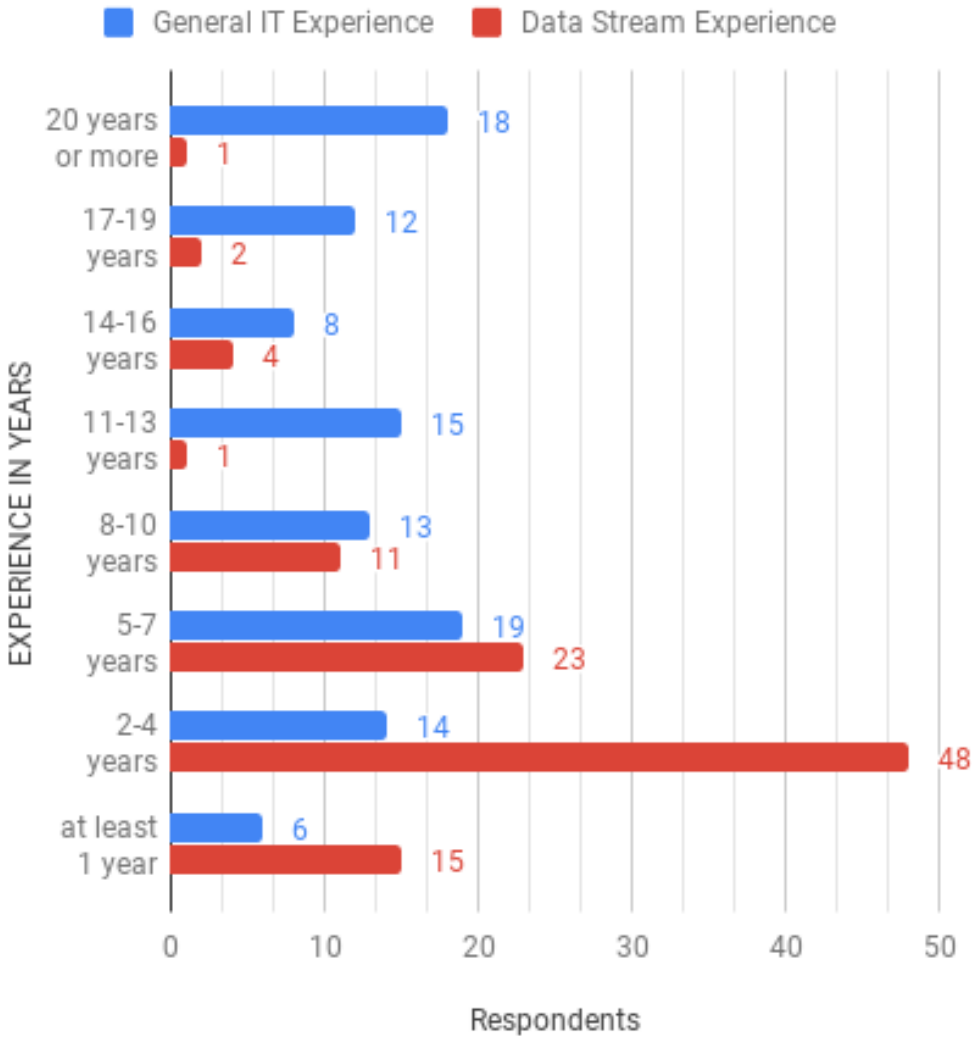


Figure 11 – General IT experience versus data stream experience of questionnaire respondents.

4.1.2 Interview Demographics

Table 11 summarizes the participant's profiles. We reached 12 participants with varied profiles in both time of experience and profile of the companies and projects in which they work. As for the professional experience, there are some participants with little experience time,

such as P5, P9 and P11, who have three years or less of professional performance. Although they have little experience time, they have enough to contribute by being involved in companies with very relevant cases in the data stream field. For example, participant P9 has one year of professional experience but works with stream processing in a company that delivers meals by applications; this company operates in Latin America, where more than 400,000 requests are processed daily.

Table 11 – Interview Participant Profile.

P#	Education Level	Job Title	ITX	DSX	Company Main Sector	#ITE	#DSP Developers
P1	Master	Backend Engineer	14	2	Financial (Benefits & Rewards Services)	8+	2
P2	Master	Software Architect	21	4	Software Consulting	2000+	7
P3	Bachelor	Software Engineer	8	2	Location Data Services	70+	6
P4	Bachelor	Software Engineer	9	3	Financial (Banking Services on App)	200+	100
P5	Bachelor	Software Engineer	3	2	Financial (Banking Services on App)	200+	100
P6	Bachelor	Big Data Engineer	16	4	Financial (Traditional Bank)	5000+	-
P7	Master (in progress)	Big Data Engineer	10	4	Financial (Investment Broker)	1600+	8
P8	Master	Software Engineer	6	3	Software Consulting	2000+	100
P9	Bachelor	Data Engineer	1	1	Food Delivery Service	250+	50
P10	PhD	Software Engineer	22	10	Solutions for Smart Cities	10+	5
P11	Bachelor	Data Engineer	3	2	Data Science Solutions	11+	3
P12	Bachelor	Software Developer	8	4	E-commerce	40+	6

ITX means IT experience in years, DSX means DSP applications development experience in years, ITE means the number of IT employees in the company, and DSP Developers refers to the estimated number of developers working in DSP context.

We also interviewed very experienced professionals, such as P2, P6 and P10, who have more than 15 years of professional performance. Participant P6 works with DSP in a large bank and has a vast background in the Big Data domain, having certifications in stream processing platforms, besides having worked in several companies in the financial sector. Participant P2 has 21 years of experience and currently works in an international software consultancy that serves a large bank. Also, he has a master's degree in which he researched and proposed design standards for event processing. P10 has a PhD degree and academic experience with stream processing, having a background in the development of cloud service platforms for mobile apps, and today is a co-founder of a start-up that develops solutions for smart cities.

Company profiles are also diverse, ranging from traditional large companies, some of them listed on top rankings from Fortune (FORTUNE, 2021)and Forbes (FORBES, 2021), with thousands of IT employees, to start-ups with few professionals who are developing innovative services involving DSP. This diverse spectrum allows us to capture different experiences and visions on the subject.

4.2 ANALYSIS ON SURVEY AND INTERVIEW DATA

As mentioned in Section 3.1, we analysed our data following some coding techniques. With these techniques, the core category *Testing Data Stream Software* emerged as well as three subcategories: *Testing Approaches*, *Testing Tools* and *Test Data*. In the next sections, we present each category.

4.2.1 The Core Category: Testing Data Stream Software

Testing emerged as a highlighted issue for DSP applications, with more than one hundred code references. For instance, a comment from interviewee P6 reveals the importance of quality assurance in a DSP of anti-fraud features in the financial sector. This is crucial because failures result in substantial losses in a short time.

> *"The anti-fraud feature processes the user interaction data stream. It checks in real-time against a previously trained model if these data fit the pattern of regular interaction or a fraudulent interaction. What is the cost of false negatives? It can be huge."* (P6, Data Engineer, Financial)

Participants were asked about test culture in their companies. In response, 10 out of 12 participants said that a test culture is present in their companies. On the other hand, two interviewees (P10 and P12) commented that testing is essential in their companies, but sometimes they give up automated testing. In such cases, their applications have complex infrastructures and the tests are manually performed by manipulating inputs from the system running in the acceptance environment.

> *"We have a test culture, but we are not fanatics of testing. When things start getting complicated, and we have to develop many systems to be able to test, then we end up not doing it. We run the system in the acceptance environment with real data and perform system-level testing."* (P10, Software Engineer, Smart Cities Sector)

> *"There is a strong test culture in the company. We have a robust and evolved test stack for the complete transactional part, which concerns the purchase, payment, and the entire checkout flow of the marketplace."* (P12, Software Developer, E-commerce Sector)

In some interviews, participants commented on specific issues that threat DSP application testing. The most frequent problems are message synchronization, multiple data streams operations, data heterogeneity, fault tolerance, difficulties on traceability and generating test data. Below, a response from Q-59 highlights message synchronization issues.

> *"Our data stream is immensely partitioned and duplicated. We have a complicated tumbling window with custom 'go-next' logic requiring various events. Stream synchronization and window tumbling is the biggest challenge."* (Q-59)

This category also emerged from questionnaire answers. When asked (question 32) about challenges in DSP, 14 of 73 valid responses mentioned topics related to software quality. For example, fault recovery, test data generation, ensure exactly-once semantics, and automated testing in continuous integration systems. Participant Q-45 explicitly mentioned a difficulty in testing DSP applications as well as the need of tools for this purpose.

> *"The testing aspect of stream processing is difficult, and there is not a great tool set for testing."* (Q-45)

Participants reported relevant aspects that hamper the development and testing of DSP applications. The quotation below synthesises these comments:

> *"The biggest challenge is scale. The problem is the data growth rate, so we reached a petabyte here, and it keeps growing. Then things start to get rough."* (P9, Data Engineer, Food Delivery Service)

There is a consensus on the need for testing automation, with 73% of survey respondents considering that tools to automate the testing process are valuable and desirable, and 10 of the respondents argue the importance of testing tools being embedded into continuous integration systems. This idea corroborates with the practices of the DevOps culture, including

questionnaire participant Q-10, who reinforces this belief with the statement *"Automated testing is critical in a DevOps world"*. When the test process involves manual activities or specific technical knowledge, this may be a bottleneck in the development process. The importance of test automation was also explicitly mentioned by questionnaire participant Q-46.

> *"There is a lot of manual work in application testing. In the future, it would make sense to work on a higher abstraction development."* (Q-46)

Finally, the importance of testing was broadly commented in both the interviews and questionnaires. Specific difficulties were also identified for testing DSP applications. Therefore, data stream testing emerges as a fruitful topic to explore as the core category of this work.

4.2.2 Category: Testing Approaches

The following subsections present a summary of the most recurring approaches on testing DSP applications.

4.2.2.1 Category: Levels of Testing

Traditional testing methodologies continue to be employed to test DSP applications. However, participants also reported adaptations and special configurations in traditional test levels for DSP applications. The comment of interviewee P6 illustrates this position:

> *"Software development has not changed, so unit testing and integration testing are continuing, but now considering data stream particularities."* (P6, Data Engineer, Financial)

Regarding unit tests, 10 interviewees and 59 survey respondents indicate that unit tests are consolidated for testing data stream modules. In general, no particular difficulties were reported when implementing unit tests in the context of DSP. The comment of P7 is an example of this position:

> *"Usually, we test based on unit testing in development, and only after passing in all unit tests, we can deploy it to production."* (P7, Big Data Engineer, Financial)

As for integration tests, no challenge has been reported, and it can be easily automated using test data mocking tools. However, difficulties start in system tests, because it involves all services running. Some participants reported building a copy of the production environment for system testing, but this requires an abundant infrastructure, and it is a costly solution. Comments of P2 reinforce challenges on system test:

> *"Here we perform system tests. But in the stream part, it is not that easy to test end-to-end. It involves raising and connecting several micro-services."* (P2, Software Architect, Software Consulting Sector)

Interviewee P2 also comments on performing manual tasks on applications testing.

> *"Today our life cycle is not fully automated. There is some process of stream testing that is performed by humans; the analyst assembles the system test and monitors it. Our continuous integration system does not have something that automatically triggers that test and sends a report with the results."* (P2, Software Architect, Software Consulting)

Another issue highlighted by the participants was the relevance of load tests in system test levels to find typical data stream problems that occur with system stress:

> *"Data stream has exacerbated the importance of load testing, especially in the stage environment with all modules installed."* (P6, Data Engineer, Financial)

Finally, the following quotation synthesizes the testing process from the company of respondent Q-24. This comment corroborates the previous quotes and observations:

> *"i) Unit testing, locally on devs' machines ii) Integration testing, locally on devs' machines. We leverage Flink's ability to run stream-processing in a local environment. iii) End-to-end testing on staging environment. We use production test data from an in-house built generator."* (Q-24)

4.2.2.2 Stream Simulation

Stream simulation is an approach for testing stream applications realistically. It involves replaying historical or generated data. Nineteen questionnaire participants reported performing stream simulation as a test approach. They reported using their solutions based on existing streaming tools and homemade scripts.

> *"There are some scripts we use to simulate streams, which are method callers, one to read the stored data and the other to write on the topic."* (P3, Software Engineer, Location Data Services)

Simulation tools may provide features for handling typical problems of DSP applications, such as the message entry semantics, loss of messages, duplicate messages, corrupt messages, message asynchronicity, latency, hardware failures, throughput and others. In fact 83% of respondents consider fault tolerance features helpful in data stream simulation tools. The following excerpt illustrates these comments:

> *"Simulating fault tolerance is interesting because with a history replay we can perform the processing and then assert the results. Meanwhile, there may be some scenarios of service dying and rising, machines with problems and latency on the network. We can check if even with this high entropy, the answers are consistent."* (P4, Software Engineer, Financial)

Another use of stream simulation from stored data would be to correct the past when there is a bug in the application, and it has processed the wrong data for a period of time. In that case, you need to discard the erroneous results from that period and re-run that part of the stream with the new version of the application without bugs. This correction can be

done by simulators, giving up the real-time. This situation is associated with the concept of the immutability of events in stream processing, the simulator allows to re-execute the event history and to revert the problems by calculating the correct results. Two interviewees mention this specific case.

> *"I have to revert and reproduce the same data that was consumed and then process them again, even giving up real-time response. This can be done with this simulation tool."* (P3, Software Engineer, Location Data Services)

4.2.3 Category: Testing Tools

Below, we present test tools along with observations about the tool's purpose. For unit tests, there is the Spark Testing Base, which emerged in questionnaires and on P9's interview:

> *"There's a rich set of testing libraries, such as Holden Karau's Spark Testing Base, which is pretty standard in many companies."(P9, Data Engineer, Food Delivery Service)*

Pact is a contract testing tool, which is used for testing the schema registry in stream processing messages, ensuring that stream services can communicate with each other.

> *"There is the tool to test communication via messages like Pact, it checks when you break the contract of another team."* (P8, Software Engineer, Software Consulting)

In the context of fault tolerance, the Chaos Monkey tool was cited. It submits applications to random infrastructure failures. However, this tool does not simulate specific stream failures like message semantics problems.

> *"To test fault tolerance, we use Chaos Monkey. It creates a chaotic scenario in the infrastructure, turns off machines and services, increases latency, and affects processing capacity. It tries to force the whole architecture to fail."* (P9, Data Engineer, Food Delivery Service)

Participant Q-38 mentions that the WSO2 stream processing tools have features related to stream simulation, data replay and historical and application debugging.

> *"WSO2 Stream Processor offers debugging and allows you to replay historical data, simulate random events and replay events in an accelerated way."* (Q-38)

There are also reports about in-house tools for system tests. An example is the Ducktape tool cited by Q-1 participant.

> *"System testing is done via Ducktape (in-house built tool, but open source). It includes applications correctness tests, performance tests, and fault-tolerance tests."* (Q-1)

Finally, although many participants claim to build and use their own solutions for generating test data, the libraries Clojure Test Check and ScalaCheck were mentioned as useful tools to build generative tests based on properties.

> *"Clojure has good tools for generative testing, streaming has message contracts, and the applications have entry and exit contracts, so we usually use generative test tools that rely on contracts to generate test data in an automated way."* (P4, Software Engineer, Financial)

4.2.4 Category: Test Data

Test data is a subject that emerged a few times in the interviews and questionnaires, with many test data sources being reported, such as replay of historical data, real-time production data mirroring, and synthetic data generators. The quote below exemplifies the use of various test data sources:

> *"Replay historical data, real-time use of production data piped to test environment, custom data generation using algorithms. In replay & real-time data use, the data is often remapped/sanitized for testing & proper simulation in a test environment."* (Q-100)

Although widely adopted, replaying historical data is a sensitive point, especially in contexts where data is confidential and privacy is a priority, such as banking and financial applications. Therefore, data generation may be the only option in many situations. Interviewee P2 has expressed concern about privacy with the following comment:

> *"The ideal would be the use of historical data, the problem is the restriction of secrecy and privacy, so we work with mock data. We face this barrier in the application of insurance analysis, bank information and restricted information on people."* (P2, Software Architect, Software Consulting)

Meanwhile, eight questionnaire participants mentioned privacy concerns in using historical data, and they suggested data generators as an alternative way of getting around this problem. The following excerpt of Q-96 illustrates this concern from one of the participants in the questionnaire:

> *"Mostly replay of historical data. Privacy laws make this less straightforward. We are experimenting with data generators that take into account dependencies between different tables/topics."* (Q-96)

Regarding data generators, 59 participants in the questionnaire indicated the use of test data generators. Interviewee P3 mentioned the use of Domain-Specific Languages (DSL) based tools as a technical solution for generating test data.

> *"A tool in which I can easily produce data in a pattern, for example, a DSL in that it would be effortless to describe the data and that I want to make it happen in the stream."* (P3, Software Engineer, Location Data Services)

Test data stands out as a source of difficulties. Historical data is often unavailable for privacy reasons. Although it is a viable alternative in some cases, a random data generator may not generate data that are consistent with the real application scenarios. Participant P11 addressed the importance of using realistic data for testing DSP applications:

> *"We have difficulty in testing fraud detection. We can not always simulate every case. We usually have to simulate forcing some situation of artificial fraud with a fake GPS, which does not represent the scenario of fraud that really happens."* (P11, Data Engineer, Data Science Solutions)

In this case, a more robust approach, based on artificial intelligence techniques, is needed. Thus, generators could be intelligent to the point of receiving as input a stream set and be able to detect the format of the messages, standard values and sequential pattern of values on different topics, generating multiple test streams. In the questionnaires, three answers reinforced the idea of the need to generate meaningful test data, as one of them quoted next:

> *"Generation of meaningful data (not just random) in case there are aggregations or joins within a window, that would allow providing meaningful deterministic output."* (Q-23)

4.3 RESULTS SYNTHESIS AND DISCUSSION

The previous section presented each category, where participants reported difficulties and relevant aspects when testing DSP applications. The remainder of this section presents a synthesis of the findings and promotes discussions based on relationships between categories and literature.

4.3.1 Approaches and techniques being adopted to test DSP applications.

The traditional methodologies and techniques of software testing remain valid in the DSP context. So, considering data stream particularities, unit tests, integration tests, and system tests are recommended. The comment from P6 in Section 4.2.2 motivates this aspect.

Unit tests are widely employed, and no particular difficulties were reported on unit tests. Participant P7 reinforces this statement (Section 4.2.2). In the same way, integration testing is being performed with data mocking tools. However, many challenges emerged on system-level tests. Interviewee P2 (Section 4.2.2) addresses this issue.

Due to non-determinism in distributed systems, DSP applications are subject to inconsis-

tencies such as glitches (COOPER; KRISHNAMURTHI, 2006). The problems of DSP are revealed when different modules have to interact. Therefore, solutions to support testing of DSP applications have to focus on integration and system testing.

Finally, stream simulation emerged as a specific data stream approach for more realistic system-level testing. Simulation tools exercise specific data stream problems through load tests and fault tolerance simulation. In Section 4.2.2, we presented some benefits of using simulators according to practitioners' comments.

4.3.2 Test frameworks and tools have been adopted

Participants revealed which tools they are using for testing DSP applications as well as their purposes. Table 12 summarizes the most cited tools. Generally, practitioners indicated the use of custom-developed in-house tools, specialized test libraries for stream processing, adaptations of conventional testing tools, and tools integral to constructing the testing infrastructure. More detailed information is presented in Section 4.2.3. These tools were also identified in the subsequent study, the GLR. In Section 5.5.1.5, the use of these and other tools was discussed in detail.

Table 12 – Most significant adopted tools

Purposes Uses	**Tool Name**
Contract Testing Tool	Pact
Fault Tolerance	Chaos Monkey
Integration Tests	Local Stack, Jenkins, Flink Test Util
Property-based Test	ScalaCheck
Stream Simulation, Data Replay and Data Generation	WSO2 Stream Process
System Tests	Pepper-Box plug-in on Jmeter Ducktape
Unit Test	Spark Unit Test Modules, Mocked Streams, Flink Spector

4.3.3 Data used to test DSP applications

Test data stands out as a source of difficulties. The category *Test Data* revealed several test data sources (e.g., historical or custom data, mirroring production streams and generators).

Historical data seems to be a good option, and it has been adopted by many companies. The main issue in adopting this technique is the fact that, in many cases, stream data is sensitive (for instance, financial data). Some companies mitigate this issue by anonymising all historical data, but there are contexts where privacy laws, such as *EU General Data Protection Regulations* (VOIGT; BUSSCHE, 2017), significantly restrict access to data. Comments from participants Q-100 and P2 (Section 4.2.4) illustrate this issue.

When real or historical data is not available, an alternative is to build a test database or use data generators such as *Threat Stream Generator* tool (WHITING; HAACK et al., 2008). Participants Q-96 and P3 claim that data generators are being used in their companies.

However, many data generators are not capable of generating consistent test data. Participant P11 presents the importance of realistic data to test fraud detection features. For that matter, participant Q-23 explains the need to generate significant test data. Therefore, data generators are still an open issue for DSP applications. Solutions in that sense can explore statistical models and machine learning techniques to generate more realistic data.

4.4 THREATS TO VALIDITY

A Grounded Theory framework offers rigorous procedures for data analysis. However, any qualitative research may contain some degree of research bias. Indeed, other researchers might have different interpretations, and so diverse conclusions may emerge from the same data. This is a common threat related to GT studies. Consequently, these studies do not claim to generate definitive findings. The resulting theory, for instance, might be different in other contexts (DENZIN; BRYANT; CHARMAZ, 2007). To mitigate this threat, we reported and made available all produced material so that other researchers can reproduce the data analysis in order to prove or compare the work done in this study.

To obtain willing participants, it was essential to guarantee that any sensitive data — not just of the individual participants but also their companies and third-party clients — would be kept confidential to researchers under human ethics guidelines governing this study. That way, all data were analysed anonymously.

One of the limitations when applying a computer-administrated questionnaire is the time spent on an accurate answer (RICHMAN et al., 1999). To mitigate this threat, we ignored all answers that lasted less than 5 minutes and those that wrote too short answers. Thus, with these parameters, four answers were discarded, leaving 101 answers.

We used the triangulation technique to increase confidence and reduce the subjectivity of our results (JONSEN; JEHN, 2009). Triangulation can be defined as the combination of diverse methodologies in the study of the same phenomenon. In our case, we combined interviews (Section 3.1.2) and questionnaires (Section 3.1.1). Besides, all categories and themes that emerged from data analysis were extracted and combined by two researchers. In case of disagreement, the third researcher was considered an Oracle.

Finally, our study posits itself from the point of view of practitioners with different backgrounds and working in companies from different domains (Section 4.1). However, we do not claim that our results are valid for other scenarios. Moreover, one of the main results is that testing a stream application depends on its context.

4.5 SUMMARY

This chapter presented the exploratory study based on qualitative research, which involved semi-structured interviews with 12 professionals and questionnaires administration to others 101. The interviews went through a transcription and coding process, and the questionnaires were analysed to reinforce findings.

The study reported testing challenges related to performance, fault tolerance, realistic system-level testing, and stateful operations issues. Approaches like unit testing, integration testing, end-to-end testing, replaying historical data, and simulating real-time scenarios have been employed. Tools such as Apache Kafka, Apache Flink, Apache Storm, and other in-house solutions were prominent in handling DSP tests. However, a significant gap in formal knowledge and training in DSP testing was identified, highlighting a need for more structured learning resources and standardized guidelines to support testing activities in this field.

This study contributes to the thesis by exploring the field of research, reporting the perspective of practitioners and providing directions for topics to be explored in future studies.

5 GREY LITERATURE REVIEW

This study has been fully published in an article that presents the results in detail, providing all the evidence needed to substantiate each finding (VIANNA et al., 2023). Given that, the research report is particularly extensive and includes numerous quotes from grey sources, we have opted to provide in this chapter a concise summary of the results along with the discussion, allowing us to concentrate on discussing the main findings.

5.1 INITIAL SEARCH

The execution of our searches unfolded in the following steps:

1. We ran each of the five search strings on Google and saved the results.
2. We configured Google to search within the domains of the three specialized websites—Stack Overflow, Medium, and LinkedIn Pulse. After executing each of the five strings on these sites, we combined these results with those from the initial Google search.
3. We removed any duplicate links.

We ceased to save results once a new page of ten results no longer produced relevant sources as expected to satisfy the saturation criteria. Ultimately, our search process yielded 1960 sources, and then, we applied an automatic filter to remove 293 duplicate links, resulting in a final pool of 1667 grey sources. Throughout this process, we utilized scripts to automate tasks like searching (using a Google search python API (VILAS, 2020)), metadata collection, duplicate source checking, and content downloading for research documentation.

5.2 APPLICATIONS OF INCLUSION AND EXCLUSION CRITERIA

Two collaborating researchers assisted in the criteria application. We held a training session for the evaluators, followed by a pilot study in which each collaborator assessed five sources. Afterwards, individual meetings were held to collect feedback and address questions. The pilot study improved the checklist's applicability and criteria refinement. Each source was analyzed individually, and then through a paired review, a third evaluator decided the tie cases. In total,

the application of the inclusion and exclusion criteria yielded 101 studies. In subsequent steps, we added 53 sources using the snowballing technique (WOHLIN, 2014), examining reference lists and back-links, bringing the final count to 154 studies. Figure 12 provides an overview of the applications of inclusion and exclusion criteria results, detailing the number of sources discarded at each inclusion criterion stage. The list of selected sources is available in Appendix F and is referenced throughout the text as *Source [ID]*.

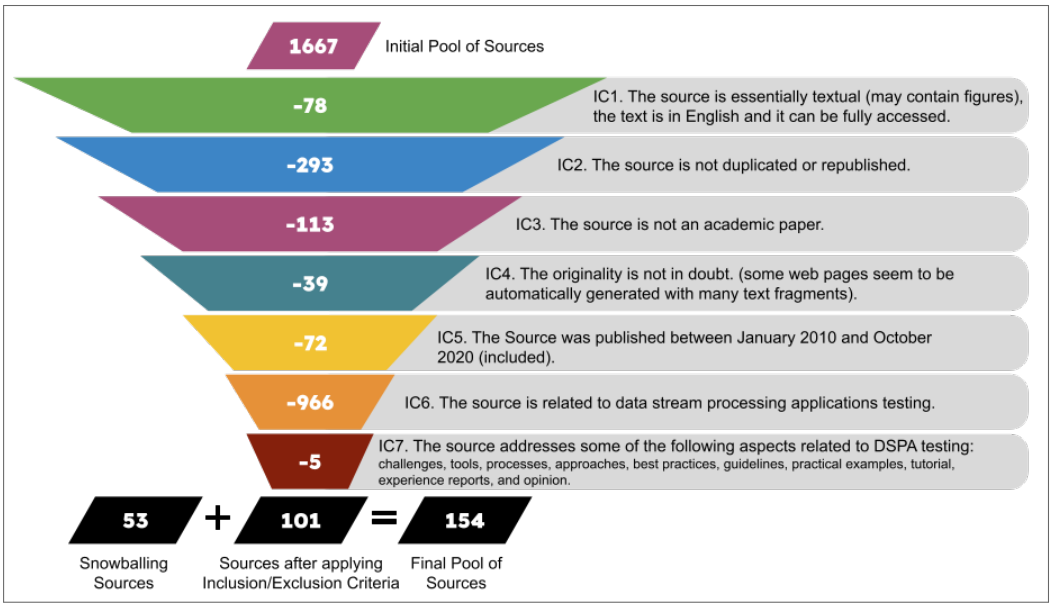


Figure 12 – Result of Source Selection in Numbers.

5.3 DEMOGRAPHIC DATA

From a total of 154 sources selected, 42 were found with standard Google searches, while 59 came from searches on specialized sites, including 25 from Medium, 28 from Stack Overflow, and six from LinkedIn Pulse. Snowballing has added another 53 fonts to the pool. The most common source type is blog posts, with 78 sources, of which 27 are on organizational blogs, 12 on individual blogs, and 39 on technical blog platforms. The following are 33 Q&A websites, 17 Software/Tool Documentation, 17 Tool Repository, 3 Email Discussion List, 4 Slide Presentation, and 2 White Paper. Figure 13 shows, as a bar chart, the number of sources selected each year. We observed that we did not have obtained sources from years before 2013 and that the number of sources increased each year in subsequent years, reaching 59 in 2020. We observed that the tendency towards an increase in the number of grey sources in recent years might have been amplified by search engine bias and automatic date updating in some

sources, such as tool documentation. Another possible reason for this trend is that some web content becomes unavailable over time. For example, companies may change their name or publishing policies, and then technical blogs may be discontinued, leading to fewer sources being found in the first few years.

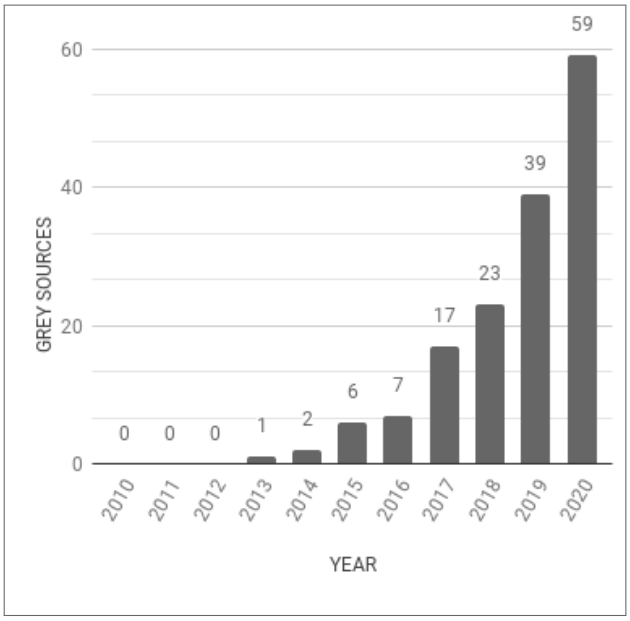


Figure 13 – Selected Sources by Year

Among the companies, we highlight the ones with the highest participation: Apache Foundation, Amazon Web Services, and Confluent; they all provide services, tools, and training related to DSP. The contents produced by these companies are the official tools documentation and posts on organizational blogs. Other traditional technology companies, such as Dell, Google, and Microsoft, are also present among the authors but with lesser participation. Companies in diverse business sectors, such as Airbnb, Alibaba, Otto Group, and Deliveroo, employ DSP in their technological solutions and have produced content on organizational blogging platforms.

Individual authors include independent consultants, book authors, members of the open-source community, and employees of companies working in the DSP field. The industry is the most frequent author affiliation, with 146 sources, three sources from academia, and five from industry and academia collaborations.

5.4 QUALITY ASSESSMENT RESULTS

As described in Section 3.2.3.1, the quality assessment score ranges between 0 and 1, representing the sources' average quality according to the aspects from Table 4. Figure 14 presents the histogram of the number of sources distributed in each score range. The average score of the 154 sources was 0.59, with the highest score of 0.92 and the lowest being 0.28. The average score on standard Google was 0.61, on Medium 0.57, on Stack Overflow 0.51, on LinkedIn Pulse 0.44, and snowballing 0.63. The higher average score of Google's standard search may be explained by the fact that this type of search brought up 1st-level Grey Literature sources, such as software/tool documentation, white papers, and blog posts on company websites. We opted not to exclude studies based on quality assessment scores to preserve potentially relevant content with low scores and acknowledge the potential for bias in the assessment process. Instead, we integrated these scores into our analysis, detailing them with the source references in Appendix F for a nuanced evaluation of each source's contribution and reliability.

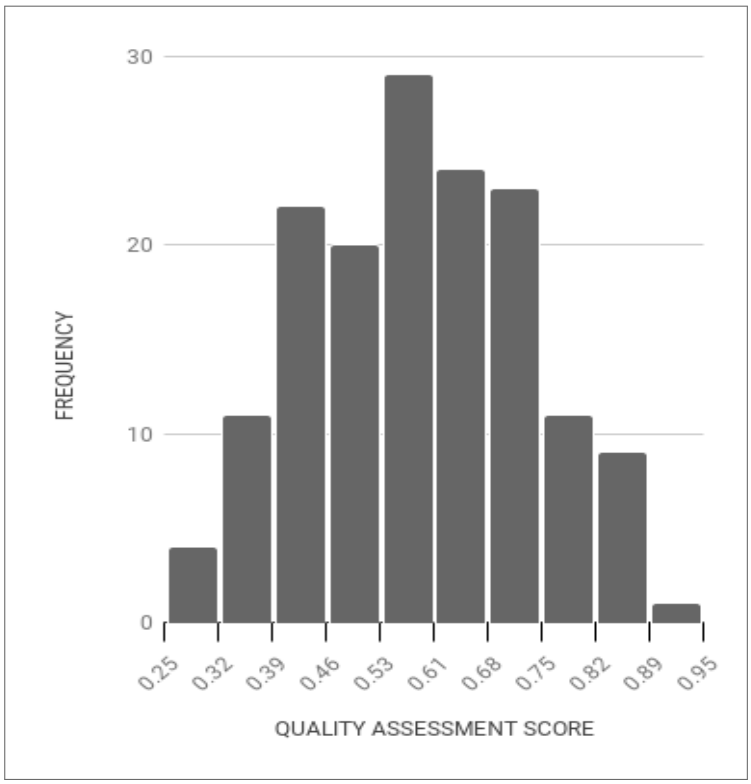


Figure 14 – Quality Assessment Score Histogram

5.5 RESULTS AND DISCUSSION

In this section, we first summarize our main research findings, discussing the most relevant ones, then the implications for research and practice, and finally, we discuss threats to validity.

5.5.1 Summary of research findings

Below, we present a summary of the results obtained for each research question, followed by a discussion involving the related academic literature.

5.5.1.1 RQ1. What are the challenges to DSP application testing?

Our first RQ focused on identifying professionals' challenges when testing the DSP applications. The RQ1 findings also reinforced the claim that testing DSP applications is challenging overall, as 92 of 154 grey sources address testing challenges in this context. At least 36 sources were topics in mailing lists or Q&A websites, on which practitioners raised specific technical questions and reported difficulties in testing DSP applications. Users sometimes reported generic difficulties and challenges addressed to particular aspects. Table 13 summarizes the most recurrent challenges and the number of sources related to each; these challenges are detailed in the subsections below. Table 14 summarizes the four main challenges that emerged from the grey literature, and the remainder of the section addresses each of them in detail.

Table 13 – Most significant challenges

Challenge	**Source**	
	#	**%**
The complexity of Data Stream Processing Applications	21	13.86
Testing Infrastructure	19	12.5
Test Data	15	9.87
Time issues	9	5.92

The complexity of DSP Applications. The complexity factor of DSP is already well known, and it has been reported in the formal literature (NAMIOT, 2015; CUGOLA; MARGARA, 2012b). The grey literature reports the factors that practitioners consider most challenging, such as

Table 14 – Key Points of Challenges to DSP application testing.

Challenges	Key points
The complexity of DSP Applications	▪ Non-determinism, asynchrony, complex transformations, lack of theoretical knowledge and changing variables make the test challenging. ▪ Testing strategies must test each transformation individually and then together, which can be time-consuming and require significant effort. ▪ Qualification of the test team and theoretical knowledge regarding DSP is required.
Test Infrastructure	▪ Building the test infrastructure as close as possible to the production environment is important, but it can be expensive. ▪ Automation tools and mocking services help reduce costs and simplify the construction of test infrastructure.
Obtaining test data	▪ Good test data is essential for realistic testing. ▪ Real data is difficult to obtain due to security and data privacy regulations. ▪ Historical data provides an initial data model, but generating and validating outputs is necessary to build a reliable oracle.
Time issues	▪ Timing affects aggregation, joining mechanisms, and the order in which events are generated, processed and ingested. ▪ Special handling of timing issues is needed when setting up tests, and tool support is essential for controlling the application's clock and watermarks. ▪ Choosing the processing time interval depends on the application's timing characteristics and requires a balance between precision and computational costs. ▪ Controlling the clock in the test environment allows speed-up test execution.

non-determinism, asynchronicity, complex transformations, a lack of theoretical knowledge on stream processing, and scenarios with continuously changing variables.

The DSP non-determinism makes test development challenging since multiple test runs can produce different results, making it hard to reproduce test scenarios and validate the correctness. In the formal literature, we find research testing strategies that may be adopted for DSP testing. For example, the study by Boroday et al. (BORODAY; PETRENKO; GROZ, 2007) proposes a test generation approach based on model verification for non-deterministic systems, while Leesatapornwongsa et al. (LEESATAPORNWONGSA et al., 2016) characterizes typical bugs related to non-determinism in distributed applications. Diaz et al. (DIAZ; SOUZA; SOUZA, 2021) propose test criteria to guide the selection of test cases in message-passing parallel programs; it helps to reveal nondeterminism-related defects, particularly in loops. Chen et al. (CHEN et al., 2015) provide a survey on deterministic replay focused on testing and debugging systems by controlling race conditions and non-deterministic factors.

Asynchronous stream processing can introduce race conditions, where the data processing order affects results, making their identification and resolution challenging. Yu et al. (YU; SRISA-AN; ROTHERMEL, 2017) proposed a framework to support testing for process-level race conditions, which considers asynchronous variables and provides some insights to address the problem in the DSP context. Asynchronicity also makes it challenging to test functions with timing dependencies, where the output of one processing step depends on the output of a previous step.

DSP applications can involve a sequence of multiple transformations, which can have complex rules and logic, where the output of one transformation stage depends on the output of a previous stage (DÁVID; RÁTH; VARRÓ, 2018). Consequently, building a test oracle also becomes a complex task. Testing strategies must test each transformation individually and then together, which can be time-consuming and require significant effort.

A practitioner also reported that a lack of theoretical knowledge about DSP made testing challenging. DSP testing requires qualified professionals with specific knowledge of the context; otherwise, they may be unable to interpret test results or elaborate efficient test strategies. This report refers to our statements regarding the complexity of DSP applications presented in the introduction and reiterated throughout the text. Furthermore, the comment reinforces the need for research work in the field to provide contributions to the industry and practitioners. Bath's study (BATH, 2020) addresses the different specialist competencies that professional testers of the next generation may need. Testing big data applications is discussed as a challenging context for testers due to characteristics such as performance, scalability, functional correctness, data currency, and testing of backup and recovery capabilities. This study supports our grey literature findings regarding the challenges of testing DSP applications.

Finally, it was also reported that the context of DSP applications is subject to continuously changing variables, such as data schema, configurations, infrastructure, partitions, offsets, and exception scenarios. These variables can make maintaining test coverage challenging, as existing test cases can become incompatible with changes, and test data may require updates to its structure. Constant changes also make it challenging to maintain automated tests and configure the test environment to accurately reflect the production environment. Regarding test case obsolescence in evolving software, Imtiaz et al. (IMTIAZ et al., 2019) conducted a systematic literature review that reports insights into how to prevent and repair test breakages.

Test infrastructure. From grey sources, we extracted reports on the importance of building

the test infrastructure as similar as possible to the production environment. Generally, the DSP applications adopt a large-scale cloud infrastructure based on microservices architecture, where the network interconnects the various distributed services. The first problem is the cost of allocating such infrastructure in the test environment, which typically includes allocating hardware resources, network services, tool licenses, and the consumption of third-party services. The second problem is the qualification of the testing team. In addition to knowledge of infrastructure technologies, they must also be skilled in testing techniques and frameworks. The mixed-methods study conducted by Waseem et al. (WASEEM et al., 2021) reported desirable skills for testing microservices systems, such as writing good unit test cases, analytical and logical thinking, and knowledge of test automation tools. In short, the infrastructure's complexity entails allocating costs and demands a qualified team.

The grey literature indicates infrastructure automation tools to reduce the time and resources spent on building test infrastructure. The tools help set up and configure services automatically, which were cited at the end of Section 5.5.1.5 and listed in Table 32. In the formal literature, Rafi et al. (RAFI et al., 2012) promoted a systematic literature review and a survey with 115 practices to investigate both views regarding the benefits and limits of test automation. The results corroborated the grey literature and indicated that a high degree of infrastructure automation saves costs. While Hynninen et al. (HYNNINEN et al., 2018) promoted a survey to explore industry practices concerning software testing, the study reported that more and more companies had adopted test automation and more sophisticated testing infrastructure, even in mission-critical software. Although test automation is an answer to facing infrastructure complexity, it is not an easy task, and practitioners considered one of the most challenging among all testing activities, according to a survey with 72 practitioners promoted by Garousi et al. (GAROUSI et al., 2020).

In addition, the grey literature also indicates mocking services, APIs, and infrastructures as a strategy to reduce costs and simplify the construction of the test infrastructure. Table 32 lists some tools for service mocks and DSP mechanisms. In the formal literature, Chen et al. (CHEN et al., 2017) state that mocking reduces testing costs in large-scale systems by reducing dependencies on external service providers. However, the author pointed out that the cost reduction would be more significant with automated approaches for creating and maintaining mocking services.

To conclude, the formal literature brings some contributions that help practitioners deal with the challenging context of the DSP applications test infrastructure. Benkhelifa et al. (BENKHE-

LIFA et al., 2019) reviewed cloud-based testing research, identified challenges, and characterized requirements associated with cloud testing environments, then proposed a framework for cloud-based virtual infrastructure testing servitization. Harsh et al. (HARSH et al., 2019) present a cloud-testing architecture focused on large-scale distributed applications, and the proposed solution benefits from managing resources and services to execute end-to-end tests simulating realistic operational conditions. Both works present approaches compatible with typical test infrastructures of DSP applications, supporting practitioners in constructing test infrastructures.

Obtaining Test Data. In the grey sources, we encountered statements emphasizing the importance of having good test data for realistic testing. However, in most cases, accurate test data based on real data is out of the question due to data security and privacy regulations such as the General Data Protection Regulation (GDPR). Strategies based on manually generating custom data can be a huge, time-consuming, and non-trivial task, whereas automatic test data generation strategies also present challenges in generating relevant test cases.

When historical data is available to develop the test dataset, it only provides an initial model of how the data streams are in the production environment, which does not guarantee a reliable test oracle. In grey literature, a practitioner reported that historical data results are not easily verifiable. Historical data may consist of only the inputs without the desired outputs, or if there are outputs, they may be unreliable. Therefore, generating and validating outputs to construct an oracle is necessary. This activity depends on the documentation containing the characterization and examples of desired outputs. Another problem is that historical data may not include new features.

In the formal literature, we found studies in line with the grey literature's findings regarding testing data challenges. The work by Felderer et al. (FELDERER; RUSSO; AUER, 2019) provides an overview of the current state of testing data-intensive software and cites test data quality as a relevant factor for testing data-intensive systems. The study by Benkhelifa et al. (BENKHELIFA et al., 2019), focused on testing cloud-based infrastructure, considers security issues and legal impediments to real business data use a challenge. Finally, the difficulty of generating high-quality data sets covering all relevant characteristics is a practical and methodological obstacle, as exposed by the study by Hummel et al. (HUMMEL et al., 2018).

Time issues. Time is a fundamental factor in DSP, as it affects aggregation and joining mechanisms, as well as the order in which events are generated, processed, and ingested (STONE-

BRAKER; ÇETINTEMEL; ZDONIK, 2005). The grey literature highlights the difficulty of simulating business-realistic temporal characteristics and evaluating time constraints in a test environment. Therefore, special handling of timing issues is required when setting up tests, and tooling support is essential for controlling the application's clock and watermarks. Testers can adopt a simulated clock to gain control for testing purposes in test environments. DSP frameworks offer control functions and interfaces in their test utilities to support the simulation of processing time.

Practitioners have reported that test execution would take a long time without clock control mechanisms in the test environment. Although no formal literature explicitly addresses this topic, the documentation of DSP frameworks such as Flink[1], Kafka[2], and Spark[3] describes clock control, functions for skipping some test cycles, and artificial watermark generation mechanisms. Test acceleration can be achieved using a simulated clock to generate time events faster than in real-time. For example, Flink supports event-time processing, where timestamps are used to process events. Clock acceleration in Flink can be achieved by implementing a version of the WatermarkGenerator interface to generate artificial watermarks that advance event time. Another option is to accelerate test execution by skipping some test cycles through functions that advance the clock. For instance, this approach can be implemented in Apache Kafka using the *TopologyTestDriver.advanceWallClockTime* function from Kafka Testing Utilities, which effectively skips over periods of time in the test environment to speed up the tests. However, this approach can result in a loss of precision and potentially hide bugs. Therefore, it is essential to consider the trade-off between test execution speed and result accuracy.

Another aspect related to time issues that testers should be aware of is the processing time interval in the test environment. The choice of the processing time interval may impact the testing process's accuracy and efficiency. A long interval may lead to inaccurate results, while a short processing interval means more frequent updates and accurate results but accompanied by an increase of computational overhead. Choosing the processing time interval depends on the application's characteristics, particularly the algorithms and functions that evaluate time factors or rely on timely updates. Some functions may work adequately over varying processing time ranges, while others may be quite sensitive and require more restricted processing time

[1] <https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/event-time/generating_watermarks/>

[2] <https://kafka.apache.org/24/javadoc/org/apache/kafka/streams/TopologyTestDriver.html>

[3] <https://spark.apache.org/docs/latest/api/java/org/apache/spark/util/Clock.html>

intervals. Experiments can be conducted to select the processing time, and it is essential to balance the need for accuracy and computational costs. In the test environment, it is recommended to reproduce the configuration of the processing interval faithful to the one used in production.

In the formal literature, we find research that addresses the difficulties that temporal aspects impose on testing distributed applications. Works such as (LIMA; FARIA, 2017; LIMA, 2019; LIMA; FARIA; HIERONS, 2020) address the conformance checking, observability, and controllability of time-constrained distributed systems. The test infrastructure communication overhead influences the test results, so checking the conformance of the observed execution traces against the specification becomes challenging. Researchers proposed Unified Modeling Language (UML) based specifications enriched with time constraints and algorithms for decentralized conformance checking in this context. Moreover, the study by Hierons and Ural (HIERONS; URAL, 2008) presents an algorithmic approach for checking the interaction sequence of distributed components and the test specification. The approach is free from controllability and observability problems. Finally, the study by Charaf and Azzouzi (CHARAF; AZZOUZI, 2016) presents an approach to generate timed distributed testing rules that avoid synchronization problems by considering the delay of messages between the under-test implementation and testers.

5.5.1.2 RQ2. What are the testing purposes?

From a business perspective, the test's purposes are related to financial impacts on the company and the competitiveness of the business's product. DSP applications handle high-volume operations per minute, turning a minor bug into a potential catastrophe and, consequently, a race against time to fix failures. Depending on the business sector, a minute of DSP failing may harm a startup as well as large companies. In the technical context, performing software testing aims to improve software quality. When analysing the sources, we observed that the test purposes placed by the practitioners corresponded to characteristics of the ISO/IEC 25010 software quality model (ISO/IEC..., 2011). We mapped the characteristics of the quality model with the practitioners' observations on the purpose of DSP application tests, such as functional suitability, performance efficiency, reliability, and maintainability. Table 15 summarizes the main aspects of each testing purpose, and the remainder of the section describes them in more detail.

Table 15 – Key points of testing purposes.

Testing Purpose	Key Point	Description
Functional suitability	Functional correctness	A key concern is ensuring that DSP applications deliver correct results. Unit tests for business rules functions are the first step towards achieving correctness, but integration and system tests in environments similar to production are also necessary.
	Variation in correctness	The notion of correctness varies according to the business context, as incorrect results may be tolerated in preference to other requirements such as reliability. The level of correctness of a DSP system can also be traded off with non-functional requirements.
	Differential testing	Approaches based on differential testing, such as DiffStream, can be used to verify the correctness of DSP applications. This involves executing two implementations in response to the same input stream and comparing the equivalence of their outputs.
	Mutation testing	Mutation testing improves test suite performance by generating syntactic variations of the original program using mutation operators to reveal programming errors caught by the test suite.
Performance efficiency	Time-behaviour	Time behaviour is affected by storage write speed, algorithmic characteristics, network throughput, latency, memory, processing power, DSP topology, and configurations. Testing the application under stress to understand data volume and latency is crucial.
	Resource utilization	The rational use of infrastructure resources is crucial as it financially impacts companies and their products' competitiveness. Performance testing assesses the application's utilization of hardware resources to promote optimizations.
	Capacity	Knowing infrastructure capabilities and understanding the bottlenecks and resources needed to scale is crucial. In a DSP system refers to the system's ability to handle a specified volume of data processing. Testing the infrastructure's operating limits helps to map bottlenecks and delimit operational limits.
Reliability	Fault tolerance	DSP applications must be fault-tolerant, as hardware failures, network oscillations, and other issues can cause the system to fail.
	Disaster recovery	The ability of the system to recover from severe failures that overwhelm fault tolerance mechanisms. The system must be restored to normal operation as quickly as possible after a catastrophic failure, and checkpoints and data loss must be considered.
	Chaos testing	It can help uncover potential system weaknesses and provide valuable insights for improving fault tolerance and recovery mechanisms.
	Overhead and performance degradation	Fault tolerance mechanisms can lead to performance degradation due to overhead, so tradeoffs between fault tolerance and the impact on system performance must be considered.
Maintainability	Modifiability	The ability to modify code without introducing new defects is crucial for cost-effective bug fixing and system evolution. This can be achieved through regression testing, ensuring that codebase changes do not negatively impact existing functionality.
	Architecture	A DSP system's architectural design can significantly impact its maintainability. Microservices-based architectures, for example, may reduce the testing effort required for regression testing by restricting tests to the modified microservice and its interfaces.

Functional suitability. In the grey literature, we identified that functional correctness is one of the concerns when testing DSP applications. However, we also found evidence suggesting

that the level of concern about delivering correct results changes according to the role of DSP in organizations' business. Furthermore, the notion of correctness in DSP applications does not strictly mean delivering correct results. Some incorrect results are tolerated depending on the context, while others are not. The findings indicate that using unit tests for business rules functions is the first step towards achieving correctness. However, the unit tests are insufficient to ensure that these functions will not fail in production by not exercising many features of complex distributed software. Although more costly and challenging, the grey sources indicate that testing strategies should include integrated and system tests in environments similar to production, where many DSP characteristics will be present.

In the academic context, we find discussions about the variation of the notion of the correctness of complex data-intensive systems. The work by Hummel et al. (HUMMEL et al., 2018) highlights the difficulty of determining whether an operation result is correct or incorrect. Such difficulty hinders the testability of this type of software, as it is not possible to have accurate test data to test. Furthermore, as Gunawi et al. (GUNAWI et al., 2014) explained, in some business contexts, incorrect results may be tolerated in preference to other requirements such as reliability. These statements support the GLR's finding that the notion of correctness varies according to the context. Following the same path, the work by Akidau et al. (AKIDAU et al., 2015) treats the level of correctness of a DSP system in a tradeoff with non-functional requirements. This approach starts from the premise that the correctness level can vary according to the problem domain and available resources. The work proposes the Dataflow Model, an approach based on formal models to help developers properly balance the dimensions of correctness, latency, and cost dimensions to suit their context.

Regarding correctness-focused approaches, Kallas et al. (KALLAS et al., 2020) propose an approach based on differential tests to verify the correctness of DSP applications. The work presents a testing framework that includes the DiffStream library for the assisted execution of two implementations in response to the same input stream. The framework also provides an algorithm to compare the equivalence of the outputs generated by the two implementations. We also found the research by Gutiérrez-Madroñal et al. (GUTIÉRREZ-MADROÑAL; GARCÍA-DOMÍNGUEZ; MEDINA-BULO, 2019) addressing the mutation testing technique to improve the performance of test suites in verifying the correctness of event processing programs. The technique generates syntactic variations mutants of the original program by applying mutation operators. Artificial faults injected into the original program represent developers' programming errors, and running the tests reveals whether the test suite caught the inserted error.

Performance efficiency. We identified 41 quotes related to performance efficiency. Table 15 outlined three key points concerning performance efficiency testing purposes: (1) Time behaviour, where testing aimed to verify the application's response times in contexts where time behaviour was crucial; (2) Resource Utilization, focusing on the rational use of resources and measuring the ideal amount allocated to develop efficient resource sizing strategies; and (3) Capacity, in a DSP system refers to the system's ability to handle a specified volume of data processing within given constraints with tests concentrating on the system's operational capacity. Capacity can be measured through throughput metrics in stress testing assisting in mapping bottlenecks and defining the infrastructure's operational limits.

From grey literature, we highlighted [Source 10], a white paper by a hardware vendor that presented results from performance experiments on a DSP-focused platform. This white paper underscored the industry's attention to DSP hardware platform performance. Performance concerns were also prevalent in academia, evidenced by several studies discussing DSP systems' performance (TANTALAKI; SOURAVLAS; ROUMELIOTIS, 2020; MISHRA; VARMA et al., 2020; ZEUCH et al., 2019). Karimov et al. (KARIMOV et al., 2018) introduces recommendations and metrics for evaluating DSP systems. Other academic studies compared DSP frameworks' performance(SAMOSIR; INDRAWAN-SANTIAGO; HAGHIGHI, 2016; CHINTAPALLI et al., 2016; SHAHVERDI; AWAD; SAKR, 2019) and evaluated specific techniques' performance (MONTE et al., 2020; TUN; NYAUNG; PHYU, 2019). Additionally, academic research examined the efficient use of resources by data stream frameworks in cloud-based infrastructures, with Chatterjee and Morin (CHATTERJEE; MORIN, 2018) conducting a comparative study on resource usage across three frameworks for the same application.

Reliability. Stream processing is usually associated with companies with high-reliability requirements, and we often found a recurrence of this subject in grey sources. In total, we found 32 quotes related to reliability. For example, in [Source 29], investment in testing was explicitly cited as an element in ensuring the reliability of the Alibaba Group's Blink DSP framework. In this context, the GLR reliability-related findings essentially focus on fault tolerance and recoverability.

Fault tolerance targets keeping the system running under adverse conditions, such as hardware, network, and software end third-party services. At the same time, recoverability focuses on the system's ability to recover from a failure with the minor damage possible. Chaos Testing strategies reported by practitioners in Section 5.5.1.3 address fault tolerance and recoverability. In this approach, controlled experiments are conducted using specific tools to simulate failures,

testing the application's ability to continue running in the face of failures and its ability to recover from failure scenarios. These experiments help uncover potential system weaknesses and provide valuable insights for improving fault tolerance and recovery mechanisms. By exposing the system to various failure scenarios in a controlled environment, Chaos Testing can also help to build confidence in the system's ability to handle unexpected events.

Fault tolerance in DSP has been addressed by several works over time, confirming the relevance of the issue that came to light in the GLR. Academic studies such as (SHAH; HELLERSTEIN; BREWER, 2004; BALAZINSKA; HWANG; SHAH, 2009; AKBER; CHEN; JIN, 2021) bring fault tolerance techniques to the DSP context. Wang et al. (WANG et al., 2022) perform a systematic overview and classification of fault tolerance approaches for DSP. It also proposes an evaluation framework tailored to measure quality in terms of runtime overhead and recovery efficiency of fault tolerance approaches. Finally, the work presents three directions for developing future fault tolerance approaches in DSP: Adaptive Checkpointing, Integration of Modern Hardware, and Parallel Recovery. Although fault tolerance is a desirable feature in DSP, such mechanisms can cause performance degradation due to overhead. Vianello et al. (VIANELLO et al., 2018) performed an evaluation experiment to measure the overhead cost generated by a fault tolerance mechanism in a DSP system. The results indicate a 10-fold increase in latency when these mechanisms are active. The fault tolerance and overhead tradeoff have also been reported in the grey literature Zachary Ennenga in [Source 18].

Maintainability. The maintainability is pervasive in several grey sources, particularly those associated with the DSP Complexity and Test Infrastructure challenges presented in Section 5.5.1.1. Such challenges increase the difficulty and cost of DSP application modification and testing. Mostly we have found grey sources referring to the practitioners' concern with the ability to modify code without introducing new defects. Low degrees of modifiability can result in costly bug fixing and system evolution processes. Additionally, it can lead to more bug insertions into previously functioning features, known as regression. At this point, practitioners emphasize the importance of regression testing in dealing with modifiability, and such a strategy is described in Section 5.5.1.3.

In the formal literature, studies focus more on architectural aspects as a relevant factor of maintainability in the context of DSP software (ASSUNCAO; VEITH et al., 2018; HOQUE; MIRANSKYY, 2018; ASTORGA et al., 2018). However, as the Silva et al. (SILVA et al., 2018) study exposes, regression testing generally requires robust automation of the development infrastructure, and the construction and maintenance of such a test suit for continuously updated

regression tests are challenging. The difficulty of maintaining large numbers of tests can be mitigated through regression test reduction techniques such as Requirement Based, Coverage Based, Genetic Algorithm, etc (SULEIMAN; ALIAN; HUDAIB, 2017). Especially, the study by Hoque and Miranskyy (HOQUE; MIRANSKYY, 2018) shows that in contrast to monolithic architectures, microservices-based architectures restrict the execution of regression tests to the modified microservice and its interfaces, leading to a reduction of testing efforts.

5.5.1.3 RQ3. What are the approaches and specific types of tests performed?

This RQ identified and selected which testing approaches practitioners have been using according to the grey literature. The findings are based on 155 citations in 82 grey sources reporting the testing approach. We first present the testing approaches according to the target of test, then focus on the most relevant specific approaches. As an aid, we have included Tables 16, 17, 18, 19, 20, and 21 with a brief summary of each testing approach in the following subsections. These tables highlight fundamental aspects, testing purposes, challenges, tools, and variables to consider for each approach. In this discussion we adopted the SEWBOK Version 3 terminology for software testing presented in chapter 4 of the guide (BOURQUE; FAIRLEY, 2014).

Approaches by Target of Test. This topic covers DSP application testing, considering relevant aspects regarding unit, integration, and system testing levels.

- Unit tests, essential for verifying module functionality, often do not address distributed systems' unique characteristics and non-functional requirements, such as end-to-end response time. Stateless operator unit testing follows traditional practices, but testing stateful operators demands careful state management and cache data handling. Timed process operators introduce additional challenges, necessitating control over application clocks and event timestamps. Stepien et al. (STEPIEN; PEYTON, 2020), and Buchgeher et al. (BUCHGEHER et al., 2020) address the limitations of unit testing in distributed systems. Hill et al. (HILL et al., 2009) and Li et al. (LI et al., 2006) discuss strategies and tools for developing unit tests that meet distributed systems characteristics.

- Integration tests, targeting logically integrated components, expose issues from module interactions, particularly timing issues identified in Section 5.5.1.1. Timing challenges

and the need for application clock handling and event timestamp configuration become apparent in integration testing.

- System testing, focusing on the entire software product, brings forth the complexity of DSP applications, as detailed in Section 5.5.1.1. This stage highlights distributed applications' typical characteristics, such as non-determinism and message asynchrony. System tests need to run in an environment closely mirroring production, making infrastructure automation crucial. The complexity of DSP applications in system tests, particularly in SaaS-based infrastructures, underscores the challenges in cost and environment configuration. This aligns with the *Testing Infrastructure* challenge presented in Section 5.5.1.1. The academic literature highlights the difficulty of testing large-scale distributed systems due to the complexity and size of the infrastructure, as evidenced by studies such as Hanawa et al. (HANAWA et al., 2010) and Harsh et al. (HARSH et al., 2019).

Table 16 – Summary of approaches by the target of the test.

Approaches by Target of Test
Fundamental aspects: Unit tests for DSP applications involve particularities on stateful and timed process operators. Integration and system-level testing are important to address the specific characteristics of distributed systems, which are associated with several defects in DSP applications. **Related challenge:** The *Testing Infrastructure Complexity* challenge as we found reports about the complexity and cost of setting up realistic system test infrastructures. The *Time Issues* challenge is also addressed on behalf of timed process operators. **Tools adopted:** Terraform, Jenkins, Confluent CLI, Ansible, Docker, Vagrant, and AWS CloudFormation. (component mocking and infrastructure automation to address complex infrastructures) **Stateful operations:** Stateful operations involve maintaining a state between successive invocations of the operation, which can lead to complex testing scenarios. The key challenges in testing stateful operations include verifying that the state values are updated correctly and that cache data is purged as required. DSP platforms, such as Apache Flink and Kafka Streams, provide utility classes to support stateful operator testing. **Timed process operators:** rely on timestamps and the application clock to control their behaviour, making it necessary to manipulate time variables. Testing timed process operators require controlling the clock and providing event timestamps. DSP platforms provide utility classes and features to support timed process operator testing, as discussed in the topic *Time Issues* from Section 5.5.1.1.

Performance Testing. We have selected 75 quotes from 27 grey sources related to performance tests. The abundance of findings from the grey literature on performance testing highlights its relevance to the industrial context. At the same time, several academic works

address the performance evaluation of DSP applications and associated contexts. Academic works bring tooling and technical contributions that meet the concerns expressed by practitioners about performance tests, so on this topic, there is consonance between industry and academia.

Table 17 – Summary of performance testing key points.

Performance Testing
Fundamental aspects: Focus on evaluating whether the performance meets the application requirements, optimizes computational resources, and brings cost savings. For optimization purposes, tests must be executed with different configurations due to the many variables affecting the results. **Testing Purpose:** Performance Testing essentially addresses the *Performance Efficiency* testing purpose. **Tools adopted:** JMeter, Pepper-Box, ZeroCode Samurai, Kafkameter, Sangrenel, Artillery, Gatling, Ducktape and Grafana. These tools support generating loads and monitoring metrics. **Variables to consider:** Hardware resources, network resources, number of topics, producers, consumers, message size, timeout values, cache sizes, replication factors, etc.

Findings from the grey literature indicate that performance testing aims to assess whether the performance meets the application requirements and helps bring cost savings by enabling developers to optimize computational resources. Practitioners state that DSP application performance tests must be executed with different configurations due to the multiple variables that affect the results, including hardware and network resources, number of topics, producers, consumers, message size, timeout values, cache sizes, replication factors, and more. Thus, performance testing is a tool that also helps tune the system for specific scenarios. It helps to estimate the adequate hardware resources and adjust the various DSP tools settings. From a business perspective, testing also brings cost savings by facilitating developers to optimize the use of computational resources.

Stream processing platforms provide several capabilities for generating loads for performance tests. However, we identified that practitioners also use other tools and libraries with functionalities to meet specific needs and to configure more accurate traffic patterns. Table 17 listed several tools for generating loads and monitoring metrics during performance testing.

Academic interest in performance testing has been increasing since the 2000s, particularly in contexts related to DSP's typical features, such as large-scale distributed applications, cloud, and microservices that handle numerous simultaneous requests (DUMITRESCU et al., 2004; ZHOU et al., 2013; BARROS et al., 2007). In 2007 De Barros (BARROS et al., 2007) proposed various techniques to overcome challenges in building environments and performance test tools

that accurately simulate the same conditions observed in the production environment of large-scale distributed systems, also providing techniques to characterize load patterns for tests. In 2015, Jiang and Hassan (JIANG; HASSAN, 2015) surveyed the state of load testing research and practice, reporting techniques for designing, executing, and analysing the results of a load test. The paper by Eismann et al. (EISMANN et al., 2020) lists the difficulties that microservices architectures bring to performance testing, as it requires additional care when setting up test environments or conducting tests.

In the formal literature, we can find several benchmarking works of DSP frameworks (CAPPELLARI; CHUN; ROANTREE, 2016; CHINTAPALLI et al., 2016; SAMOSIR; INDRAWAN-SANTIAGO; HAGHIGHI, 2016; SUN et al., 2018; KARIMOV et al., 2018; SHAHVERDI; AWAD; SAKR, 2019; MISHRA; VARMA et al., 2020). These works focus on evaluating the performance of DSP frameworks in specific application contexts by varying functionalities and configurations. They also document methods and tools for conducting performance tests and list performance metrics considered relevant to the DSP context, such as memory consumption, CPU load, response time, and network throughput.

Other studies contribute to methods and tools to evaluate DSP applications' performance. For example, Pagliari et al. (PAGLIARI; HUET; URVOY-KELLER, 2020) propose the NAMB tool (Not only A Micro-Benchmark), a generic generator of DSP applications prototypes that provides high-level descriptions language to support developers in quickly building and assessing the performance impact of design choices. Garcia et al. (GARCIA et al., 2022b) presents SP-Bench, a framework for benchmarking DSP applications which aims to support users in creating custom benchmarks of real-world DSP applications. The tool offers an API and Command Line Interface (CLI).

We also find academic works proposing and evaluating techniques to improve DSP performance in specific contexts. Nardelli et al. (NARDELLI et al., 2019) proposed heuristics to calculate the placement of DSP applications in geo-distributed infrastructures. Wu et al. (WU et al., 2018) propose reducing the intra-node inter-process communication latency by reducing memory copy operations and waiting time for every single message. Garcia et al. (GARCIA et al., 2022a) analyzes the impact of micro-batch and data intensity on DSP applications with different parallel programming interfaces.

Regression Testing. Regression tests focus on verifying that each software modification does not introduce problems in other system components. Regression tests are associated with the

maintainability objectives (detailed and discussed later in this section), as they bring more security when making changes.

Table 18 – Summary of regression testing key points.

Regression Testing
Fundamental aspects: address regression bugs and application performance degradation introduced by software evolution. Automated CI-based test infrastructure is necessary to perform these tests. **Testing Purpose:** Regression testing primarily addresses the testing purpose of Maintainability. **Tools adopted:** Apache Griffin, Flink Spector, Flink Testing Utilities, Fluent Kafka Streams Tests, Kafka for Junit, Kafka Streams Testing Utilities, MemoryStream, Passert, Spark Testing Base and Spring Cloud Stream Test Support. **Variables to consider:** Configuration changes, non-determinism of distributed systems, microservices-based architectures, test coverage of input parameters, data privacy issues, and cost of resources and time. **Microservices-based architectures:** are widely adopted in DSP applications infrastructure but bring peculiarities to regression tests. Building a regression testing infrastructure that supports isolated microservice and integration tests with other components that may affect their functioning is necessary.

Grey literature shows that practitioners are concerned about regression bugs and application performance degradation caused by software evolution. To address these issues, practitioners consider regression testing essential, and the need for an automated CI-based test infrastructure to perform these tests has emerged. The Grey Literature findings also reported the robust automation of the development infrastructure required for regression testing, including high unit test coverage and Continuous Integration (CI) mechanisms. In the case of performance regression tests, a trustworthy test environment is preferable to the production environment, which increases the cost of test infrastructure.

While there is no specific formal literature on regression testing in the DSP context, related works in real-time distributed systems, microservices architectures, and large-scale software have been explored and discussed. Yamato (YAMATO, 2015) proposed a testing framework for real-time distributed systems to prevent new bugs introduced by configuration changes. Babaei and Dingel (BABAEI; DINGEL, 2021) tackled the non-determinism issue in distributed systems regression testing with MRegTest. This replay-based framework facilitates deterministic replay of traces and reduces testing costs by replaying executions that modify critical user-specified variables.

Microservices-based architectures in DSP applications present challenges for regression testing. Kargar and Hanifizade (KARGAR; HANIFIZADE, 2018) proposed an automated method

that integrates microservices regression tests into Continuous Delivery (CD) steps. Gazzola et al. (GAZZOLA et al., 2022) propose a tool that monitors deployed service executions at run-time, recording executions that can be later converted into regression test cases for future version services. However, data privacy issues may arise when monitoring production execution, as noted by the authors.

Academic studies have also explored the resource and time costs of running numerous regression tests on large-scale systems, as shown in Orso et al.'s work (ORSO; SHI; HARROLD, 2004). Techniques such as minimization, selection, and prioritization have been developed to address these issues. Minimization eliminates redundant test cases, and selection identifies relevant test cases for recent changes, and prioritization orders test cases for early failure detection (YOO; HARMAN, 2012; SULEIMAN; ALIAN; HUDAIB, 2017).

Property-based Testing. Property-based testing was reported as a technique adopted in the DSP context to generate representative test cases based on variable properties and undergo a refinement process called shrinking, which minimizes the number of inputs required to reproduce a failure. This method helps address the problem of missing real data for testing while mitigating irrelevant data from random data generators. It is a low-cost strategy that can be easily applied and yield good test coverage.

Table 19 – Summary of property-based testing key points.

Property-based Testing
Fundamental aspects: automatically generate representative test cases based on variable properties and refine them through a process called shrinking. It is a quick and inexpensive strategy to get more relevant test data. **Related Challenge:** It relates to the *Obtaining Test Data* challenge, as it is a test approach that automatically generates test data, mitigating the problem of missing real data and generating irrelevant data. **Tools adopted:** ScalaCheck, StreamDatam and EctoStreamFactory property-based testing library. Apache Flink and Apache Spark Streaming frameworks have property-based testing tooling based on ScalaCheck. **Variables to consider:** properties of the input data and the temporal logic to guide the generation of random streams with time stamps. **Mocking API:** The adaptation of property-based testing tools to generate mock API data for unit and integration tests when API services are unavailable in testing environments.

Grey Literature shows that the DSP community is already aware of the benefits of the property-based testing approach for the DSP context; it has also been documented in the formal literature. Espinosa et al. (ESPINOSA et al., 2019) and Riesco et al. (RIESCO; RODRÍGUEZ-

HORTALÁ, 2019) have brought property-based testing tooling to the Apache Flink and Apache Spark Streaming, respectively. These tools are based on the ScalaCheck library and use temporal logic to generate random streams and verify time-related properties. Furthermore, API testing is one of the uses of property-based testing, and the work of Karlsson et al. (KARLSSON; ČAUŠEVIĆ; SUNDMARK, 2020; KARLSSON; ČAUŠEVIĆ; SUNDMARK, 2021) exemplifies its applicability for this purpose. Such use is also relevant for DSP, where APIs are increasingly incorporated. Additionally, property-based tools can be adapted to generate mock API data for unit and integration tests when API services are unavailable in testing environments.

Chaos Testing.

Chaos engineering addresses the question: *"How much confidence can we have in the complex systems that we put into production?"*. Basiri et al. (BASIRI et al., 2016) defined chaos engineering in modern distributed systems as a controlled experiment that measures a system's ability to operate under realistic conditions. Large-scale DSP, characterized by its distributed hardware and software infrastructures with numerous potential fault variables, fit well within the context of chaos engineering. Recognizing the impossibility of guaranteeing constant infrastructure reliability, chaos engineering focused on preparing systems to function in adverse conditions by accepting that failures are inevitable.

As the number of distributed software and hardware components grew, the likelihood of runtime system abnormalities increased. Chaos engineering was reported in grey literature as an effective technique for testing and verifying the fault tolerance of large and complex distributed software systems. Practitioners expressed concerns about infrastructure failures in production environments potentially causing application malfunctions. They implemented testing practices where the system was intentionally subjected to failure to ensure that auto-recovery mechanisms functioned satisfactorily.

Chaos engineering is attracting the attention of both industry and research, as it is a technique for exercising and verifying the fault tolerance of distributed, large and complex software systems. Netflix employed the approach to testing the fault tolerance of microservices-based infrastructures. The work by Tucker et al. (TUCKER et al., 2018) analyzes the implementation of the chaos engineering techniques and reports the benefits generated for the company's business.

Chaos engineering has been explored in the formal literature in contexts associated with DSP, such as cyber-physical systems (KONSTANTINOU et al., 2021), cloud infrastructure (TORKURA

Table 20 – Summary of chaos testing key points.

Chaos Testing
Fundamental aspects: reduce the likelihood of failures in production. Chaos test subjects the applications to adverse conditions, such as network instabilities, hardware failures, and third-party service instabilities, and tests their fault tolerance ability and recoverability. **Testing Purpose:** Essentially, it is related to testing propouse *Reliability*. **Tools adopted:** Chaos Monkey randomly terminates service instances to test their fault tolerance. Jepsen allows the simulation of network partitions and faults to test the system's consistency and availability. Thundra is a serverless observability platform with a chaos engineering tool for testing serverless applications' fault tolerance. WireMock includes a feature for simulating network faults to test the system's fault tolerance. **Optimizing fault tolerance configurations:** Chaos testing can be used to optimize fault tolerance configurations for DSP jobs, which can help balance performance and availability. **Variables to consider:** Network instabilities, hardware failures, software failures, third-party service instabilities, timing issues, resource utilization and QoS constraints.

et al., 2020), and containerized applications (SIMONSSON et al., 2021). Specifically, in the DSP context, the work by Geldenhuys et al. (GELDENHUYS et al., 2021) proposes the Khaos tool. This tool employs Chaos Engineering principles to allow automatic runtime optimization of fault tolerance configurations for DSP jobs.

Chen et al. (CHEN et al., 2022) introduced a framework for adopting chaos engineering in large-scale distributed big data systems. The framework focuses on fault tolerance, resilience, and reliability and includes steps such as exception injection, exception recovery, system correctness verification, observable real-time data statistics, and result reporting. The work is based on real industrial applications and draws from experiences introducing Chaos Engineering at the Swedish company ICA Gruppen AB.

Contract/Schema Testing. The schema works as a contract to specify the message format in DSP systems, and validation consists of checking the compatibility of messages following the schema. Reports in the grey literature warn that a lack of data schema correctness control can cause compatibility breakages in DSP applications. Schema issues can be caused by programming errors, failure to communicate between development teams or schema updates. The larger the system, in terms of subsystems interacting and third-party services involved, the greater the chance of schema incompatibility failures

Practitioners recommend adopting unit and integration tests to validate that interfaces follow the message schema. A compatibility fault tolerance approach is recommended for third-party modules, such as external APIs. The ability to interoperate between different schema

versions to support schema evolution has also been identified. Schema registry management tools, such as Apache Avro and Confluent Schema Registry, can provide compatibility models among schema versions.

Table 21 – Summary of contract/schema testing key points.

Contract/Schema Testing
Fundamental aspects: Ensure that messages exchanged between different system components comply with a specified message schema. This is essential for preventing compatibility breakages in DSP applications, which can result in crashes or incorrect data processing. **Testing Purpose:** Supports the *Maintainability* testing purposes, as it helps prevent contract incompatibility caused by updates. **Tools adopted:** Apache Avro provides functionality for schema evolution and backward and forward compatibility. AWS Deequ supports both static and dynamic schema validation. Great Expectations provides functionality for defining and validating data expectations, including schema validation. **Variables to consider:** message format, message source and destination, integration with third-party systems, backward and forward compatibility, and recovery from errors related to schema incompatibilities.

Data schema incompatibility is a common problem in distributed systems, and several research projects have addressed it (BLANCHI; PETRONE, 2001; BAQASAH; PARDEDE; RAHAYU, 2015; FILIP et al., 2019; LITT; HARDENBERG; HENRY, 2021). DSP applications, which use APIs extensively, also face API interface compatibility issues. In this matter, Bustamante and Garcés (BUSTAMANTE; GARCÉS, 2020) addressed API incompatibility by adding an intermediary service between IoT devices and API servers that identifies different protocol versions and performs compatibility adjustments. Yasmin et al. (YASMIN; TIAN; YANG, 2020) proposed an automatic verification approach to identify impacted operations by deprecated API elements, collecting the OpenAPI Specification (OAS) format and performing a source code scan to verify calls.

In the context of DSP, Corral-Plaza et al. (CORRAL-PLAZA et al., 2020) proposed an architecture for processing heterogeneous data sources that typically involve JSON, XML, YAML, or unstructured raw data. To address recognizing and adapting to evolution changes, the authors proposed five steps to transform heterogeneous data into simple events: Data Consumption, Data Format Detection, Data Homogenization, Schema Generation, and Simple Event Generation. The architecture was evaluated in a real-world case study of IoT sensors, demonstrating the ability to automatically process and analyze heterogeneous data. This work is also classified as a fault tolerance approach as it provides automatic tolerance to schema changes.

5.5.1.4 RQ4. What are the strategies adopted by practitioners to obtain testing data?

In our GLR, we identified 65 quotes from 33 studies addressing issues on data for testing DSP applications. Practitioners consider it essential to take test data seriously and recognize the importance of having representative test data that closely resemble real data. Test data is necessary at all levels of testing, and in each situation, there are different requirements. For example, in unit testing, the data should enable finding corner cases, while in performance testing, a considerable volume of test data is needed to stress the application. Questions regarding data privacy and security emerged, and several strategies were listed. These strategies typically involve using various synthetic data generators, historical or production real data, and combinations of real and synthetic data. Table 22 summarises the key points of the strategies for obtaining test data. The remainder of this section discusses the advantages and disadvantages of the strategies and presents relevant works from the formal literature associated with each strategy.

Historical Data. Practitioners consider using historical data for testing interesting because it exposes the application to realistic scenarios and facilitates real-world bug reproduction in a testing environment. In favour of using historical data, practitioners argue that real data streams can be quite complex, making them difficult to reproduce manually or with automatic generators. While there are advantages, several quotes also claim the impossibility of obtaining real data due to privacy and security concerns. Data-related challenges were reported and discussed in Section 5.5.1.1.

Furthermore, historical data may not provide adequate coverage for detecting bugs in the future. Additionally, having access to historical data does not necessarily mean having a test oracle, as they can only be inputs without corresponding expected outputs. Even when outputs are available, the oracle is not guaranteed reliable. Also, historical data may not apply to new features or versions of an application, rendering it incompatible. As a result, historical data may not be sufficient for comprehensive testing, and complementary strategies for obtaining data should be considered.

Even when there are no restrictions on the use of real data, there are technical challenges to capturing and storing data streams because of the data's high volume and speed. In this sense, researchers propose solutions based on multiple services to efficiently collect and store massive data stream (MALENSEK; PALLICKARA; PALLICKARA, 2011; ALSHAMRANI et al., 2020; LV; LIU; JIN, 2021). In the formal literature, no works explicitly address using real historical data for testing

Table 22 – Key Points of Strategies for Obtain Testing Data.

Strategies to obtain testing data	Key points
Historical Data: It is real stream data obtained from the application in production.	▪ Exposes the application to realistic scenarios. ▪ Difficulties in obtaining historical data due to privacy and security concerns. ▪ May not provide coverage for detecting bugs in the future. ▪ Having historical data does not necessarily mean having a test oracle. ▪ May not be applicable to new features or versions of an application.
Production Data Mirroring: Strategy based on redirecting a replica of input data streams from the production to the test environment. Compares the outputs of two software versions running simultaneously.	▪ The application is tested with real data. ▪ Test could be very time-consuming as there is no possibility of speeding up the execution. ▪ Incurs costs associated with building and maintaining a replica of the production infrastructure. ▪ Execution in "shadow mode" is a strategy based on an automatic comparison engine to ensure that sensitive data remains secure.
Semi-synthetic data: The technique involves automatic data generation methods based on real data. Advanced techniques are based on extracting statistical properties from real data to establish a data model to generate new data.	▪ Suitable for customizing the data in order to contemplate more complex test cases. ▪ Can be used to expand a limited amount of real data available. ▪ Address privacy issues, as semi-synthetic data generation techniques focused on privacy preservation have been reported.
Synthetic data: Automatic data generation. Various techniques are available, ranging from simple random data generators to sophisticated algorithms for data generation.	▪ Take as input a formal description of the data schema to generate test data. ▪ Exploring information extracted from UML diagrams helps generate more representative test data. ▪ Generators employ various statistical methods that are configurable by the user. As a result, a solid understanding of statistics is crucial to effectively utilize these tools and generate high-quality test data.

DSP applications. However, in recent years, several works have been published proposing techniques for anonymizing real data to bypass privacy issues while mitigating the data's loss of usefulness (BEGUM; NAUSHEEN, 2018; KENTHAPADI; TRAN, 2018; MADAN; GOSWAMI, 2018; MAJEED; LEE, 2020; SHARMA; SINGH; REHMAN, 2020; HOSSAYNI; KHAN; CRESPI, 2021; SHREE et al., 2022; VASA; THAKKAR, 2022).

Production Data Mirroring. We found descriptions of mirroring production data to a testing

infrastructure for evaluating new software versions. Our previous exploratory study (VIANNA; FERREIRA; GAMA, 2019) also reported this approach. In this strategy, replicas of production data streams are redirected to a server with the application under test. The advantage of mirroring production data is to expose the application under test will to a real-world stream, which is valuable for checking regressions in new application versions.

However, this approach also raises data privacy issues. In the Grey Literature, we identified a proposed solution to maintain data privacy while mirroring production data. The [Source 89] describes an approach where an application was executed in 'shadow mode', computing results and automatically comparing them with the production version's results. The result comparison strategy should adopt specific statistical methods to compare results, as values may vary widely due to various factors, even in accurately replicated environments.

This approach is helpful for testing without compromising data privacy, as an automatic comparison engine ensures that sensitive data remains secure. It provided an alternative to using real data for testing purposes while maintaining data confidentiality.

On the other hand, the tests are limited to cases where it is possible to compare results. For instance, new features do not have comparison parameters because they are absent in the previous version. We also observed that this type of test could be very time-consuming, as there is no possibility of speeding up the execution. It also incurs costs associated with building and maintaining a replica of the production infrastructure.

In the formal literature, we can relate this approach to differential testing, where the oracle is the consistency between the outputs of executions of comparable systems (MCKEEMAN, 1998; GULZAR; ZHU; HAN, 2019). Although an older approach, it has been adapted and used in the context of modern software. For example, the work by Godefroid et al. (GODEFROID; LEHMANN; POLISHCHUK, 2020) employed it for regression testing of REST API services in a cloud environment. A similar approach, known as A/B testing, has been adopted in the industrial context. Large e-commerce companies like Netflix and Amazon apply it to evaluate the impact of changes in business variables. This approach redirects a pool of users to a variant version of the e-commerce site to measure the sales conversion rate caused by specific changes (KOUKOUVIS; CUBERO; PELLICCIONE, 2016; SALEEM et al., 2019; RAHUTOMO et al., 2020; WINGERATH et al., 2022).

In the background (Section 2.5) was described the work of Kallas et al. (KALLAS et al., 2020), which brings the differential approach to testing DSP applications. We identified a correspondence between the efforts of academia and industry in this regard. Although several

reports employ this approach for diverse testing purposes, none has mentioned its use as a solution for performing tests with real data while maintaining data privacy, as described in the grey literature.

Semi-synthetic data. The semi-synthetic data approach, also called hybrid data, automatically generates test data based on real data. In the grey literature, it was reported to generate a telephony data model from real data to create realistic fake data. Although we collected only one practitioner report, there are several works in this direction in the formal literature. Semi-synthetic data generation has been employed to target different application contexts, such as smart grid, such as smart grid (LAHARIYA; BENOIT; DEVELDER, 2020) and health care data (WANG et al., 2021). In general terms, this approach extracts statistical properties from the data and establishes a model of the data pattern in order to feed automatic generators (LI et al., 2016; POPIĆ et al., 2019; TAN; BEHJATI; ARISHOLM, 2019; BEHJATI et al., 2019; MANCO et al., 2022).

For example, we bring the research by Tan et al. (TAN; BEHJATI; ARISHOLM, 2019), which investigates machine learning techniques to generate synthetic, dynamic, and representative test data. The proposed approach is based on building a statistically representative model of a real (reference) population (TAN; BEHJATI; ARISHOLM, 2019). The work considers that Recurrent Neural Networks (RNNs) and Generative Adversarial Networks (GANs) techniques are suitable for synthetic data generation since these neural networks can learn the statistical properties of data sets. The authors emphasize this approach's ability to maintain data privacy since the work was motivated to provide a solution for sharing test data between institutions without compromising sensitive data. Although the technique has been pointed out as a way to preserve data privacy, this statement is not valid in some business contexts where the data distribution pattern is sensitive information.

Specifically promoting using semi-synthetic data to test DSP applications, Grulich et al. (GRULICH et al., 2019) proposed an open-source out-of-order data stream generator. The tool inputs a real data stream, including event timestamps, and simulates arbitrary fractions of out-of-order tuples and their respective delays (GRULICH et al., 2019). The generation process considers histograms and statistics on the temporal distribution of events. This approach is an example of modifying real data to meet a specific testing purpose, in which case the purpose was to test out-of-order events. Kim et al. (KIM et al., 2018) provide the method that generates data for performance testing using the frequency distribution of the real data streams. Komorniczak et al. (KOMORNICZAK; ZYBLEWSKI; KSIENIEWICZ, 2022) also reported employing

semi-synthetic streams generated based on real-world data to evaluate concept drift detectors. Although the focus is slightly different, the concept drift is within the scope of machine learning applications in DSP, and the testing of detectors needs stream data. The evaluation considered synthetic and semi-synthetic data, as the difference in effectiveness between these data sets was a variable to be controlled.

Considering all the works cited, using semi-synthetic data seems to be a promising direction to address privacy issues when testing DSP applications. Additionally, we emphasize that such a technique can only be adopted when real data is accessible to generate these models.

Synthetic data. Synthetic data generation has been reported along with various tools to support this strategy. Generally, synthetic data generators take as input a formal description of the data schema to generate test data. Although the data is random initially and only follows the schema format specification, practitioners have also reported building more elaborate scripts and implementing rules to generate more representative data. For example, these scripts enable the developer to assign weights to the proportion of data generated for each field. It was also reported that some tools, such as Kinesis Data Generator, provide specific features to configure temporal aspects, thus making it possible to define data periodicity through statistical distribution functions with user-configurable parameters.

In the formal literature, Popić et al. (POPIĆ et al., 2019) state that the primary motivation for adopting this strategy is the unavailability of real data for privacy reasons, and on the other hand, the author points out the low ability to generate realistic test data as the weakness of the strategy. In this sense, researchers have directed efforts to develop techniques to improve test data generation. As the grey literature reports, formal research also investigates using statistical resources to improve the generated data (MANNINO; ABOUZIED, 2019). For the same purpose, some works explore information extracted from UML diagrams (JAFFARI; YOO; LEE, 2020; LAFI; ALRAWASHED; HAMMAD, 2021) and the application's symbolic execution (YE; LU, 2021).

In line with approaches based on statistical models, Mannino and Abouzied (MANNINO; ABOUZIED, 2019) proposed the Synner, a synthetic data generator based on specifying the statistical properties of data to generate real-looking synthetic data. With this tool, the developer specifies statistical properties through a data generation language or rich visual resources. Data samples, histograms, and basic statistics, such as mean and standard deviation, are constantly updated in a spreadsheet view as the user interacts with the interface. The tool makes it possible to generate random data or data fitting well-known or custom statistical distribu-

tions. In addition, it also provides a mechanism to add noise and statistical properties to the relationship between fields.

Another path reported in the formal literature is analysing information from UML diagrams to complement and guide the generation of test cases. For example, the approach by Jaffari et al. (JAFFARI; YOO; LEE, 2020) used the activity diagram to generate more representative test data, while Lafi et al. (LAFI; ALRAWASHED; HAMMAD, 2021) employed natural language processing to extract information from the use case description to guide data generation. The analysis of modelling documents is an interesting method to be investigated, as such diagrams describe aspects like time, concurrency, and state characteristics, which are relevant for triggering user-defined events in DSP application tests.

Looking at the data stream context, Ye and Minyan proposed SPOT, a test data generator based on symbolic execution for DSP applications. This approach generates test data by solving path conditions obtained by mapping test inputs during execution and thus identifying the respective execution path (YE; LU, 2021). SPOT also supports temporal issues, such as the interval between data and the data order variation by iterative algorithms. The study evaluation aimed to compare the SPOT symbolic execution approach with the property-based testing approach implemented by the DiffStream (KALLAS et al., 2020) and FlinkCheck (ESPINOSA et al., 2019) tools (both tools were discussed in the Background Section 2.5). In summary, the evaluation showed that SPOT test data has benefits in triggering program failure more efficiently and with better path coverage than property-based solutions. The approach proposed by Ye and Minyan has shown promise in generating higher-quality synthetic datasets for testing DSP applications.

Focusing on machine learning in the DSP context, Iglesias et al. (IGLESIAS et al., 2020) proposed a synthetic data generator named MDCStream. The tool generates temporal-dependent numerical datasets to stress-test stream data classification, clustering, and outlier detection algorithms. MDCStream provides functionalities to create challenges related to non-stationarity, concept drift, and algorithm adaptiveness. The tool allows the developer to configure several cluster-specific characteristics of the generated data, such as the range of dimensions, the number of elements, shape, orientation, statistical distributions, dependencies, feature correlations in the cluster, and cluster rotations. MDCStream is a practical alternative for generating test data for machine learning algorithms that process data streams and is also helpful for discovering new classes, clusters, and anomalies.

Despite being considered an automatic and first-hand approach as a quick alternative to

generating test data, a manual effort to customize the data generation cannot be ignored. The effort may involve configuring various generator parameters and specifying the data's statistical distributions. Furthermore, generating good test data depends on the professional's knowledge of statistical distributions and complex data patterns to exercise certain test target functionality.

5.5.1.5 RQ5. What are the tools and under what circumstances are they used in the context of DSP application testing?

We compiled a list of 50 tools from GLR sources mentioned by practitioners as employed in testing activities in the context of DSP. For each cited tool, we verified its existence and compatibility with the purpose for which it was mentioned. Appendix E contains this list. It includes the tool's name (*Tool Name*), its authorship (*Author*), software license type (*License*), and association with testing approaches outlined in Section 5.5.1.3 (*Main Testing Approaches Support*). The table also distinguishes if a tool is specifically designed for DSP (*Focused on data stream context?*) and the number of grey literature citations each tool received (*Document Citations*). The remainder of this section discusses relevant aspects of these tools.

We identified that 26 tools were developed for the DSP context, while the other 24 are tools for other contexts but also used in some DSP testing activity. For each tool, we looked in the official documentation for additional information about license, authorship, functionalities, and purpose of use, and then we made notes about this information. We also observed that of the 50 tools listed, only four are proprietary, while the others are some variations of open-source licenses. In part, the predominance of open-source software can be explained by adopting a business model around open-source software, as reported by August et al. (AUGUST; CHEN; ZHU, 2021) and Shahrivar et al. (SHAHRIVAR et al., 2018). In this model, companies contribute to developing open-source tools with the community while commercializing software customizations, training, certifications, consulting, support, and infrastructure services. The companies Data Brics and Confluent are examples of this business model, as they provide services centred on open-source DSP frameworks such as Apache Spark and Apache Kafka. Even the case of Confluent is explicitly mentioned in the work of August et al. (AUGUST; CHEN; ZHU, 2021), and it is categorized as an open-source distributor business model by Riehle (RIEHLE, 2021). This model encourages the community to participate in developing and improving various tools around these technologies. The Embedded Kafka tool is an example of a tool with

hundreds of community contributions.

Another aspect observed is the participation of large companies versus individuals in developing and maintaining the tools. The work by Suhada et al. (SUHADA et al., 2021) suggests that companies' motivation to participate in open-source projects comes from the strategy based on Open Innovation to capture new external ideas and solutions. At the same time, individual contributors seek personal and career development. The selected tools with large companies behind the creation or maintenance are JMeter (FOUNDATION, 2021b) from Apache Foundation, Flink Spector (OTTOGROUP, 2019) from Ottogroup, Fluent Kafka Streams (BACKDATA, 2021) Tests from Backdata, Kinesis Data Generator (INC., 2021) from Amazon Incorporation, and Ksql-datagen (INC., 2021d) from Confluent Incorporation. Alternatively, there are also contributions from individuals who initially create the tools and later make them open-source projects to be maintained by the community and supporting companies. Examples of individual contributions include Kafka for Junit (GüNTHER, 2021) by Markus Günther, Spark Testing Base by Holden Karau (KARAU, 2021), Awaitility Library (HALEBY; COMMUNITY, 2021) by Johan Haleby, Ansible (DEHAAN; INC., 2021) by Michael DeHaan, and Sangrenel(ALQUIZA, 2021) by Jamie Alquiza.

In order to identify the use of the tools in the context of testing DSP applications, we analyzed the practitioners' comments extracted from the grey literature and the official documentation regarding the features. Following, Table 23 bring the tools grouped according to their main uses in testing DSP applications, and then we associate them with the challenges raised by RQ1.

5.5.2 Implications for research and practice

This study benefits researchers and practitioners by summarising knowledge fragmented in various grey sources in a single document and including related scientific papers to complement or confront findings. In general, this work's main contribution is narrowing the gap between academia and the industry. Throughout the discussion, we have included observations considering implications for practice and research. The following subsections summarize the most relevant implications.

Table 23 – Tools, mains uses and associated challenges

Main uses of tools in testing DSP applications & Associated challenge	**Tool Reference**
Testing Utilities provide various utility resources to support testing activities, such as integration with other testing tools, command-line interfaces, domain-specific test specification languages, test annotations, and scripts for testing automation in CI environments. Test utilities help to address the challenge *The complexity of DSP Applications* from Section 5.5.1.1, as they bring functionality to deal with DSP characteristics.	ScalaTest(COMMUNITY, 2021b), Jackdaw Test Machine (CIRCLE, 2021), Ducktape (INC., 2021c), Kafka Streams Testing Utilities (FOUNDATION, 2021c), Spark Testing Base (KARAU, 2021), Embedded Kafka Cluster (COMMUNITY, 2021a), Flink Spector (OTTOGROUP, 2019), Flink Test Utils (FOUNDATION, 2021a), Kcat (APACHE, 2021), and Kafka Datagen Connector (INC., 2021d).
Infrastructure automation supports the management of complex test infrastructures, including automation features such as code deployment, network configuration, cloud management, infrastructure as code, command line interfaces, and remote management. In the context of DSP testing, these tools can be used to build CI environments, enable automated test execution, and help developers address the challenge *Testing Infrastructure* presented in Section 5.5.1.1.	Jenkins (JENKINS, 2021), Terraform (HASHICORP, 2021), Ansible (DEHAAN; INC., 2021) and Confluent CLI (INC., 2021b).
Data Generation supports generating test data and provides features such as the generation of data based on the schema registry, the configuration of timestamps, shaping the fake data distribution into real data or custom distribution, and DSL for specifying data properties. These tools help practitioners to deal with the challenge *Obtaining Test Data* discussed in Section 5.5.1.1.	Ksql-datagen (INC., 2017), Kafka Datagen Connector (INC., 2021d), StreamData (LEOPARDI; VALIM, 2021), Avro Random Generator (INC., 2021a), Mockaroo (LLC, 2021), Ecto Stream Factory (BARCHENKOV, 2019), Faker.js (Java Script) (MARAK, 2021), Faker (Python) (FARAGLIA, 2021), and Kinesis Data Generator (INC., 2021).
Time issues provide specific functionality to deal with time issues, such as controlling the current time of the function under test. Such tools help practitioners to deal with the challenge *Time Issues* identified in Section 5.5.1.1.	Awaitility (HALEBY; COMMUNITY, 2021) (library that facilitates asynchronous testing by a DSL to express test expectations), Flink Test Utils (FOUNDATION, 2021a), Kafka Streams Testing Utilities (FOUNDATION, 2021c) and Spark Testing Base (KARAU, 2021).

5.5.2.1 Implications for practice

This document provides helpful information for practitioners to guide decision-making concerning designing and implementing tests in DSP applications. The critical information for practitioners is the list of testing purposes, challenges, approaches, tools, and strategies for obtaining test data. We have included throughout the discussion valuable notes for practition-

ers, such as advantages, disadvantages, and specific details regarding the techniques and tools presented. We also associate the tools and testing approaches with the testing objectives and challenges, which helps the practitioner consider using the presented techniques and tools.

Additionally, this document includes complementary content extracted from the academic literature (usually not consulted by practitioners), such as new testing approaches and tools created in the academic context. For example, regarding test data generation, the formal literature contributes with approaches based on machine learning to improve the quality of generated testing data and preserve data privacy. The cited formal literature also introduces the mutation testing technique to increase the effectiveness of test suites. All of the selected content, which is related to each other, commented on, and linked to the formal literature, has resulted in a robust and reliable document for practitioners. In summary, practitioners have a starting point of study covering many aspects related to DSP application testing in this document.

5.5.2.2 Implications for research

From the research point of view, the work contributes by selecting and summarizing the intrinsic practice content fragmented into several informal documents. Such information is now part of the academic documentation and is available for future research works. Furthermore, we identified research studies corroborating with practices in the industry. For example, when discussing synthetic data generation, we reported grey literature and formal literature efforts on using statistical resources to improve the generated data.

We have also identified research-based techniques that can be evaluated for use in the testing of DSP. For instance, analysing modelling documents can help improve test data generation quality since these diagrams describe aspects such as time, concurrency, and state characteristics. On the other hand, we identified in the grey literature approaches not yet explored in the formal literature. For example, no papers cover equivalent testing techniques based on production data stream mirroring, as reported in Section 5.5.1.4. The approach resembles differential testing with production data but preserves data privacy by an automatic conferencing mechanism called shadow mode.

Finally, we highlight two promising issues for future research that lack formal literature. The first is data privacy in testing activities since it is a topic of increasing importance these days. Although there are several proposed solutions, the problem has several facets to be explored.

The second is fault tolerance testing approaches since these DSP applications should always keep running and recover from runtime failures automatically. In particular, chaos engineering is a testing approach that can be further explored for this purpose. We also emphasize the need for research involving experiments to validate the effectiveness of testing approaches in the context of DSP applications. These studies are essential for systematizing testing processes and consolidating the testing techniques reported in this paper.

5.5.3 Threats to Validity

Although we have conducted our research based on consolidated guidelines for GLR and followed expert recommendations, this study is still subject to validity threats typical of this chosen method. For example, limitations in searching for grey literature and subjectivity and inaccuracies when selecting sources, extracting data, and analysing results. Since the study preparation, we have been concerned with these possible threats to validity, which we have systematically identified throughout the process and promoted mitigation strategies. We organized our threats according to the classification schema proposed by Ampatzoglou and colleagues (AMPATZOGLOU et al., 2020): Study Selection Validity, Data Validity, and Research Validity.

Study Selection Validity.

This category includes threats to the validity of the search process and studies filtering steps. For the search process, we adopted a general-purpose search engine, but the criteria for selecting and ranking results from these tools are unclear and may yield biased or irrelevant results. We seek to mitigate this issue by conducting targeted searches on domains popularly known to contain technical content produced by practitioners, such as Stack Overflow, Medium, and LinkedIn Pulse. In addition, to mitigate possible search engine inefficiencies, we performed snowballing, which added 53 more sources to the review.

At the end of the search process, we identified some content republished in different domains but with slight variations, updates, or increments. While it is common practice for authors to mark them as "reposted" and include the link to the original post, not all do. To identify similar textual content, we adopted the tool WCopyfind[4], which identified 15 cases of duplicate content.

[4] <https://plagiarism.bloomfieldmedia.com/software/wcopyfind/>

In the source selection phase, threats are related to elaborating and applying inclusion and exclusion criteria. For example, conflicting or very generic criteria can hamper the selection process. To mitigate these issues, we built an initial set of criteria based on the field's literature and then held discussions among authors to refine the criteria. To minimize the chances of participants misunderstanding the criteria, we prepared a document of guidelines and examples and conducted a training session and a pilot study. Only after aligning the divergences did the selection process begin. Each source from the initial set was analyzed independently by two reviewers. In case of disagreement, a third reviewer participated in conflict resolution by discussion or a tie-breaking vote. Furthermore, we acknowledge a potential bias in the search string used for RQ3, as detailed in Table 2. This search string, focused on specific types of tests, may have restricted the breadth of our results. While this limitation was identified post-study, we report it here for transparency and completeness.

Data Validity.

This category addresses threats to the validity of the data extraction and synthesis phase. The choice of variables to be extracted may influence the subsequent study phases. To mitigate this threat, we built a set of categories based on terms associated with the RQs and the categories adopted in previous studies. In addition, throughout the extraction, we promoted the refinement and inclusion of newly emerged categories. To minimize interpretation bias in the extraction process, we conducted a pilot study, pre-extraction meetings to align the understanding of the categories, and discussions during extraction to resolve conflicts. Also, the extraction process was paired, in which two authors performed the extraction and mutually reviewed each other's work.

Finally, the first author performed the analysis, and to minimize analysis bias, periodic discussions were held between all authors. Besides, to support the analysis, we considered quantitative data from the categories to identify the most relevant topics and searched the academic literature to confront or confirm the findings.

Research Validity.

Choosing the wrong methodology is a threat to the study as a whole, so we selected ours in consultation with experimental software engineering experts and confirmed that the method is suitable with GLR adoption checklists from Garousi's guidelines. Once the methodology was chosen, we invited a specialist in Grey Literature to participate in the research by supporting the application of the method. Inadequately formulated or non-comprehensive RQs can bias

the research or make it irrelevant. So, the goals of this article were based on topics identified in a previous study we conducted (VIANNA; FERREIRA; GAMA, 2019) based on grounded theory, which included data collected with practitioners through a survey (101 participants) and 12 interviews. In this study, we promoted discussions among authors to define the RQs. Although we selected more than 154 sources, the results may not be generalizable as they are based on a fraction of the existing content. To mitigate this threat, we reinforced our findings by comparing them with the literature related to each find. Study non-repeatability is a well-known threat; therefore, we have documented the protocol employed and described each process step in the article. Documents and data necessary to reproduce the study are available online (VIANNA et al., 2022).

5.6 CONCLUSIONS

This work promoted research into practitioners' knowledge of DSP application testing by addressing grey literature in the field. The GLR selected 154 from the initial 1667 sources, and then the content of interest was extracted by coding technique in order to obtain answers to the RQs.

Below, we list the main results obtained according to the RQs. **RQ1** The challenges for DSP application testing: (1) The complexity of DSP Applications, (2) Testing Infrastructure Complexity, (3) Time issues, and (4) Obtaining Test Data. **RQ2** The main testing purposes identified (1) Functional suitability, (2) Performance efficiency, (3) Reliability, and (4) Maintainability. **RQ3** The main test approaches reported: (1) Performance Test, (2) Regression Test, (3) Property-based test, (4) Chaos test, and (5) Contract/Schema Testing. **RQ4** The strategies adopted by practitioners to obtain test data: (1) Historical Data, (2) Production Data Mirroring, (3) Semi-synthetic data, and (4) Synthetic data. **RQ5** We reported 50 tools used in various testing activities, which are used for the automation of test infrastructure, test data generation, test utilities, time issues, load generations, mocking, and others.

In the discussion, we included observations regarding the findings of the grey literature while associating academic articles in order to complement, confirm or confront them. In short, the review of grey literature showed that the industry has advanced in the practice of testing DSP applications and has proposed techniques and tools to meet their demands. We even reported approaches and tools developed by the industry but not reported in the formal literature that could be the subject of research investigation. At the same time, we identified

that academia has also been promoting advances in the subject. Some academic contributions are aligned with what has been developed by practitioners in the industry. In contrast, others bring complementary advances that can benefit the industry.

Finally, this study is the result of applying proper research methodologies combined with the effort to search, select and analyze content scattered in many informal documents produced by practitioners and published on the internet. The main contribution is the summarization of various knowledge related to the DSP applications test, which is made available in a document organized for consultation by practitioners and researchers in the field.

5.7 SUMMARY

This chapter presented the results of the GLR, which aimed to map and synthesize industry knowledge and experience in testing DSP applications. Searches were conducted using Google's regular search engine and in IT specialized websites, resulting in 154 grey sources. The key findings include:

- Testing Challenges: the complexity of DSP applications, test infrastructure complexity, timing, and data acquisition issues.
- Testing Objectives: functional suitability, performance efficiency, reliability, and maintainability.
- Testing Approaches: Performance Testing, Regression Testing, Property-Based Testing, Chaos Testing, and Contract/Schema Testing.
- Strategies for Obtaining Testing Data: Historical Data, Production Data Mirroring, Semi-Synthetic Data, and Synthetic Data.
- Tools: The study reported fifty tools used for automating infrastructure, generating test data, addressing timing issues, and load generation, among others.

This study contributed to the thesis by providing a deep mapping of practical knowledge, relating it to formal literature, and offering an information base for developing the guidelines.

6 TESTING GUIDELINES FOR DATA STREAM PROCESSING APPLICATIONS

This chapter presents the testing guidelines for DSP applications, constituting the main contribution of this doctoral thesis. The guideline development process is described in Section 3.3. Initially, we introduce the commented guidelines, which include explanations, justifications, and references pertinent to each guideline. Subsequently, we discuss the results of the guidelines' evaluation process.

6.1 THE GUIDELINES

As outlined in the methodology (Section 3), this section introduces the V2 version of the guidelines, a refined iteration of the V1 version following the evaluation stage (refer to Section 6.2). For practitioner convenience, a user-friendly version of the guidelines is accessible at <http://datastreamtesting.space>. This version is succinct, adopts an informal style, and aligns with the language typically found in IT technical publications online. This section presents the same guidelines and explanations that contextualize the recommendations and related references. A brief overview of each guideline is provided in Figure 6.1, highlighting their central elements. The guidelines are organized into seven items, and each is numbered using the format #G[1 to 7], as follows:

Section 6.1.1 - #G1 Information Collection: This section focuses on the collection of pertinent application and business context information to support test development.

Section 6.1.2 - #G2 Establishment of Testing Objectives: This section provides a questionnaire and guidance for identifying and ranking testing objectives.

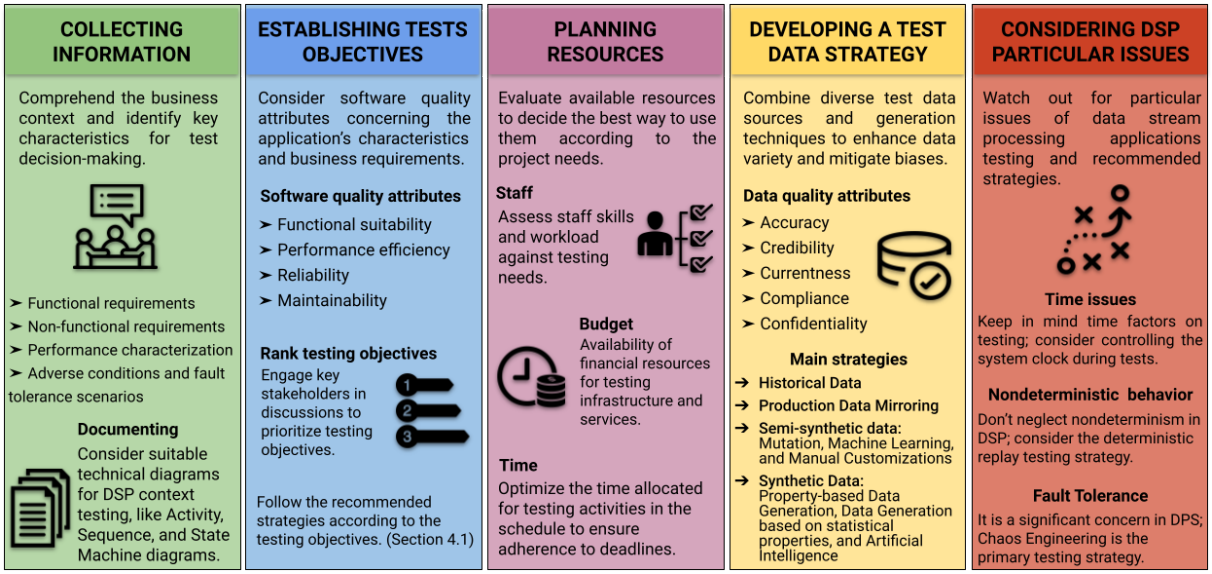
Section 6.1.3 - #G3, #G4, and #G5 Resource Planning: These sections address the planning of human, financial, and time resources, respectively.

Section 6.1.4 - #G6 Development of a Test Data Strategy: This section presents key observations regarding test data effectiveness and outlines various data acquisition strategies.

Section 6.1.5 - #G7 Consideration of the Particular Issues of DSP: This section addresses specific issues in DSP application testing and recommends strategies.

Section 6.1.6 - Provides a scenario exemplifying the use of the guidelines.

Figure 15 – Guidelines steps overview



6.1.1 Colleting Information

In order to address the emerging complexities of testing DSP features preemptively, it is advisable to incorporate test planning into the early stages of project design, a well-established practice supported by literature, including works by Baresi and Pezze (BARESI; PEZZE, 2006), and Taley and Pathak (TALEY; PATHAK, 2020). In this sense, #G1 emphasises the importance of collecting pertinent information for tests, such as comprehending the project's business context, determining parameters needed for test preparation, anticipating adverse conditions and fault tolerance scenarios, and generating technical documentation.

During the requirements elicitation process, it is essential to collect and analyse the most pertinent requirements and application characteristics to support test development (EBERT; RAY, 2020). It allows testers to clearly understand quality requirements, prepare the required type of testing, establish testing priorities, and set the appropriate environment for testing. For instance, the specification of functional requirements can provide input and expected output details for creating test cases. At the same time, non-functional requirements outline desired performance characteristics, response time, expected data volume, and other key information for conducting performance tests (RAHMAN; REZA, 2020). Furthermore, characterising adverse

conditions under which the software must perform is necessary, as such information supports developing fault tolerance tests.

Guideline #G1-D advocates for producing documentation that anticipates information needed to facilitate testing activities, and all testing-relevant information should be documented and made accessible to testers. In this matter, we emphasise the importance of models and technical diagrams to support test development, as they assist testers in comprehending complex software-intensive systems (SCHIEFERDECKER; HOFFMANN, 2012; TIWARI et al., 2021). For instance, UML diagrams such as activity, sequence, and state machine diagrams are helpful as they can express concurrency and state characteristics of DSP applications (AHMAD et al., 2019; SHIROLE; KUMAR, 2021; SHIROLE; KUMAR, 2023).

In some situations, planning tests from the project's inception may not be possible, and the application might already be deployed in production. For these cases, if the existing documentation does not provide the necessary information for test preparation, it is recommended to collect the required information retrospectively.

#G1 - Collect information

A. **Understand the application context:** In the planning phase, get a solid understanding of your project's business context. Identify relevant characteristics that can guide your testing decisions. This could include determining the desired testing objectives and quality characteristics. SBE (Specification by Example) aids understanding of the application's context by developing sample data covering critical use cases.

B. **Gather parameters needed for test preparation:** Collect information needed for test preparation, such as expected inputs and outputs, response time, and expected and maximum data throughput rates. Test-Driven Development (ATDD) promotes a collaborative environment and helps ensure that crucial information for testing is gathered early on, as it involves stakeholders, including developers, testers, and business representatives, in defining acceptance criteria. Section 6.1.1.1 presents a list of suggested information to be collected.

C. **Identify potential issues:** Understand adverse conditions and fault tolerance scenarios. These could include service interruptions, fluctuations in hardware resources, network fluctuations, variations in demand, and potential failures. Identify probable factors that hinder testing, such as timing issues and non-determinism. Scenario-based testing could support the specification of fault tolerance issues by describing scenarios of adverse conditions and related tolerance tests. Business-targeted planning of fault tolerance strategies is essential, as different businesses have varying tolerances for DSP issues.

D. **Document your process:** Testing-relevant information should be documented and accessible to

testers. Produce technical documentation that anticipates information needed for testing activities, such as UML activity, sequence, and state machine diagrams, which express the inherent characteristics of DSP. Such documentation might include details about concurrency and operation states, supporting testers in understanding complex, software-intensive systems. BPMN notation can also be employed to describe DSP workflows, facilitating communication with business experts. The Imixs-Workflow tool integrates a BPMN workflow engine with Apache Kafka, facilitating DSP-based workflow development.

E. **Testing people should get involved:** Include a test specialist, such as a quality assurance expert, during the information-gathering phase in order to ensure the collection of test-relevant information early on, thus minimizing potential issues arising from a lack of such information in later stages.

F. **Balance agility and robust planning:** The agility of the development process should not compromise the depth and quality of testing in DSP projects. For example, inadequate quality requirements testing specifications and insufficient integration testing can result in the discovery of defects at the end of the software life cycle. Large-scale and complex DSP applications require well-mapped requirements for testing.

6.1.1.1 Information to be collected in the initial phases of the project.

Below, we present a list of several information that can be collected in the initial phases of the project. Clearly, the list includes much information that does not apply to all projects, so the recommended use is to filter by the compatibility with the target project.

(1) Identifying message formats and schema structure.

(2) Mapping data stream producers and consumers.

(3) Identifying dependency of third-party services (APIs, data sources): Include details about data source reliability, consistency, and frequency of updates.

(4) Mapping the stream data process that leads to automatic business decisions/actions and manual decisions via dashboards.

(5) Checking whether the business model is compatible with data post-processing in downtime scenarios: Identify alternative data processing strategies for maintaining business continuity.

(6) Mapping response times, throughput rates, and time-out durations.

(7) Data validity: Duration of data retention in cache.

(8) Potential data loss considerations: Include strategies for data recovery and redundancy.

(9) Identifying transaction semantics: Atomicity, Durability, Ordering Guarantees, and Exactly-Once Processing.

(10) Identifying data availability for test purposes (historical, synthetic, or custom example data): Evaluate the representativeness and quality of the test data.

(11) Confidentiality and privacy features: Assess compliance with international standards like GDPR and industry-specific regulations. What are the PII (Personally Identifiable Information) in the data set?

(12) Error Handling and Recovery Procedures: Document how the system handles failures, errors, and retries.

(13) State Management in Stream Processing: Identify how the state is managed, maintained, and accessed in the system.

(14) Scalability and Load Balancing Strategies: Understand how the system should scale and manage varying loads.

(15) Data Transformation and Processing Logic: Detail the logic and algorithms used in processing the data streams.

(16) Version Control and Schema Evolution: How schema changes are managed and versioned over time.

6.1.2 Establish test objectives

In order to plan the test strategy and allocate resources effectively, it is essential to establish clear testing objectives. Guideline #G2 addresses recommendations to support establishing test objectives. Each application has unique characteristics that demand distinct quality requirements. For instance, some business contexts prioritize result correctness over performance, while others may tolerate occasional inaccuracies. We recommend first evaluating and correlating the application's characteristics with the desired quality categories. Establishing testing objectives requires identifying the most critical quality aspects of the application, which depends on understanding the quality from the business perspective involved (PAWLAK; PONISZEWSKA-MARAŃDA, 2020; EVERETT; JR, 2007).

To facilitate the setting of testing objectives, we provide a set of questions associated with the ISO/IEC 25010 model's quality categories (ISO/IEC 25010, 2011). In the GLR study (VIANNA et al., 2023), we identified relevant quality categories for the DSP context: Functional Suitabil-

ity, Performance Efficiency, Reliability, and Maintainability. These questions highlight quality aspects typically relevant to the DSP context, enabling the identification of the most critical elements of the target application context. Additionally, we include pertinent observations, test recommendations, and suggested strategies. We recommend discussing these questions with business analysts on the client side to support the assessment of potential business impacts due to varying quality levels across different categories in the software quality model.

#G2 - Establish Test Objectives.

A. **Evaluate software quality requirements concerning the application's characteristics:** Discuss with stakeholders the *Guiding questions to establish test objectives* (Section 6.1.2.1) to prioritise testing objectives according to quality categories: functional suitability, performance efficiency, reliability and maintainability. When establishing testing objectives, consider the trade-offs between the different categories of the quality model. For example, by focusing on reliability, you may be reducing performance efficiency.

B. **Comprehend quality from the perspective of the business involved in the application:** Ask about the ideal behaviour expected from the application in the context. Map the types of failures and categorize them according to the degree of impact on the business.

C. **Involve stakeholders in the testing objective setting process:** Business analysts, developers, DevOps, testers, clients, and users. Discussing these issues with client-side business analysts will help assess potential business impacts due to varying levels of quality across different categories of the software quality model.

6.1.2.1 Questions to support establishing test objectives

The careful establishment of test objectives is paramount for test planning, as it helps guide strategic choice and aids in prioritising the most pertinent activities. In this section, we present questions to facilitate discussions on test objectives, focusing on various aspects of the ISO/IEC 25010 quality model (ISO/IEC 25010, 2011). The aim is to guide practitioners and project management teams in determining their priorities and strategies. We recommend involving diverse stakeholders such as clients, business context experts, developers, testers, and DSP specialists in these discussions, as outlined in #G2-C.

Functional suitability

Q-A *How critical is the correctness of the results delivered by the application?*

(1) Metrics: Define and establish metrics for assessing the correctness of the application's output.

(2) Prioritisation: Rank the application's features based on the importance and criticality of their correctness. This ranking will guide test planning and ensure that the most crucial features receive proper attention.

(3) System-level Testing: Pay special attention to system-level tests that involve all integrated modules, dependencies, and services. In DSP applications, various system-level factors, such as concurrency, asynchrony, latency, glitches, node crashes, out-of-order, lost, and duplicate messages, can impact the correctness of results.

Q-B *How critical is the accuracy of results that the application must present?*

(1) Metrics: Establish metrics and acceptable thresholds for the accuracy of results. In some DSP application contexts, data may be subject to fluctuations (e.g., sensor data or geolocation) and some degree of variation in result accuracy may be acceptable in others not.

(2) Non-Determinism: Acknowledge that non-determinism in DSP can affect results accuracy. #G7-B brings recommendations to face this issue, such as establishing acceptable thresholds for result variations in the test oracle, conducting multiple test executions observed for functionalities impacted by variations in results, applying statistical analysis techniques to determine if the variations of the results are acceptable, and others (check details in Section 6.1.5.2).

(3) Monitoring: Use application monitoring tools, such as Grafana, to gather result accuracy metrics and facilitate the analysis of the application's performance.

Performance efficiency

Q-C *How critical are the time requirements (e.g., delay, response time)?*

(1) Parameters: Identify the relevant time parameters for the application context and specify thresholds for these parameters based on business requirements.

(2) Clock Control: Manage the application's clock in test environments, as specialised time handling is required for testing purposes.

(3) Simulate real-world: Time factors are vulnerable in real-world conditions, such as network latency, hardware overhead, and communication overhead with third-party services. Thus, the test environment should closely simulate the production environment conditions.

(4) DSP Frameworks: Typically, DSP frameworks provide control functions and interfaces in their test utilities to address time-related issues, such as clock simulation and manipulation of processing time and watermarks. Consider these aspects when selecting DSP frameworks.

(5) Detailed Recommendations: Check details recommendations regarding time issues in Section 6.1.5.1.

Q-D *What is the importance of meeting the requirements of quantities and types of resources the application utilises when performing its functions?*

(1) Resource Estimation: Estimate the available resources for running the application in production (e.g., memory, CPU, server instances, and pay-per-use services).

(2) Network Considerations: Consider the network resources and characteristics required to run the application, such as latency, throughput, and bandwidth.

(3) Testing: Conduct system and infrastructure-level testing using resources allocated for the production environment. Employ monitoring tools to evaluate whether the application operates as expected with the estimated resources.

(4) Monitoring: Utilise hardware resource monitoring tools such as Zabbix, Paessler PRTG (network-focused), Nagios, New Relic (cloud-based monitoring), and Intel Platform Analysis Technology (dedicated hardware monitoring).

Q-E *How critical is efficient resource usage?*

(1) Scaling Strategy: For efficient resource utilisation, strategise dynamic hardware scaling, which is essential in DSP infrastructures that frequently rely on elastic cloud services with variable, demand-proportionate costs.

(2) Testing Scenarios: Establish diverse scenarios involving hardware resource usage (i.e., low, medium, and high demand). Conduct tests to optimise settings for each scenario.

(3) Code Optimisation: Efficient resource usage may also involve optimising application code. Perform tests at all levels to identify resource-intensive operations and optimise them.

(4) Profiling: Use built-in DSP platform features to monitor metrics, track consumer lag, and log requests.

Q-F *How important is it for the application to meet its maximum capacity limits?*

(1) Parameters: Define parameters and characterise the application's operation at maximum capacity. Determine the frequency and duration of maximum capacity exposure.

(2) Stress Test Scenarios: Create maximum stress scenarios and execute stress tests.

(3) Monitoring: Use tools to monitor application performance and hardware resource usage during testing.

Reliability

Q-G *How important is the application's reliability during regular operation?*

(1) Reliability Parameters: Establish parameters to measure application reliability during regular operation.

(2) Defining Standards: Specify what constitutes acceptable and unacceptable situations or issues during regular operation.

(3) System-Level Testing: Conduct system-level testing that simulates regular operating conditions to verify compliance with reliability parameters.

Q-H *How important is the application's operational availability of the application?*

(1) Availability Metrics: Determine the required application availability rates. Specify the acceptable frequency and maximum duration of interruptions.

(2) Fault Tolerance Tests: Perform fault tolerance tests in order to verify the application's availability under various scenarios.

(3) Mitigation Strategies: Identify the causes of application unavailability to propose effective mitigation strategies. For instance, failures in third-party services can result in application unavailability, so preparing a backup service is recommended.

Q-I *How important is the application's resilience to adverse conditions, such as hardware or software failures, network oscillation, and sudden increase in data volume?*

(1) Adverse Conditions Definition: Detail adverse conditions and clarify whether the application can function with limited capabilities under such scenarios, specifying which functionalities would continue or stop.

(2) Performance degradation approach: Specify if the application's performance might deteriorate under adverse conditions and delineate the potential extent of this degradation.

(3) Fault Tolerance Testing: Conduct fault tolerance testing to identify scenarios of adverse conditions where the application can still perform as required.

(4) Chaos Engineering: Employ chaos engineering in testing to assess application robustness, using an experimental setup to simulate failures like network degradation, node crashes, third-party service outages, and reduced computational resources.

Q-J *How important is the application's ability to recover affected data and restore the desired system state in the event of an interruption or failure?*

(1) Interruption Scenarios: Characterise potential service interruption scenarios. Specify the recovery time from outages and whether data from the outage period can be discarded or requires further processing.

(2) Recovery Plan: Establish a disaster recovery plan and processes to promptly reestablish application services.

(3) Fault Tolerance Testing: Perform fault-tolerance tests to verify autonomous recovery mechanisms and application and data integrity following recovery.

Maintainability

Q-K *How important is it for the application to evolve without introducing defects or degrading quality?*

(1) Evolution Plans: Establish application evolution plans. This includes outlining future functionalities, changes in performance needs, and data volume growth.

(2) Regression Testing: Before deploying new releases, perform regression testing for result correctness and potential performance degradation. This process requires automation, skilled personnel, time, and funding.

(3) Contracts Integrity Tests: Conduct thorough tests to confirm the integrity of message contracts, as schema changes frequently trigger regression failures.

Q-L *How important is it to minimize maintenance efforts during application evolution, considering testing resources and the workload of developers/testers?*

(1) Test Automation: Automating tests can significantly reduce the workload on developers/testers while ensuring consistency and extensive coverage in testing, hence minimising maintenance efforts.

(2) CI/CD Pipeline: Implementing a continuous integration and delivery pipeline to catch bugs and errors early in the development process, minimising maintenance efforts.

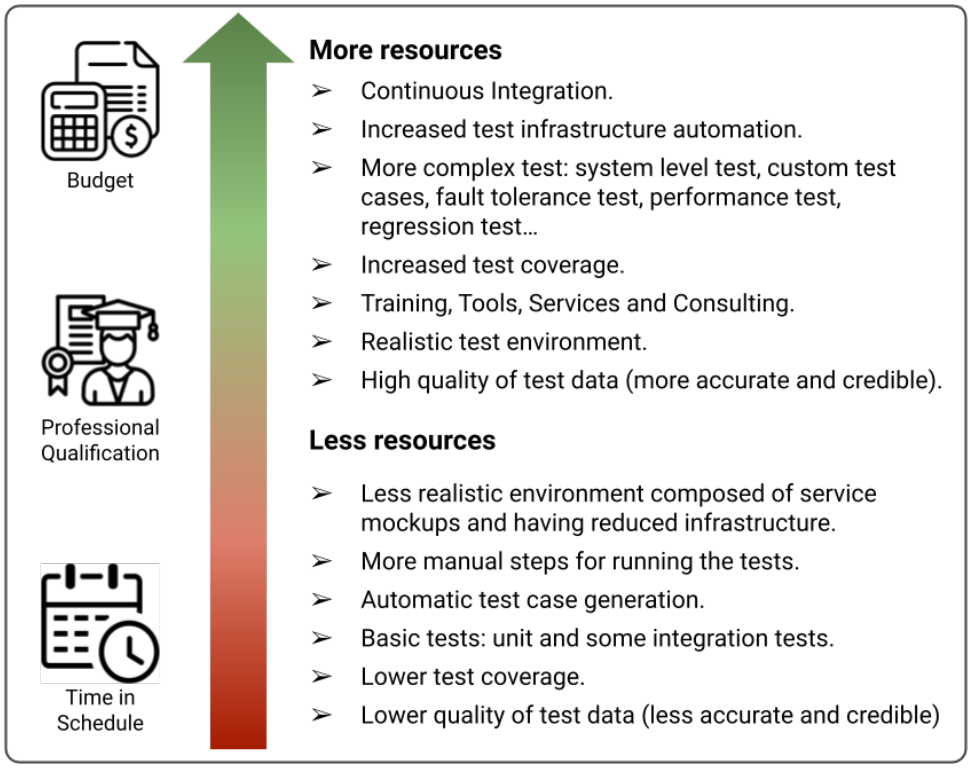
(3) Test Case Maintenance: Consider maintaining automatic test cases, as constant changes in application scenarios require regular refactoring. Focus on automating tests for stable components and avoid creating excessive test cases during project maturation to prevent unnecessary effort wastage.

6.1.3 Resource Planning

The associated costs of testing present a significant concern for the software industry. In some scenarios, testing expenses can account for a significant proportion of resource usage (HYNNINEN et al., 2018; JONES; BONSIGNOUR, 2011; MAILEWA; HERATH; HERATH, 2015). This is especially valid for distributed systems like DSP applications, where testing may involve complex cloud infrastructures, large data processing workloads during performance tests, specialised tools and services and a team of skilled professionals (MOUTAI et al., 2019). In a perfect scenario, we would have access to unlimited resources for software testing, including a dedicated team of experts, a generous timeline, and extensive funding. However, real-world conditions typically impose limitations.

Hence, it is crucial to utilise resources as efficiently as possible to meet the project's quality requirements. A recommended approach involves accurately estimating test resources during the early stages of the project. This facilitates the allocation and prioritisation of activities following the testing objectives. The #G3, #G4 and #G5 recommend proactive planning of the allocation of human, financial, and time resources to prevent test ineffectiveness, inflated project costs, lack of resources for priority activities, and schedule delays.

Figure 16 – Resources overview



We propose three categories of resources: time, financial, and human. It is important to note that these resource categories are interrelated. For instance, a team with limited testing skills might need extra time allocated for learning and a larger budget to provide training and consultancy. Conversely, a highly skilled team, when complemented by adequate funding, can produce satisfactory outcomes even within a constrained timeline. With this in mind, we introduce Figure 6.1.3, which illustrates the compatibility of resources combined into a single variable with testing activities. As resources increase, we can expect more comprehensive test coverage, increased automation in test infrastructure, the ability to develop more complex test cases, and a more realistic test environment. In the following subsubsections, we discuss specific aspects of each resource category.

6.1.3.1 Human Resources

Our previous studies (VIANNA; FERREIRA; GAMA, 2019; VIANNA et al., 2023) underscored that testing DSP applications requires a broad range of competencies, such as expertise in test environment automation techniques, a solid understanding of DSP characteristics and specificities, familiarity with DSP frameworks, knowledge of the target application context, concepts of data science and analytics, and proficiency in cloud technologies. Automation of the test infrastructure is particularly crucial in the context of DSP, with Garousi et al. (GAROUSI; MÄNTYLÄ, 2016) indicating that tester skill level is a significant consideration in the automation of test tasks. Furthermore, a qualitative study evaluating 1000 job ads in the field of software testing underscored the recurrence of soft skills, such as strong communication, critical thinking, team collaboration, problem-solving, and planning and organizing abilities (KASSAB et al., 2021).

Considering the relevance of professionals' skills on test quality within the DSP context, Guideline #G3 focuses on human resource planning. It is essential to ensure that the right individuals are involved in testing and have the necessary skills and expertise to execute the planned testing activities. During the planning stage, we recommend conducting a survey of professionals and assessing the skill sets required to perform each test activity. In case the current test team's skills prove inadequate, team training, additional learning time, and the inclusion of more professionals should be carried out to fill in the skills gaps. Moreover, the available workload must also be considered, as it impacts the extensiveness of test coverage and the execution of labour-intensive activities.

#G3 - Sync Team Skills with Testing Strategy

A. **Skill Sets:** Ensure that the testing activities align with the skills of the testing team. Typical skills for testers in the DSP context may include fundamental DSP understanding, DSP architectures, DSP platforms, distributed systems knowledge, real-time analytics, data modelling, performance, fault tolerance and resiliency testing, and debugging. Additionally, it is crucial to have team members skilled in developing and automating testing infrastructures, such as DevOps, particularly within Continuous Integration/Continuous Deployment (CI/CD) environments.

B. **Training:** If necessary, consider providing training and development opportunities to enhance the test team's skills. Promote a culture of continuous learning within the team to stay updated with evolving DSP technologies and methodologies to anticipate needed skill sets. Motivate participation in DSP workshops, seminars, or conferences for hands-on learning and networking. Training data-specialized quality assurance professionals can be a long-term plan within companies.

C. **Workload:** Prioritize workload allocation based on the criticality of the testing activities. To ensure balanced task distribution, consider the available workload when determining the number and type of test tasks that can be assigned.

D. **Domain Knowledge:** Ensure the team understands the nuances of the specific industry context where the DSP application will be deployed (finance, telecommunications, marketing, social media feeds, multiplayer games, etc). For example, in the context of finance, many rules and security considerations play a significant role in shaping the testing strategy.

Finally, every project has unique characteristics to consider when forming the test team. Besides skills, the team formation process should also consider test objectives and the trade-offs among human, financial, and time resources.

6.1.3.2 Time Resources

Time pressure, as reported by studies from Deak et al. (DEAK; STÅLHANE; SINDRE, 2016) and Kuutila et al. (KUUTILA et al., 2020), can significantly hinder testing activities. Interview-based research conducted by Florea and Stray (FLOREA; STRAY, 2020) revealed that the primary stress source for testers is time scarcity. This pressure can lead to teams rushing, compressing, or even sacrificing crucial activities due to a shrinking timeframe. Therefore, the careful planning and optimisation of time resources become pivotal to prevent delays, relieve time pressure, and avoid contractual breaking. Guideline #G4 provides recommendations for planning and allocating time resources.

#G4 - Time Allocation Planning

A. **Allocate time in the project schedule for critical activities,** including the development of test cases, configuration of environments, creation of test datasets, execution of tests, and analysis of results. In summary, there are two dimensions of time resources: planning time and execution time.

B. **Consider the complexity of the activity,** the time required for test execution, and the number of test cases involved when estimating the time required for each testing activity.

C. **Prioritise activities based on test objectives** and project characteristics, as detailed in Guideline #G2.

D. **Develop the testing schedule by considering the priority and estimated time required** for each testing activity, the available time in the schedule, and the feasible workload.

E. **Employ automated test case generation techniques,** such as property-based tests, to get many basic test cases quickly.

F. **Gradually initiate the automation of the testing infrastructure** as the application gains stability, prioritising the most time-consuming activities.

G. **Accommodate Time-Consuming Tests in Agile Development Processes:** System-level tests, especially those involving large-scale volumes of data such as performance tests, demand more accurate time allocation. Tests involving external dependencies (data sources or third-party services) and teams must be carefully scheduled and coordinated with those managing these services, which generally deviates from the flow of sprints. Therefore, adapt your agile process to accommodate time-consuming tests, particularly in large-scale projects.

During the planning phase, detail the time for each activity and prioritise them according to the testing objectives (supported by #G2 and questions from Section 6.1.2.1). Favour activities with high-priority objectives. Moreover, consider the time required for different types of tests. For instance, custom test cases for complex functionalities may require more preparation time, while automated test case generation can be a faster alternative.

Investing in automation, both in test infrastructure operation and automated test case generation, can optimise time. Garousi et al.'s MLR (GAROUSI; MÄNTYLÄ, 2016) highlights that time-saving is a key factor influencing test automation decisions. Consider employing the property-based testing technique for test case generation, which offers good coverage with relatively low time investment.

Running DSP applications in system-level tests can be time-consuming due to the operational work needed to configure different services for each test round. Despite the initial time and effort required for infrastructure automation, it can yield substantial time savings in

the long run. We recommend a gradual approach to building the automation of the testing infrastructure, initially developing scripts for the most critical activities.

6.1.3.3 Financial resources

Financial resources play a crucial role in testing DSP applications. These resources are critical for various testing activities, including hiring and training professionals, purchasing consultancy services, acquiring software and tools, contracting outsourced services, hiring hardware, and maintaining and evolving the test infrastructure.

It is crucial to consider financial resource allocation strategies early in the project planning phase to ensure sufficient resources and pave the way for testing activities. For this reason, we present Guideline #G5, which provides insights for planning the allocation of financial resources. The financial resource allocation strategies should take into account the test objectives priority established following Guideline #G2.

#G5 - Plan Financial Resource Allocation

A. **Test Objectives Alignment:** Ensure that the allocation of financial resources aligns with the testing objectives outlined in #G2, and then prioritise investments that drive the most significant impact on achieving these objectives.

B. **Comprehensive Costing:** Account for all potential costs related to the testing process, including infrastructure (hardware, cloud services), personnel (in-house tester salaries, training fees), consultancy contracts, contracting of services and tools (e.g., test frameworks and third-party services used in tests), and the ongoing maintenance and evolution of the testing infrastructure. We highlight that testing expenses are particularly significant in large-scale projects, especially if it is necessary to replicate large and complex infrastructures in test environments faithfully.

C. **Cost-Reduction Strategies:** Implement strategies to minimise costs, consider strategies such as infrastructure automation, optimising the use of on-demand paid hardware resources, employing open-source tools, and utilising mock infrastructure and services. Evaluate the cost-effectiveness of cost-reduction strategies against test efficacy.

D. **Policies on the Use of Cloud Resources:** Machine allocation for testing on cloud-based testing environments can represent a high financial cost to the project. In this sense, we recommend establishing internal policies to schedule and execute financially intensive tests.

E. **Financial Impact of Load Testing:** Usually, load testing is a financially demanding activity requiring an infrastructure similar to the production environment. This is particularly crucial in projects with

very restrictive Service Level Agreements (SLAs). Be prepared to identify whether your project will require resources for this type of testing.

F. **Test to Validate the Architecture:** It is essential to validate architectural decisions in the DSP context, especially in large and complex projects where adjusting the architecture at advanced phases can incur considerable costs.

It is worth remembering that the DSP application testing infrastructure, which typically involves cloud-distributed infrastructures with numerous microservices, can be financially demanding (ASSUNCAO; VEITH et al., 2018).

Cloud testing infrastructures, which can mirror the live environment or be scaled-down replicas temporarily employed for test runs, present an opportunity for cost savings (MOUTAI et al., 2019). The DevOps development philosophy, which encourages the automation of testing infrastructure (BATTINA, 2020), is a particularly effective strategy. Using automation tools and scripts to manage hardware and software resources, conduct tests, and collect and assess results may bring significant results in resource optimisation. However, this approach requires skilled professionals capable of implementing test automation strategies that reduce resource consumption without compromising test effectiveness.

6.1.4 Developing a test data strategy

Test data must effectively identify application defects, confirm feature functionality as expected, and ensure adherence to non-functional requirements. Guideline #G6 focuses on test data, providing relevant insights and recommendations to assist the development of testing data. Following this, Section 6.1.4.1 explores data quality characteristics relevant to the DSP application testing context. Lastly, Section 6.1.4.2 enumerates, describes, and discusses relevant aspects of recommended strategies for test data acquisition.

#G6 Develop a test data strategy

A. **Consider data quality attributes to assess your test data set:** Accuracy, Credibility, Currentness, Compliance and Confidentiality. Check Section 5.1 for detailed descriptions ISO/IEC 25012 data quality attributes. The Great Expectations tool is recommended for evaluating the test data quality, especially synthetically generated. Furthermore, this tool can monitor and issue data quality alerts for the production pipeline.

B. **Combine diverse test data sources and generation techniques** to enhance data variety and mitigate potential biases associated with individual techniques.

C. **Do not over-rely on historical data,** as its effectiveness might be limited due to many never manifested defects in production. Historical data's currentness may also be compromised, as it does not exercise new features and could become incompatible with future application versions.

D. **Improve historical data efficiency by utilizing semi-synthetic data generation strategies** such as mutation, machine learning and manual customization. Check Section 6.1.4.2 for details.

E. **Maintain vigilance over the data schema** by employing tools such as Apache Avro to prevent contract breaks in your pipeline and Apache Delta to manage and minimize issues throughout the schema's evolution.

F. **Adhere to privacy regulations,** such as GDPR, during test data handling to prevent legal issues. Utilize approaches like machine learning and shadow mode running to safeguard confidential information when mirroring production data, as outlined in Section 6.1.4.2. Some examples of anonymization techniques are redaction, replacement, masking, crypto-based tokenization, bucketing, date shifting, and time extraction. However, we emphasize that these processes are labour-intensive and time-consuming, as scripts and procedures must be tailored for each case.

G. **The property-based data generation is a cost-effective approach,** as it is fast and easy to apply, making it suitable when time and resources for generating test data are limited.

H. **High-quality documentation is a valuable asset when real data is unavailable.** Natural language processing algorithms can be employed to extract information from documentation to supply automatic approaches with relevant parameters, generating more accurate synthetic data.

6.1.4.1 Data Quality Characteristics

ISO/IEC 25012 (ISO/IEC 2008) outlines data quality attributes in software development. In this section, we selected the most relevant characteristics to address test data quality for DSP application testing.

(1) **Accuracy:** Data accurately represents the intended attribute values of a concept or event within the application context. DSP applications' operations and filters can be highly sensitive; testing such functionalities relies on the precision and relevance of values corresponding to the variable's concept.

(2) **Credibility:** This concerns the data's authenticity or whether it is believable as real-world data from the application's usage context. Addressing credibility in the DSP context is more complex due to additional factors, such as the temporal distribution of data, frequency of variable values, and intervals between messages. Furthermore, the 4Vs of

Big Data (volume, velocity, variety, and veracity) introduce unique aspects to stream data.

(3) **Currentness:** This relates to the data's age validity. Data characteristics can change over time in the DSP context, making them ineffective for testing (similar to how concept drift affects ML algorithms). Furthermore, application updates may cause data to be incompatible with newer application versions. Establishing policies for data lifecycle management can help address these issues.

(4) **Compliance:** This involves data adhering to standards and conventions. DPS applications can consist of numerous entities interacting through various message patterns. At this point, test data must be compatible with the data structures in use. Adaptations may be necessary, and message schema management tools provide functionalities to provide compatibility between different message structures. In addition to structure, there are issues with standards and formatting not captured by schema management, like incompatible text encodings, GPS data coordinate patterns, variations between metric and imperial systems, and conflicts between signed and unsigned data.

(5) **Confidentiality:** This concerns protecting sensitive information. DSP applications often operate in contexts involving confidential or sensitive data, such as personal, geo-location data, and financial data. Strategies to maintain the confidentiality of real data include anonymization, masking, using artificial intelligence techniques, and mirroring production data in shadow mode (details in Section 6.1.4.2).

6.1.4.2 Test Data Strategies

Following, we present the primary strategies for acquiring test data applicable to the DSP context, along with pertinent observations and recommendations for each. These strategies were identified through the GLR. The following subsubsection summarizes the essential aspects of each strategy.

Historical Data. Historical data are assumed to be real data extracted from the application in production, which should also include metadata regarding temporal properties such as origin timestamps, network delay, and processing time. Historical data provides an initial model of data streams in the production environment; however, it does not guarantee a reliable

test oracle. Historical data may consist of inputs without expected outputs, and any existing outputs could be unreliable. Thus, generating and validating outputs for building a test oracle is necessary; this activity depends on documentation that includes data characterization and examples of correct outputs. Even with an extensive set of historical data, it may not provide comprehensive coverage for potential future bugs due to the application's complexity and the infinite possibilities of latent bugs that may still arise in production. Moreover, historical data does not exercise new functionality and may be incompatible with new message schemas. Conceptual characteristics of the data may also change over time, diminishing the effectiveness of historical data in simulating real conditions (similar to concept drift in ML).

Production Data Mirroring.

In this approach, replicas of the input data stream are redirected from the production environment to the testing environment, enabling the application to be tested with real data. This strategy allows the detection of critical failures that could disrupt the application's execution and facilitates the comparison of performance parameters and results accuracy across different application versions. Additionally, this strategy can be employed while ensuring privacy and data security through mechanisms such as shadow mode, which conducts automatic verification of parameters and execution results. Shadow mode, a finding from the GLR, is discussed in more detail within the article. Implementing this technique requires the availability of resources to replicate the infrastructure and skilled professionals to build the verification mechanism.

Synthetic Data

This type of data is generated automatically without using real data in the process. Techniques within this category range from simple random data generation to more sophisticated approaches based on Artificial Intelligence. Below, we present the main techniques:

(1) **Property-based Data Generation:** generates data by exploiting the message contract properties of the data stream. The technique can generate immense amounts of data, which can be refined by a process called shrinking, where the objective is to find the minimum data set to manifest the failure. this is a low-cost technique that requires effort to prepare and is, therefore, quick and easy to apply. The following tools support this technique: FlinkCheck, ScalaCheck, StreamData, and Ecto Stream Factory.

(2) **Statistical Properties-Based Generation:** results in more accurate data by configuring the generated data's statistical distribution and the temporal variations in the data

distribution. This approach requires a solid understanding of mathematics and statistics. Custom scripts can be built using statistical libraries like Scipy in combination with fake data generation libraries. Pay-per-use services like Mockaroo offer ready-to-use solutions where users simply specify data generation properties.

(3) **AI-Based Generation:** Natural language processing algorithms can extract information from project documentation, providing valuable input for machine learning algorithms in generating more meaningful data.

Semi-synthetic data.

This approach combines synthetic data generation with real-world data and may also involve customization steps. This strategy can be beneficial for increasing test coverage of historical data and adapting data to address more complex test cases.

(1) **Mutation:** This process generates new data by slightly altering the values of existing data. These minor modifications increase data diversity, reflecting the variability and complexity of real-world data while preserving the essential characteristics of the original data.

(2) **Machine learning:** This method employs machine learning algorithms to extract features from an existing dataset and create models for generating new data. It can be used to expand limited test datasets, produce data variants to enhance test coverage and preserve real data privacy. Implementing this technique relies on skilled professionals in the field of machine learning.

(3) **Manual customizations:** This process involves refining data to test functionalities that depend on a specific set of conditions, which synthetic data may not be able to trigger. Customizations can be achieved through iterative script execution processes and manual adjustment of generation variables until the desired result is achieved. This approach requires a professional who understands the application's context and has access to comprehensive documentation detailing the functionalities.

6.1.5 Particular aspects of Data Stream Processing

This section addresses the specific aspects inherent in DSP applications that must be considered in test planning and execution. Within Guideline #G7, we highlight three particular

issues to testing in the DSP context: timing issues (#G7-A), the non-deterministic nature of distributed DSP (#G7-B) and fault tolerance (#G7-C). For each element, we provide a brief explanation followed by pertinent observations and suggested testing strategies.

#G7 Be aware of particular issues in data stream processing application testing

A. **Keep in mind time-related factors during testing,** such as message ordering, timeouts, delays, and response time requirements. We recommend practices like controlling the system clock to simulate the production environment's timing characteristics, accelerating the clock to speed up testing, and adjusting the processing time interval to maintain a balanced result precision and computational load. Test the system's ability to handle out-of-order data, a common occurrence in DSP applications. Utilize a checkpoint system to preserve consistent snapshots of all timer states. Consider the clock control features present in stream processing platforms and tools like the Awaitility library to synchronize operations during testing. For a more comprehensive discussion on time-related concerns, please refer to Section 6.1.5.1.

B. **Do not neglect the non-deterministic behaviour of DSP,** which can cause the application to deliver varied results across multiple executions. Recommended approaches include the deterministic replay to identify and manage non-deterministic variables during testing and the creation of test oracles by setting acceptable thresholds for result variations. Cogitate adopting chaos engineering to check the system's robustness under non-deterministic conditions. Testers should also be aware of common non-deterministic bugs, such as race conditions, ordering issues, state inconsistencies, and problems related to lost, duplicate and delayed messages and timeouts. Additional information concerning non-determinism matters can be found in Section 6.1.5.2.

C. **Fault tolerance is a significant concern in DSP applications.** In this sense, Chaos Engineering is the primary strategy for testing fault tolerance and system recoverability. Identifying appropriate fault tolerance mechanisms and testing whether they work suitably is also essential. Common fault tolerance mechanisms in the DSP context are infrastructure redundancy, scalability of hardware and network resources, service redundancy, operation downsizing, application version rollback, operations rollback, and message contract compatibility. Section 6.1.5.3 provides details on fault-tolerance mechanisms.

6.1.5.1 Time Issues

Testing time-related aspects in DSP applications presents a significant challenge due to the importance of timing in aspects such as message ordering, late events, timeouts, response time requirements, timed aggregation or joining operators, data lifetimes in stateful operators, concurrent processing, and network characteristics like latency. Testing DSP applications requires

understanding how time operates in production and testing environments. In production, the timing characteristics of data stream messages are inherent to the business context. However, these timings will differ in the test environment and can affect the validity of test results. Following, we propose some recommended strategies for handling specific timing issues.

(1) **Clock Simulation:** Controlling the system clock in the test environment is essential to emulate the time aspects of the production scenario as closely as possible. This feature is particularly useful for testing temporal windows, as the number of messages in each window can vary depending on the intervals between messages. The same applies to testing algorithms and functions that evaluate time factors. Be aware of how the event generation frequency in your test environment could influence test outcomes.

(2) **Speeding up the clock:** Speeding up the clock in the test environment is a valuable strategy to minimize the duration of tests. Many stream processing platforms provide functions that allow for clock manipulation, including skipping certain test cycles, generating artificial watermarks, and configuring event timestamps to match an accelerated timeline. However, it's necessary to balance speed with result accuracy when employing this approach. Excessive acceleration of the clock may lead to losses in precision, which could obscure potential issues in the application. Essentially, if the test clock runs too fast, bugs tied to specific timing scenarios may go undetected.

(3) **Adjusting the Processing Time Interval:** Calibrating the processing time interval is also crucial in testing DSP applications. Longer intervals can yield inaccurate results, while shorter intervals result in more frequent updates and more accurate results but at the expense of increased computational overhead.

(4) **Checkpointing Mechanisms:** This is a valuable mechanism that periodically stores consistent snapshots of all states in timers and stateful operators, including connectors, windows, and any user-defined state. Platforms like Apache Flink come with built-in checkpointing features. This approach provides valuable state data to reproduce conditions in specific testing scenarios.

(5) **Testing Asynchronous Operations:** Asynchronous operations are particularly tricky to test, as firing an event may involve timeouts and manipulating states stored in stateful operators. Testing these operations requires special attention due to their non-linear

execution, and it's crucial to ensure that your testing environment can accurately monitor, manage, and validate these operations. One recommended tool for handling asynchronous operations is the Awaitility library, as it supports testing asynchronous operations by synchronizing these operations during the test, enabling the test to wait until certain pre-set conditions are met.

6.1.5.2 Non-determinism

The non-determinism of DSP applications adds a layer of complexity to the testing process. The potential for different results to be produced in multiple runs complicates the result's consistency and the establishment of accurate test oracles. Non-determinism manifests at the system level when numerous variables contributing to non-determinism are present simultaneously. Several characteristics intrinsic to DSP applications, such as stateful operations, window-based operations, concurrent operations, and out-of-order messages, make applications inherently non-deterministic. DSP application functionalities often involve complex processes, encompassing multiple sequential and concurrent transformations. They may also rely on temporal windows and keep numerous shared states (DÁVID; RÁTH; VARRÓ, 2018). Such functionalities tend to exhibit variation in their results when subjected to fluctuating network delays or changes in message order from data producers, complicating the construction of reliable test oracles.

Based on our GLR, we advocate for the deterministic replay approach. This strategy aims to identify and manage the variables that contribute to non-determinism, allowing for greater control during test execution (BORODAY; PETRENKO; GROZ, 2007; LEESATAPORNWONGSA et al., 2016; DIAZ; SOUZA; SOUZA, 2021; CHEN et al., 2015; BABAEI; DINGEL, 2021). We recommend conducting several system-level tests to secure consistent and accurate results over time. These should subject the application to varying loads and delays, thus ensuring result reproducibility and accuracy. The construction of test oracles initially necessitates the determination of acceptable thresholds for result variations. Then, statistical methods should be applied to ascertain whether the observed variations adhere to the predetermined limits. Furthermore, adopting chaos engineering practices could help identify the system's behaviour under non-deterministic conditions. Lastly, automated testing tools should help to conduct repeated tests under different conditions efficiently. These recommended practices should assist in overcoming the challenges of non-determinism in DSP applications. Below, we summarized the main

testing recommendations related to non-determinism.

(1) **Test Oracle Construction:** Establish acceptable thresholds for result variations during test oracle construction. Then, statistical methods can be used to validate whether the observed variations align with the predetermined limits. The tool Great Expectations provides a feature for setting data variation thresholds in order to monitor data quality.

(2) **Deterministic Replay:** This approach involves managing and identifying variables contributing to non-determinism, thus providing better control during test execution.

(3) **Chaos Engineering:** In order to check results consistency, experiment repeated tests run application tests under non-deterministic variables like out-of-order messages and network and data volume oscillation.

(4) **Consider Typical Bugs Related to Non-Determinism:** Watch out for typical non-determinism bugs, such as race conditions, ordering issues, state inconsistencies, lost, duplicate or delayed messages, and timeout-associated bugs.

(5) **DSP platforms provide features to deal with some aspects of non-determinism:** First, event time processing and watermarks manage out-of-order and late-arriving data, allowing the treatment of issues arising from delays in messages caused by oscillations caused by non-deterministic factors. Second, state management and exactly-once-processing semantics maintain consistency in processing.

6.1.5.3 Faul Tolerance

DSP applications run uninterruptedly 24/7 operations valuable to the company's businesses. Therefore, this application must keep running in adverse conditions with disaster recovery capabilities to self-recuperate from crashes. In addition to application construction failures, such as a bug resulting from a programming error, we should also be concerned with failures arising from glitches and interruptions or oscillations of computational resources, networks and third-party services. For an application to be fault tolerant, it is necessary to build tolerance mechanisms. Such mechanisms involve first the autonomous ability to identify failures when they occur or predict failures about to emerge and then the action to prevent or reverse failures.

In this context, the testing aims to verify the fault tolerance mechanisms, ensuring they can operate correctly under abnormal scenario conditions. Chaos Engineering plays a significant role in testing fault tolerance and system recoverability. It involves subjecting the DSP application to a controlled set of abnormal scenarios and verifying whether the system can restore checkpoints and resume regular functionality. Tools like the Thundra, Chaos Monkey and WireMock allow for injecting errors, network oscillations, randomly terminating service, and simulating a range of possible failures to assess their impact. This process provides valuable insights for improving fault tolerance and recovery mechanisms by identifying potential weaknesses in the system. Below, we list and describe some examples of fault tolerance contingency that can be adopted in the context of DSP applications:

(1) **Infrastructure redundancy:** This strategy involves having redundant backup servers ready to take over in the event of primary server failure. DSP platforms often provide easy-to-use integrated replica features. However, this strategy may be financially costly, and budget availability must be evaluated. Furthermore, the number of replicas increases system latency due to synchronisation overhead.

(2) **Scalability of hardware or network resources:** Upon detecting an increase in demand, adjust resources to keep the service running within specified performance requirements. Elastic scalability is a feature of cloud infrastructures that performs this task. This mitigation action must consider the strategy for allocating financial resources. To scale up a broker cluster horizontally, consider the scaling capabilities of other services like APIs, consumers and producers to ensure that real-time processing is not affected.

(3) **Service redundancy:** This strategy involves having alternatives for backup services that are automatically activated when a third-party service fails. For example, backup providers can easily replace SMS, encryption, and freight calculation services if they become unavailable.

(4) **Operation downsizing:** In the face of a failure that cannot be automatically circumvented, the impacts of different mitigation strategies must be evaluated, such as temporarily interrupting the service, deactivating certain functionalities, or continuing to operate under extraordinary conditions. The mitigation strategy depends significantly on the application's context and will be tied to business decisions. For example, an e-commerce company can extraordinarily pre-authorize purchases from frequent customers

when a particular payment service is temporarily offline. Conversely, a bank would prefer to turn down sensitive services when some security features are offline.

(5) **Version rollback:** In the face of unstable behaviour or failures after the release of a new version, a mechanism for easy version update rollback is recommended to quickly contain problems in the production environment.

(6) **Operations rollback:** when an operation has been delivering incorrect results due to a bug for some time. First, it is necessary to identify the period when incorrect results were delivered to reprocess them with a backup infrastructure. In addition, issues related to legal aspects and the business context must be evaluated, as reprocessing operations a posteriori can be useless or harmful. For example, credit card companies attend legal procedures for reversing and correcting incorrect charges.

(7) **Contracts compatibility:** Large and complex DSP applications can have complex data schema with many message contracts. Updates can cause contract incompatibilities, especially if many modules interact and several teams promote changes in these modules and third-party services. Mitigation involves maintaining backward compatibility with contracts until contract updates propagate. Among the solutions in this context, we mention Avro, which supports compatibility for evolving contracts over time.

(8) **Fault Tolerance Tools:** Chaos Monkey, developed by Netflix, is acknowledged for introducing random infrastructure failures. WireMock can simulate faults in HTTP-based services like APIs by mocking responses. Jepsen performs black box testing and fault injection on unmodified distributed data management systems. Thundra provides error injection capabilities, allowing for a more controlled testing environment where specific failure modes can be simulated and analysed.

(9) **Data Loss Strategy:** In addition to the traditional mechanisms provided by DSP platforms to prevent data loss, another solution consists of synchronizing all incoming messages in a data lake. However, this approach might not be suitable for high-volume and intensive data scenarios.

6.1.6 Example scenario

To illustrate the application of the guidelines, we developed a test plan for a stream processing context based on these guidelines. The scenario, titled "Stream Data on an Electric Scooter Rental Application" is grounded in a real-world case derived from an interview with participant P10 of our exploratory study (VIANNA; FERREIRA; GAMA, 2019).

6.1.6.1 Colleting Information

We started planning the testing strategy with #G1, which drives the information-gathering phase to understand the application's business context and identify important information for testing efforts.

#G1-A Context: The application processes stream data from each scooter in the fleet. The stream data includes real-time GPS coordinates, battery level status, and events related to the scooter's usage (e.g., ride start/end). Features related to DSP include scooter release and blocking events (due to low battery and maintenance), geofencing areas where scooters are permitted for use and parking, monitoring data for battery levels, and real-time location data. The performance requirements are not stringent, as there is a certain tolerance for response time. The volume of data does not significantly scale since there is a fixed number of scooters. Minor inaccuracies, slight delays in data, out-of-order data, and loss of some location data do not constitute serious issues. However, the availability of the stream processing service is critical, as it would affect the scooter rental operation.

#G1-B Collecting Parameters for Testing: The expected response time for the stream processing operations during regular operation is 3 seconds. The data volume refers to 10,000 scooters distributed in multiple cities, each transmitting a stream of location data and battery-level information.

#G1-C Characterising Adverse Conditions and Fault Tolerance Scenarios: The primary concern is the intermittent nature of mobile internet, which can result in communication delays and feature timeouts.

#G1-D Producing Testing Documentation: Activity, sequence, and state diagrams are appropriate to represent relevant aspects for testing in this scenario.

6.1.6.2 Establishing test objectives

As proposed in item #G2-A, we established and prioritised the test objectives with the support of the questions from Section 6.1.2.1.

High Priority

Question A - Correctness is critical for user experience, particularly for remote locking and unlocking scooters.

Question H - Operational availability is critical to the operational efficiency of the business, as outlined in the scenario context.

Question G - It is a high priority to ensure the applications' reliability during regular operation.

Question C - Meeting the application's time requirements is relevant for user experience.

Given the high priority of these objectives and related recommendations, testing efforts should primarily focus on verifying correctness, time requirements, and reliability through system-level tests that involve all integrated modules, dependencies, and services. Concerning operational availability, it is appropriate to conduct fault tolerance tests using chaos engineering approaches to validate the application's ability to keep running under atypical conditions. Building infrastructure and service redundancy mechanisms would be advisable due to the high priority of reliability.

Medium Priority

Question B The accuracy level of the geolocation data is valuable, but it is not a critical matter.

Question D Given the limited availability of resources, it is suitable to meet the quantities and types of resource requirements.

Question E Efficiency in employing resources is important for cost-effectiveness due to limited financial resources.

Question I Performing as required despite adverse conditions is relevant, primarily due to mobile network intermittence.

Question J Quickly recovering to the desired system state in the event of failure is valuable, as the states and data of ongoing scooter ride operations need to be recovered following failures.

Question L Minimising costs and workload for test maintenance during project evolution is pertinent due to the scarcity of financial resources and the demanding workload of professionals.

Considering medium-priority objectives, system-level tests are recommended to verify real-time location accuracy, battery status, and geofence limit. As outlined in #G7-B, when assessing the accuracy of results, one must be aware of the non-determinism factor in DSP. In this case, it would be appropriate to build test oracles with limits on result variations as well as to adopt the deterministic replay approach (see details in Section 6.2).

Due to the need for efficient use of hardware resources and adapting the application to meet the resource requirements, it is recommended to monitor the use of hardware resources during testing and then promote optimisations for resource-intensive operations. Moreover, regarding resource constraints, effort should be made to minimise test maintenance workloads to reduce costs by gradually implementing automatic tests and CI/CD pipelines.

To ensure the applications performed as required despite adverse conditions, chaos engineering practices were applied to evaluate the application's robustness and identify scenarios in which the application could still perform as required, even under adverse conditions. Regarding data recovery and state restoration after an interruption, it was appropriate to establish a disaster recovery plan first and then perform fault-tolerance tests to verify autonomous recovery mechanisms and application integrity following recovery.

Low Priority

Question K Modifying the application without introducing defects is opportune, but it is not a priority.

Question F The application's performance at maximum capacity limits is not a significant concern. The data volume is predictable because the maximum number of scooters is constant.

Testing activities related to low-priority objectives come last. Concerning regression bugs, it is necessary first to establish plans for application evolution and then conduct regression testing to verify correctness, assess any degradation in performance, and confirm the schema integrity. The regression test suite would evolve, beginning with the most critical functions, and its progress will also depend on the maturity of the CI/CD pipelines. Since there will not be sudden fluctuation in data volume, performance at maximum capacity is not a priority, so stress tests should be conducted only if the scooter fleet expands.

6.1.6.3 Resource Planning

In general, resources are limited in this scenario. Comments on #G3, #G4, and #G5 regarding human, financial, and time resource planning will be below.

#G3 Human Resources: The team consists of five skilled developers, but their testing experience in the DSP applications is somewhat limited. Concerning the workload (#G3-C), none of the team members is exclusively dedicated to testing; instead, developers share the workload between development activities and testing based on demand. As proposed in #G3-A, when evaluating the team's skills, it is clear that at the project's beginning, developers are more proficient at implementing more traditional test approaches, such as unit tests. However, following #G3-B, it is recommended to provide learning opportunities for the team to study and employ specific test techniques pertinent to the DSP context.

#G4 Time Resources: The deadlines are short, as a beta version is already in production. As indicated by #G4-A-B-C-D, the time dedicated to testing must be allocated into the schedule, estimating the duration of each activity and prioritising the most relevant ones based on test objectives (which will be established with #G2). Particular attention should be given to test automation and generative techniques for test data creation, as suggested by #G4-E-F; these

are valuable recommendations given the constraints of short deadlines and limited workload.

#G5 Financial Resources: Financial resources for contracting testing services and infrastructure are limited. #G5-C provides suggestions for reducing costs applicable in these scenarios, such as automating test infrastructure, adopting open-source tools, and utilising mocked infrastructure and services. #G5-B advises considering the future costs of maintaining and evolving the test infrastructure. The recommendations are especially pertinent in this scenario due to the limitation of financial resources.

6.1.6.4 Test Data Strategy

A small set of anonymised historical data is available for testing purposes. According to data quality attributes from #G6-A, test data accuracy and credibility are the most relevant for this scenario. Therefore, data must reflect credible and accurate scooter parameters like location and battery level. The GPS data must realistically simulate the typical riding behaviour of a scooter, while battery data should mimic various drain patterns, such as gradual and abrupt drops.

As recommended in #G6-C, one should not overestimate the effectiveness of historical data for testing, especially when the quantity of data is limited. Therefore, as proposed in #G6-D, we first should focus on improving the efficiency of historical data through semi-synthetic data generation strategies. Given the limited resources available, data mutation is a straightforward implementation technique. At the same time, manual customisation can occasionally be employed for test cases related to critical operations, such as locking and unlocking scooters.

Later, we opted to generate synthetic data to diversify the test data generation technique and thus mitigate potential bias, as pointed out in #G6-B. Property-based data generation is an appropriate technique for this scenario because it is quick and easy to apply and provides cost-effective coverage, as indicated in #G6-G.

6.2 GUIDELINES EVALUATION AND RESULTS DISCUSSIONS

When analysing the data, we considered the focus group and the survey inputs together. Generally, there is no antagonism between the opinions expressed in the survey and those in the focus group. The comments typically confirm or complement the information provided. However, a difference in the tone of critical opinions was noted. Criticisms in the survey were more direct, whereas, in the focus group, criticisms were presented more diplomatically. Therefore, regarding the triangulation strategy established in the methodology, we can place more confidence in the validity of the opinions given by practitioners during the evaluation. We note that the version submitted for evaluation is version V1 in PDF format, which is in Appendix I.

The remainder of this section is structured as follows: Section 6.2.1 offers a concise overview of the evaluation participants' profiles. Section 6.2.2 outlines the conduction of the focus groups. Subsequent sections present each guideline's feedback and a brief but relevant discussion. Finally, Section 6.2.10 provides feedback and discussions related to general assessment considerations.

6.2.1 Participants Overview

The recruitment process for focus group participants involved direct messaging on LinkedIn. In total, 73 professionals were invited to participate, 16 accepted to participate, 12 refused, and another 45 did not respond to the contact. However, among those who agreed to participate, only 13 confirmed and scheduled a time to participate. During the process, 2 participants cancelled due to personal and agenda issues, and another two did not attend the scheduled focus group sessions, resulting in nine participants.

Table 24 summarizes the participants' profiles. The 'ID' column designates each participant's identification code and their corresponding focus group. Significant variations are noticeable in participants' total IT experience, but specific experience in the DSP context showed less variation (between 1 and 5 years). Regarding the distribution of experiences across the focal groups, FG3 has an average IT experience of 18 years, the highest. In contrast, FG2 has the lowest average of 5.34. Another interesting observation is that 5 out of 9 participants have a master's degree, indicating the high level of study of the professionals involved in the field. However, a bias may be associated with this profile due to their academic background, as they

may have more empathy and willingness to collaborate in research.

Table 24 – Focus Group Participants Profile

ID	Gender	Working Country	Occupation	IT Exp. (years)	DSP Exp. (years)	Educational Level
FG1-01	male	Brazil	Ph.D. Student	4	1	Master
FG1-02	male	Brazil	IT Consultant	25	3	Master
FG1-03	male	Brazil	Software Development Specialist	16	3	Bachelor
FG2-01	male	Brazil	Software Engineer	6	2	Master
FG2-02	male	Brazil	Staff Data Engineer	5	4	Bachelor
FG2-03	male	Brazil	Data Engineer	6	3	Bachelor
FG3-01	male	Brazil	Director of Data Analytics	18	5	Bachelor
FG3-02	male	Brazil	Cloud Data Engineer	17	3	Master
FG3-03	female	Brazil	Data Engineer	19	3	Master

The survey received 22 responses through an outreach campaign using LinkedIn messages, discussion groups and dedicated DSP tool user email lists. The geographic distribution of interviewees, represented in Figure 17, covers several countries, with a concentration of 5 participants in the USA. Figure 18 shows the formal education level of the participants, highlighting that, as in the focus group, the participants' educational level is also high, with 17 out of 22 having a doctorate or master's degree. Finally, Figure 19 presents side-by-side the experience with IT and the specific experience in the context of DSP. It is noticeable that most respondents (17 out of 22) possess more than ten years of experience in IT. Regarding the DSP domain, the majority (17 out of 22) have between 2 and 10 years of experience.

6.2.2 Focus Groups Conduction

The focus group sessions were carried out on the 23rd, 29th and 31st of August 2023, using video conferencing alongside the Miro Online Whiteboard tool, which facilitated discussions with visual aids, as shown in Figure 20. The first section (FG1), which served as a pilot, lasted 73 minutes, while FG2 and FG3 lasted 78 and 101 minutes, respectively.

FG1's (pilot) assembly included a doctoral student researcher engaged in DSP-related study, an IT consultant with broad experience including DSP, and a practitioner working with DSP in the telecommunications sector. The section proceeded normally, validating the method

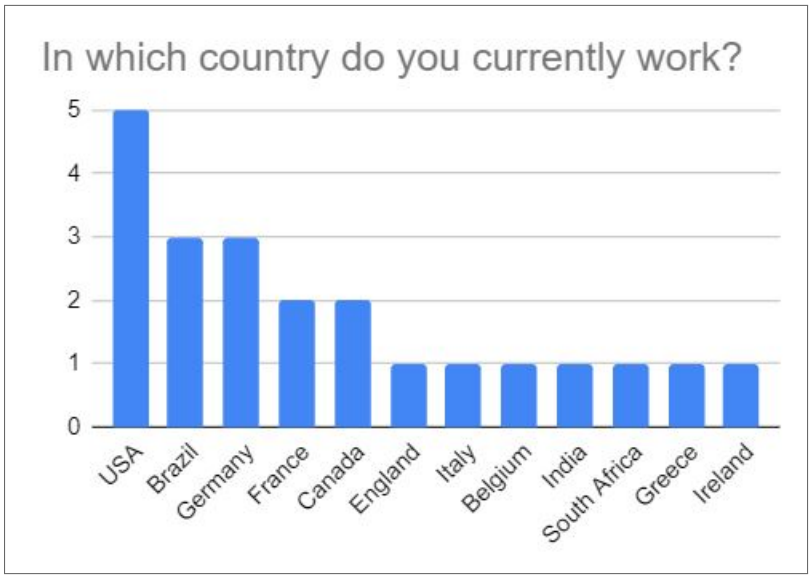


Figure 17 – Survey Participants Working Country.

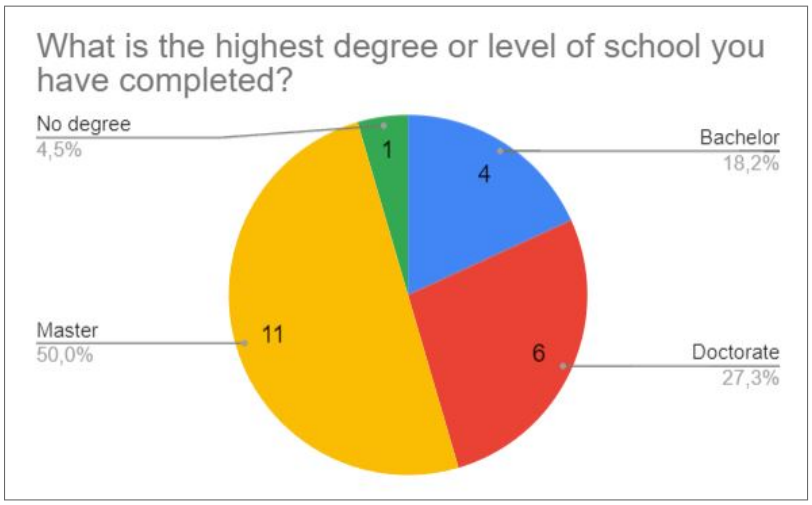


Figure 18 – Survey Participants Education Level.

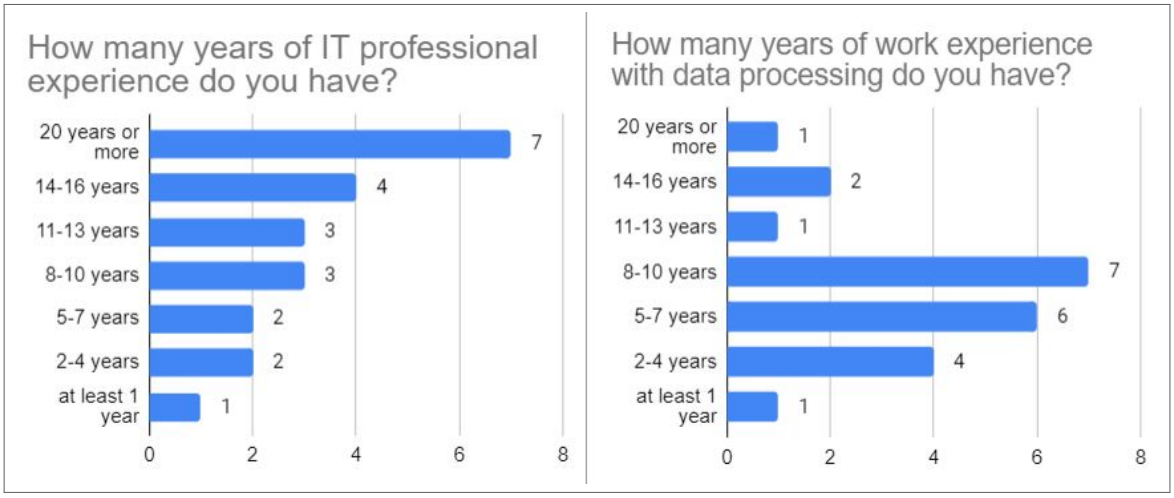


Figure 19 – Survey Participants Experience.

Figure 20 – Focus Group Interaction in Miro Online Whiteboard.

application, improving the script and training the moderator. A key improvement was reducing the review time per guideline item, allowing for more time for discussions. As there were no unexpected events that would invalidate the data collection and the pilot's results were considered as relevant as those from subsequent groups, we decided to include FG1 data in the final analysis.

FG2 and FG3 occurred as planned, except for a participant in each group who did not attend. Although this reduced the variety of opinions, on the other hand, it allowed more time for in-depth feedback. Particularly in FG3, participants were notably more engaged by providing detailed feedback, resulting in the session's extension. We emphasize that, throughout the discussion, we alerted participants about the time and consulted them about extending the section.

6.2.3 Evaluation of Guideline #G1

The evaluation of Guide #G1, conducted through focus groups and surveys, underscores the importance of collecting testing-relevant information at the project's outset. Several explicit comments highlighted this view. Moreover, on a scale of 1 to 5 regarding Guide #G1's

relevance, the results yielded an average score of 4.18. The key findings are summarized below:

(i) **DSP workflow modelling practices:** Participants reported that some form of data workflow modelling is vital for the testing team to understand the application and plan tests. Participants reported that informal diagramming for internal use is commonly practised in a development context. Additionally, BPMN (Business Process Model and Notation) have been identified by participant FG2-03 to be used for describing DSP workflows, especially during discussions with business specialists: *"In the context of industrial IoT, we used BPMN to model the DSP architecture. It was fine because the processes were complex and completely asynchronous; as we worked with Apache Kafka, which is completely asynchronous, we modelled with BPMN"*. Internet searches further validated this information, revealing several informal sources (blog posts) and academic studies reflecting similar practices (STEINDL; KASTNER, 2021). Moreover, the Imixs-Workflow tool (GMBH, 2023), a BPMN 2.0-based workflow engine, integrates with Apache Kafka and facilitates this approach's applicability in DSP-based workflow modelling. Therefore, item #G1-D in the guidelines was updated to include BPMN notation as an alternative modelling tool. This addition aims to promote improved communication between technical and business teams.

(ii) **UML usage:** While the utility of UML is acknowledged, its application within industrial contexts has become borderline. Participant FG3-02 briefly expressed this thought: *"UML diagrams have fallen out of fashion, and they are rarely used (except sequence diagrams)"*. This observation aligns with the findings of a qualitative study with professionals (JÚNIOR; FARIAS; SILVA, 2021), where 74% of respondents reported not using UML in practice. The study also shows a trend towards a preference for informal notation. However, the study reveals that practitioners recognize that the effectiveness of UML cannot be entirely dismissed. Despite the reported decline in its usage and professionals' tendency to favour informal notation, item #G1-D in the guidelines continues to advocate for using UML. This is due to the recognition of its benefits and the potential of UML diagrams to express characteristics pertinent to the context of DSP.

(iii) **Test specialist involvement in requirement gathering:** Participant FG3-03 emphasised the significance of involving quality assurance experts during the requirement-gathering phase: *"Bring Quality Assurance personnel to participate in the project plan-*

ning stages; they help in collecting test information and defining cards indicating which tests are appropriate for each case". The absence of such involvement often leads to a lack of crucial information for building tests, resulting in redundant work to re-gathering information for tests and additional costs, as exposed in the work by Pargaonkar (PARGAONKAR, 2023). Thus, the item #G1-E was included in response to this feedback. It advocates for the early inclusion of testing specialists in the information-gathering phase. This guidance aims to ensure that test-relevant information is collected from the beginning, minimising potential issues due to insufficient test information.

(iv) **Impact of agile methodologies on DSP testing:** The participant FG3-02 commented: *"Long-term planning is lost in agile, where code is prioritized over documentation"*. Although agile methodologies are known for speeding up development and meeting deadlines, they sometimes fail to fully collect requirements and crucial information for testing. Studies focusing on agile methodologies in large-scale and critical-mission projects reveal that agile processes face difficulties in meeting quality assurance practices (ALSAQAF; DANEVA; WIERINGA, 2019; RUSSO, 2021). For example, inadequate quality requirements testing specifications and integration testing can result in the discovery of defects at the end of the software life cycle. Large-scale and complex DSP applications require well-mapped requirements for testing. Recognizing this gap, item #G1-F was introduced in the guidelines. This addition underscores the necessity of striking a balance between the speed of agile development and the need for more careful planning, particularly in projects characterized by high complexity and significant financial risks. This balance is crucial as meticulous attention to test steps can prevent serious issues, ensuring that the agility of the development process does not compromise the depth and quality of testing in DSP projects.

(v) **SBE and ATDD:** During the evaluation, Specification by Example (SBE) and Acceptance Test-Driven Development (ATDD) were identified as practical methodologies for supporting the information collection testing phase. SBE (Specification by Example) aids in test development by enriching the understanding of the application's context. This is achieved through the development of sample data covering critical use cases, as outlined in Adzic's work (ADZIC, 2011). SBE recommendation was incorporated into item #G1-A of the guidelines. Similarly, ATDD was recognized for its inclusive approach that involves stakeholders, including developers, testers, and business representatives, in the process

of defining acceptance criteria (GÄRTNER, 2012). This methodology promotes a collaborative environment and ensures that crucial information for testing is gathered early on. Accordingly, a recommendation for adopting ATDD was included in item #G1-B of the guidelines.

(vi) **Scenario-based testing for fault tolerance:** Participant FG3-03 recommended the specification of fault tolerance issues through the scenario-based testing method: *"Scenario-based testing may be useful in describing scenarios of adverse conditions and associated tolerance tests"*. The study by Gong et al. (GONG et al., 2021) reported using this method to specify fault tolerance scenarios regarding reliability in a smart grid context. Business-targeted planning of fault tolerance strategies is essential, as different businesses have varying tolerances for DSP issues. For example, tolerance to delayed data, data loss, and downtime differs for the financial and e-commerce contexts. In response to this feedback, scenario-based testing was recommended and included in item #G1-C of the guidelines.

(vii) **Include essential information for test planning:** Throughout the evaluation, we collected relevant information for tests to be gathered, which was compiled in a list in Section 6.1.1.1 associated with item #G1-B.

- Identifying message formats and schema structure.
- Mapping data stream producers and consumers.
- Identifying the use of third-party services (APIs, data sources): Verify SLAs (Service Level Agreements), taking into account service availability, amount of TPS (transactions per second), response time, and downtime policies.
- Mapping the stream data that leads to automatic business decisions/actions and manual decisions via dashboards.
- Checking whether the business model is compatible with data post-processing in downtime scenarios.
- Mapping response times, throughput rates, and time-out durations.
- Data validity: Duration of data retention in cache.
- Potential data loss considerations.
- Identifying transaction semantics: Atomicity, Durability, Ordering Guarantees, and Exactly-Once Processing.

- Identifying data availability for test purposes (historical, synthetic, or custom example data).
- Confidentiality and privacy features: Consider specific regulations and protections for each data type. What are the PII (Personally Identifiable Information) in the data set?

The evaluation of Guide #G1 underscored the role of early information collection for effective DSP testing. Essential feedback includes adopting BPMN for workflow modelling, emphasising test specialists' inclusion in requirement gathering, balancing agile methodologies with detailed planning, and advocating for SBE and ATDD methodologies. Additionally, scenario-based testing for fault tolerance was recommended. These comprehensive changes are summarised in Table 25.

Table 25 – Summary of promoted changes regarding #G1 evaluation

Feedback	**Promoted Changes**
(i) DSP Workflow Modelling Practices	Item #G1-D was updated to include BPMN notation as an alternative modelling tool aimed to promote improved communication between technical and business teams.
(ii) UML Usage	No changes, item #G1-D continues to recommend UML.
(iii) Test Specialist Involvement in Requirement Gathering	Item #G1-E was introduced, recommending the participation of a testing specialist in the information-gathering phase.
(iv) Impact of Agile Methodologies on DSP Testing	Item #G1-F was added to emphasize that, in projects with high complexity and financial risks involved, there is a need for a balance between agile development and more careful planning.
(v) SBE and ATDD	SBE and ATDD methodologies were incorporated into items #G1-A and #G1-B, respectively.
(vi) Scenario-Based Testing for Fault Tolerance	Scenario-based testing was advised in item #G1-C.
(vii) Include Essential Information for Test Planning	Item #G1-B now features an expandable chart, including a suggested list of testing information to collect in the early phase of the project.

6.2.4 Evaluation of Guideline #G2

The evaluation of Guide #G2 reveals a consensus on the positive effect of aligning testing objectives by considering nuances of business requirements and operational realities. The average score for this item on the survey's relevance scale was 3.95. Participants perceived

the proposed questionnaire as a valuable tool to guide early-stage discussions in establishing testing objectives, essential for preempting future problems. This sentiment is reflected by FG3-01 comment: *"Sometimes this conversation does not happen, leading to unpleasant surprises when it is too late, so this questionnaire helps map this information to guide which tests priority"*.

(i) **Challenges in detailing project planning in advance:** A survey comment highlighted challenges in predicting and detailing extensive information at the beginning of a project, especially in innovative projects with evolving requirements: *"It is better to get the important things right - the ones that are expensive to change later. But trying to think about every little detail seems like a waste, especially if the business context changes requirements frequently"*. In response to this feedback, a text was added to the beginning of the guidelines advising on the flexible use of the guidelines, guiding readers on recommendations to adapt the guidelines to suit the evolving nature of their projects, thus ensuring that the guidelines remain practical in the face of evolving business contexts. Although this feedback specifically regards #G2, its relevance extends to #G1 as well.

(ii) **Stakeholder engagement in testing objectives:** The evaluation feedback corroborates with item #G2-C, underscoring the benefits of engaging a diverse group of stakeholders, including business analysts, developers, DevOps, testers, clients, and users, in the process of establishing testing objectives. This inclusive strategy ensures alignment between the testing approach and both technical and business quality requirements (GREGORY; CRISPIN, 2014). Participant FG1-03 illustrates this context: *"When the quality team understands the core concerns of the business and the commercial side knows something about the nuances of data processing, communication flows better, leading to more accurate testing objectives"*. Consequently, item #G2-C of the guidelines was revised to include a wider range of stakeholder categories, thereby broadening the diversity of perspectives and expertise contributing to the testing process.

(iii) **Relevance of questionnaire items and need for a concise version:** Regarding the questionnaire items, feedback indicated no solid preferences for removing specific items. They found all questions to be pertinent for discussions, though they noted that the applicability of these questions could vary depending on the project size. Recognizing

the need for versatility, a suggestion was made for a more concise version of the guidelines tailored to smaller projects. In response to this feedback, the cheat sheet version (Appendix J) would meet this need for conciseness.

(iv) **Trade-off between quality categories:** A participant suggested appending an observation that a trade-off often exists between functional suitability, performance efficiency, reliability, and maintainability, and focusing on one aspect may compromise another. The content of item #G2-A was revised to encompass considerations about the trade-offs among different quality categories.

The evaluation validates the importance of aligning test objectives with business requirements and operational realities. Comments included the need for flexible project planning, especially for innovative projects with evolving requirements. The importance of involving diverse stakeholders in testing discussions was also emphasized, ensuring communication between technical and business people. These insights led to updates to item #G2, all summarized in Table 26.

Table 26 – Summary of promoted changes regarding #G2 evaluation

Feedback	**Promoted Changes**
(i) Challenges in Detailing Project Planning in Advance	We included an excerpt at the beginning of the guidelines on how to use the document, which provides advice on the flexible use of the guidelines.
(ii) Stakeholder Engagement in Testing Objectives	We expanded item #G2-C to detail additional stakeholder categories that could contribute, enhancing the breadth of perspectives and expertise.
(iii) Relevance of Questionnaire Items and Need for a Concise Version	In response to the recommendation for a more concise version, we believe the cheat sheet version (Appendix J) meets this requirement.
(iv) Trade-off Between Quality Categories	Item #G2-A was changed to include an observation regarding trade-offs between quality categories.

6.2.5 Evaluation of Guideline #G3

In both the focus group and the survey, participants agreed that the qualifications of individuals involved in the testing processes are a relevant issue and should be considered. However, on the survey's relevance scale of 1 to 5, the average relevance of #G3 was 3.27, the lowest level among the seven guidelines. There were some reservations regarding the team

setup and the separate planning of the testing workload. In addition, the feedback brought new topics to be included and reflections on related issues.The main topics discussed are summarized below:

(i) **Importance of DevOps skills:** Participant FG1-01 discussed the significance of DevOps skills in automating the testing process: *"Today, development has become intrinsic to DevOps activity because of development automation, mainly regarding the testing automation in CI and CD environments"*. DevOps skills are especially desirable for testing DSP applications, as more complex tests require building many containers and configuring several services. This massive work needs to be automated to optimize the work, with DevOps benefits for this purpose being consolidated (PATEL; TYAGI, 2022). We emphasized the importance of DevOps-related skills in item #G3-A.

(ii) **Understanding of stream platforms and architectures:** A deep understanding of the adopted stream platforms, the main concepts behind these tools, and the data life cycle within the platform's architecture were highlighted as essential for practical testing. Participant FG2-01 cited that misconfigurations in platform features often cause significant testing issues: *"When testing, there are many infrastructure configurations, such as those of the broker and stream processor, which drastically change the results of the tests. The tester must know these features"*. We detailed item #G3-A, considering the need for specific knowledge about tool configurations and architecture in the DSP context.

(iii) **Data-specialized quality assurance professionals:** Two focus group participants underscored the importance of data-specialized QA professionals who comprehend data flows and architecture, the following focus participant FG3-03 comment represents this discussion: *"It is desirable to have QA professionals specialized in data. They must understand data processing workflow, data engineering fundamentals, automation architecture, and methods for testing data volume and variety. The emergence of Data QA as a distinct specialization is already occurring, and I know some of these professionals"*. Taking that into account, the changes to item #G3-B also included observations regarding the importance of data-specialized QA professionals who comprehend data flows and architecture.

(iv) **Diverse specialists should get involved:** Participants pointed out that the complexity of DSP testing, particularly for high-volume and real-time data processing, often requires the involvement of various specialists beyond conventional QA roles. The importance of the infrastructure engineering team in testing the performance of large-scale DSP applications was specifically noted by participant FG3-01: *"We were only able to test performance at the systems level, with the entire architecture and implementation team participating to verify that everything that was done is working correctly"*. We updated item #G3-A to include more professionals' specialities to collaborate in tests.

(v) **Understanding the business domain:** Feedback highlighted the critical importance of business domain knowledge for testing. A focus group participant emphasized this, stating: *"Personnel involved in testing must have experience in the business's specific context. For instance, in a financial setting, numerous regulations and security concerns significantly influence the testing process; e-commerce is another story"*. Item #G3-D was included to address recommendations regarding the importance of knowing the business context for testing activities.

(vi) **Prioritizing hands-on experience over formal training:** There is a general agreement that formal training in DSP testing is uncommon, with most learning occurring on the job. The real-world expectation is for the current testers and developers to adapt and acquire the required skills for testing demands. Respondents emphasized the necessity of hands-on experience with streaming data systems, highlighting a lack of structured training in this field. The feedback consistently pointed to a continuous learning approach as critical for successfully acquiring testing skills for the DSP context. These two comments from the survey illustrates this issue: *"Very rarely companies have time to provide proper training before a test. People usually learn on the go"* and *''We are all learning while doing. There is no separate training before getting to work building tests"*. Acknowledging that formal training in DSP testing is uncommon, we updated item #G3-B to reflect the importance of hands-on experience. This change corroborates the notion that practical experience is a primary avenue for learning in this context, reinforcing the need for a resource like these guidelines to consolidate industry knowledge on DSP testing.

In summary, the evaluation provided feedback regarding the relevance of DevOps skills, a

deep understanding of DSP platforms and architecture, and Data-specialized QA Professionals. Furthermore, the importance of business domain knowledge and the prioritization of practical experience over formal training was reported. These insights guided revisions to item #G3 to cover a wider range of skills and knowledge for testing in the DSP context. The changes are summarized in Table 27.

Table 27 – Summary of promoted changes regarding #G3 evaluation

Feedback	**Promoted Changes**
(i) Importance of DevOps Skills	We emphasized the importance of DevOps-related skills in item #G3-A.
(ii) Understanding of Stream Platforms and Architecture	We detailed item #G3-A, considering the need for specific knowledge about tool configurations and architecture in the DSP context.
(iii) Data-specialized Quality Assurance Professionals	Item #G3-B was updated to incorporate observations regarding the importance of data-specialized QA professionals.
(iv) Diverse Specialists Should Get Involved	The new version of item #G3-A covers the diversity of professionals' specialities involved in tests.
(v) Understanding the Business Domain	We included item #G3-D to address the suggestion regarding the importance of knowledge concerning the business domain.
(vi) Prioritizing Hands-on Experience Over Formal Training	Item #G3-B was to reflect the importance of hands-on experience.

6.2.6 Evaluation of Guideline #G4

In the evaluation of Guideline #G4, participants' opinions corroborate the importance of time management in testing. This guideline received an average relevance score of 3.64 on a 1 to 5 scale in the survey. Although some feedback indicated the recommendations align more with general testing principles than DSP-specific practices, other insights highlighted unique time planning challenges in the DSP contexts. These include the time required for system test configurations, executing performance and load tests, coordinating with external entities, and planning complex use case tests. The following is a summary of key feedback and related discussions:

(i) **Understanding time dimensions in testing:** Participant FG1-02 emphasized the need to make clear the distinction between the development time required for complex tests and the time spent executing them: *"It is worth highlighting that there are two dimensions of time there, the time to develop a complex test case and the time to execute tests;*

for example, load and performance tests are particularly time-consuming". We confirm that understanding the dimensions of testing time would be helpful. This distinction is especially significant in scenarios involving large data loads, a common DSP testing circumstance where test execution times may vary greatly. In response to this insight, we updated item #G4-A to underscore the importance of considering both planning and execution aspects in time management for testing.

(ii) **The time recommendations are general:** A survey response suggested that, while this guideline is fundamentally important, such recommendations regarding time allocation are general, valid for different contexts, and are a project management concern: *"This guideline is significant, but it is generic and has more to do with project management than DSP testing."* Despite this, the necessity of highlighting the importance of time planning in DSP testing will be maintained in the guidelines, as complex tests in this field often require considerable development time, and specific test types, such as load and performance tests, are known for their prolonged execution periods. This aspect of time management in DSP testing is reflected in other feedback on #G4, justifying why we choose to maintain our guide on time planning.

(iii) **Automation and time wfficiency:** The role of automation in enhancing testing efficiency emerged during both the focus group discussions and the survey feedback. Participants emphasised the role of automation in reducing time spent on various testing activities. Participant FG1-03 noted: *"Automating common tasks helps a lot to speed up the testing process; having prepared containers and scripts to run the tests, that kind of thing"*. Automation could minimise the time invested in repetitive and labour-intensive tasks, such as test case generation and infrastructure setup. However, it is essential to assess the cost-benefit of the automation effort (GAROUSI; ELBERZHAGER, 2017). This feedback aligns with findings from other studies involving practitioners on test automation (GAROUSI; MÄNTYLÄ, 2016). Nevertheless, this feedback did not drive changes to the guidelines; it only reinforces the importance of items #G4-E and #G4-F in our guidelines.

(iv) **Underestimating system level testing time in agile development processes:** In agile development processes, the integration of testing into the overall development workflow is standard practice, with testing being a continuous part of the development

cycle rather than a separate phase. However, a common issue identified in agile environments is the underestimation of system-level testing time, particularly for more complex tests, as highlighted by one survey respondent: *"Allocating time for system and performance testing is crucial, yet it is often insufficiently considered in projects and agile sprints"*. To address this challenge, item #G4-G was introduced, focusing on time planning within agile environments. This addition recognizes that basic unit and simple integration tests typically align with sprint schedules. System-level tests, especially those involving large-scale volumes of data such as performance tests, demand more accurate time allocation. The challenges associated with adapting an agile process to accommodate time-consuming tests, particularly in large-scale projects, have been reported and in academic literature (KASAULI et al., 2021; DINGSØYR et al., 2019).

(v) **Schedule for testing with external dependencies:** Conducting tests that involve external dependencies must be carefully planned and coordinated with the teams managing these services, as underscored by a participant FG3-01, who emphasised the need for advanced scheduling: *"The time dedicated to testing third-party applications needs to be planned and often scheduled and agreed with external entities"*. This need becomes evident in the context of testing with external data sources and third-party services. In fact, in the formal literature, the work of Sablis et al. (SABLIS; SMITE; MOE, 2021) addresses the issue of coordinating external teams in testing activities. Regarding the importance of this coordination, the item #G4-G in our guidelines was updated to address scheduling tests, which requires collaboration with multiple teams.

In summary, the discussions encompassed several key areas: considering time dimensions, exploring the role of automation, delving into the nuances of agile methodologies, and addressing the challenges of external dependencies in testing. The feedback received from these discussions has led to changes in the guidelines, which are summarized in Table 28.

6.2.7 Evaluation of Guideline #G5

In evaluating this guideline, participants underscored the significance of financial considerations, particularly in large-scale projects with extensive infrastructures and substantial data volumes. The guideline received an average relevance score of 3.33 on a 1 to 5 scale. Feed-

Table 28 – Summary of promoted changes regarding #G4 evaluation

Feedback	**Promoted Changes**
(i) Understanding Time Dimensions in Testing	We added a note to item #G4-A to consider both planning and executing aspects of time management.
(ii) Time Recommendations are General	No changes, but the many minor #G4 improvements reinforce DSP-specific issues of time planning.
(iii) Automation and Time Efficiency	No changes, the feedback affirms the significance of items #G4-E and #G4-F.
(iv) Underestimating System Level Testing Time in Agile Development Processes	Item #G4-G was introduced to address issues on time planning in agile environments.
(v) Schedule for Testing with External Dependencies	The importance of scheduling tests with multiple teams and third-party participants is now covered in item #G4-G.

back predominantly focused on the costs related to cloud environments, strategies for reducing expenses, and the financial implications of load testing.

(i) **The significance of early financial planning and cost management:** Participants agree that proactive financial planning is crucial in the testing process to prevent budget overruns. Participant FG1-03 shared an experience where the unregulated cloud service usage for testing led to budgetary issues. Consequently, the company implemented strict regulations for cloud machine allocation: *"Our testing resource allocation management was inefficient and resulted in budget overruns. As a result, we are now facing a limitation on the number of available machines for testing until the fiscal year concludes"*. The feedback underscores the importance of financial planning in DSP testing; this affirms the importance of #G5.

(ii) **Cloud costs:** The evaluation highlighted the significant cost implications of cloud-based testing environments, particularly when handling high data volumes. Participant FG3-01 pointed out: *"One of the primary expenses is linked to cloud services, which becomes particularly significant with large data volumes"*. In response to recurring feedback concerning the high costs associated with cloud infrastructures, we introduced item #G5-D. This item emphasizes the need for policies on machine allocation for testing and timing the execution of financially intensive tests.

(iii) **Costs considerations in large projects:** Participants observed that building reliable test environments for large-scale projects is costly. Participant FG3-01 remarks encapsulated this: *"A large test setup which resembles the production environment can get very*

expensive". In light of this, we revised item #G5-B to emphasise that testing expenses are particularly significant in large-scale projects.

(iv) **Employing infrastructure mocking for costs reduction:** In the discussions, infrastructure mocking was recognized as an effective cost-reduction strategy, particularly for correctness testing and local testing scenarios, as outlined in item #G5-C.

(v) **Strategies to reduce system test costs:** Participant FG3-03 emphasized the financial challenges of frequently running system tests with large data volumes. He suggested: *"It is financially unfeasible to run large-scale system tests after every update"*. Policy establishment to dictate when to conduct system tests was recommended. In cases of resource limitations or critical service needs, testing directly in the production environment was proposed as an alternative.

(vi) **Financial impact of load testing:** Participants specifically identified load testing as a financially demanding activity requiring an infrastructure similar to the production environment. This requirement is particularly crucial in projects where adherence to Service Level Agreements (SLAs) is imperative. Participant FG2-01 noted: *"Load testing demands a lot of financial resources for cloud infrastructure and is thus not conducted frequently. It is typically performed prior to deployment to ensure compliance with SLAs"*. To address load testing costs in projects with rigorous SLA requirements, we added item #G5-E.

(vii) **Financial risks of ignoring architecture validation:** Participant FG3-01 shared that inadequate architectural solutions led to extensive project restructuring: *"During the load testing phase, we discovered that the application could not handle the necessary volume, the architecture had to be changed, and a lot of work had to be redone"*. This unforeseen rework led to substantial extra costs not accounted for in the initial project budget. The participant suggested early testing for architectural validation as a preventive measure against such financial setbacks. Accordingly, we consider validating architectural decisions essential in the DSP context, especially in large and complex projects where adjusting the architecture at advanced phases can incur considerable costs. In response, we introduced item #G5-F, which focuses on the importance of early architecture validation.

Discussions included the need for early financial planning to avoid overspending, the high costs associated with cloud-based testing, and the expensive nature of test setups for large projects. Strategies such as infrastructure simulation to reduce costs and the financial implications of load testing, especially in projects with strict SLAs, were also discussed. These insights led to changes in #G5, detailed in Table 29.

Table 29 – Summary of promoted changes regarding #G5 evaluation

Feedback	**Promoted Changes**
(i) The Significance of Early Financial Planning and Cost Management	No changes; the evaluation affirms the importance of #G5.
(ii) Cloud Costs	We introduced item #G5-D regarding the need for policies on machine allocation for testing and timing the execution of financially intensive tests.
(iii) Cost Considerations in Large Projects	We updated item #G5-B to highlight that testing costs are particularly significant in large-scale projects.
(iv) Employing Infrastructure Mocking for Cost Reduction	The feedback affirms the relevance of item #G5-C.
(v) Strategies to Reduce System Test Costs	Item #G5-D also includes recommendations for establishing protocols for conducting expensive tests, such as system-level tests.
(vi) Financial Impact of Load Testing	We added item #G5-E regarding load testing costs in projects with strict SLA requirements.
(vii) Financial Risks of Ignoring Architecture Validation	Item #G5-F was included to focus on the importance of early architecture validation.

6.2.8 Evaluation of Guideline #G6

The evaluation of Guideline #G6 revealed a strong consensus among participants regarding the importance of a comprehensive test data strategy. This guideline achieved an average relevance score of 4.22 from 1 to 5. Feedback from practitioners and survey respondents emphasized the necessity for diverse, high-quality test data and underscored the importance of careful data privacy management and schema evolution.

(i) **Schema management and evolution:** Participants recognized the significance of item #G6-E, which focuses on data schema management. This perspective is illustrated by FG3-01 feedback: *"Avro is mandatory. I have seen cases in projects lacking schema management, leading to schema changes without proper communication and resulting in a complete pipeline breakdown"*. Despite this, participants pointed out that Avro

is unsuitable for managing schema evolution. Apache Delta was recommended for its schema management capabilities, facilitating data compatibility checks and maintaining schema versioning. This allows navigation across various schema versions, as expressed by participant FG3-02: *"Avro does not handle evolution; Delta Lake is the advised option, offering comprehensive maintenance and 'time travel' on several schema versions"*. We reviewed item #G6-E, refining guidance on Avro's use with a focus on preventing contract breaking. We also incorporated a recommendation to employ the Apache Delta tool for schema evolution management.

(ii) **Data validity and lifecycle management:** The evaluation highlighted issues related to data validity. Over time, the utility of data for testing decreases due to changes in concepts, data structures, and the necessity for data that tests new features. Participant FG3-03 advised establishing policies for data lifecycle management: *"It is essential to think about the lifecycle of our test data. We must consider how long to retain this data, recognizing that storage costs vary based on data accessibility. Frequent data access leads to increased costs"*. Item (3), addressing Currentness in the Data Quality Characteristics board, deals with data validity issues. We updated this item to include a data retirement policy and concept drift considerations.

(iii) **Data validation challenges:** Even with effective schema management, issues like incompatible text encodings, GPS data coordinate patterns, variations between metric and imperial systems, and conflicts between signed and unsigned data types can still arise. The participant FG1-03 report underscores this: *"In a recent project, we encountered a data format issue. A binary number received and interpreted as a signed integer was actually an unsigned integer from the source. Two days of reverse engineering were needed to resolve this issue"*. Item (4), focusing on Compliance in the Data Quality Characteristics board, concerns adherence to regulations and data structure integrity. We enhanced this item's text with examples for better clarity.

(iv) **Utilizing great expectations for data quality:** Feedback highlighted the efficacy of frameworks like Great Expectations in evaluating the quality of synthetically generated test data. Participants in the focus group also emphasized its applicability in monitoring production stream data quality by operating it within a parallel pipeline. This approach generates reports and issues alerts regarding deviations in data quality. The Great Ex-

pectations tool, previously identified in the GLR, is now explicitly recommended in item #G6-A.

(v) **Diversity in test data:** Consistent feedback from focus group discussions and survey responses underscored the importance of diversifying data sources and generation methods. This approach, as outlined in item #G6-B, ensures data diversity and helps minimize biases.

(vi) **Limitations of historical data:** Participants agree with the concerns raised in item #G6-C about the limitations of relying exclusively on historical data. They confirm that historical data might fail to encompass newly developed features or previously unmanifested defects and could quickly become outdated. This perspective emphasizes the necessity for a balanced approach in test data strategy that supplements historical data with recent and varied datasets.

(vii) **Data privacy and regulatory compliance:** In discussions related to item #G6-F, participants emphasized the practical challenges of adhering to privacy regulations while managing test data. Participant FG3-02 outlined anonymization techniques and commented about the significant effort involved: *"Various techniques such as Redaction, Replacement, Masking, Crypto-based tokenization, Bucketing, Date shifting and Time extraction are used to maintain data privacy. Typically, these processes are labour-intensive and time-consuming, as scripts and procedures must be customized for each specific case"*. In response to feedback affirming the relevance of item #G6-F, we included specific scripting techniques that assist in preserving data privacy within testing environments.

Highlights from the evaluation included the importance of managing data schemas effectively, ensuring data validity and lifecycle, addressing data validation challenges, and using frameworks like Great Expectations for quality assessment. The need for multiple test data sources and the limitations of historical data were confirmed. Maintaining data privacy and adhering to regulatory compliance have also emerged as crucial concerns. These insights led to significant updates to the #G6, concisely summarized in Table 30.

Table 30 – Summary of promoted changes regarding #G6 evaluation

Feedback	**Promoted Changes**
(i) Schema Management and Evolution	Item #G6-E was changed to guide Avro's use, focusing on preventing contract breaking, and the Apache Delta tool for schema evolution management.
(ii) Data Validity and Lifecycle Management	We updated Item (3) in the Data Quality Characteristics board to include a data retirement policy and concept drift considerations.
(iii) Data Validation Challenges	Item (4) in the Data Quality Characteristics board was incremented with examples for better clarity.
(iv) Utilizing Great Expectations for Data Quality	The Great Expectations tool is now explicitly recommended in item #G6-A.
(v) Diversity in Test Data	No changes. The evaluation confirms the importance of item #G6-B.
(vi) Limitations of Historical Data	No changes. The feedback validates the significance of item #G6-C.
(vii) Data Privacy and Regulatory Compliance	Item #G6-F was updated to suggest scripting techniques that assist in preserving data privacy.

6.2.9 Evaluation of Guideline #G7

The overall feedback from focus groups and survey responses reinforced the significance of Guideline #G7, addressing practical experiences related to the guideline's items and recommending tools and strategies to handle particular DSP issues. The guideline received a relevance score of 4.04 on a scale of 1 to 5. Below is a summary of the most pertinent feedback.

(i) **Reproducing temporal characteristics in tests:** Participant FG2-01 confirmed that accurately replicating time-related conditions in tests is challenging. A focus group participant stated: *"It is challenging to replicate production environment conditions in testing, especially concerning network characteristics and latency issues"*. Participant FG2-03 emphasized the effective features of Apache Flink for clock manipulation, stating: *"With Apache Flink's clock simulation capabilities, it is possible to accelerate, pause, and manipulate latency, as well as advance or delay messages"*. The feedback affirms the significance of item #G7-A concerning time-related issues. In response, we added a note item in #G7-A regarding clock control features present in stream processing platforms.

(ii) **Stateful operation and non-deterministic behavior:** Feedback corroborated the statement of item #G7-B regarding the stateful operations as a recurring cause of the

non-deterministic behaviour, as participant FG3-01 remarked, *"When creating stateful processes, it is impossible to guarantee the same request will be executed by the same component"*. Participant FG3-01 also reported avoiding stateful processes to mitigate non-determinism, highlighting the complexity statefulness adds to ensuring result consistency: *"We avoid non-determinism by minimizing stateful processes in the pipeline"*.

(iii) **Divergent opinions on non-determinism role in the DSP context:** Differing views were expressed regarding non-determinism. Some survey respondents argued that non-determinism is not a major worry with modern stream processors designed to minimize it: *"Modern stream processors aim to avoid non-determinism wherever possible, achieving determinism across a wide range of use cases and operations"*. While tools do provide functionality to minimize non-determinism, the issue remains a significant consideration in complex applications context, as reported by other participants. Item (5) was included in the non-determinism testing strategies, containing supplementary information detailing how stream processing platforms help to minimise non-determinism.

(iv) **Test oracle and non-determinism:** Participant FG3-02 acknowledged the practical use of test oracle strategies from item (1) regarding non-determinism: *"The recommendation on using thresholds for the test oracle is indeed applied in practice. But this term is uncommon; here in my company, we call it statistical data control"*. Participant FG3-03 emphasized that this strategy plays a key role in data validation by using the tool Great Expectations, which includes a feature for setting data variation thresholds: *"With Great Expectations, you can set thresholds to validate the data"*. The feedback reinforces the validity of item (1) in the Non-determinism Testing Strategies board. We enhanced this item by including information on the Great Expectations tool's capability to set data variation thresholds.

(v) **Fault tolerance infrastructure testing:** Participants confirmed the importance of fault tolerance testing, particularly in complex applications requiring high reliability and availability. The Netflix tool Chaos Monkey was mentioned as adopted in such type of test. Participant FG3-01 shared his experience: *"I worked at a company with a team focused on fault tolerance testing. They had autonomy over the infrastructure and would deliberately turn off machines, deactivate services, increase latency, and observe the outcomes"*. The feedback confirms the importance of item #G7-C. Item (8) was

included in the Non-determinism Testing Strategies board as a list of tools used in fault tolerance testing.

(vi) **Horizontally scaling up a broker cluster:** A participant detailed a general strategy for managing sudden increases in data volume: *"Configure the broker for horizontal autoscaling to accommodate the additional load, but check if consumers support it"*. However, it is important to note that if the consumers processing the stream cannot match the escalated demand, messages will accumulate in the broker's cache, leading to a loss of real-time processing. We updated item (2) on the Fault Tolerance Recommendations board to reflect insights on the importance of preparing the broker for scaling in response to increased data volumes. Additionally, we included observations about aligning consumer capabilities with intensified demand to ensure that real-time processing is not affected.

(vii) **Broker replicas:** Participants emphasized the robustness and reliability of data stream brokers in maintaining service availability. These tools can be configured with multiple replica instances that substitute the central broker, reducing the likelihood of service interruption. However, increased latency for message confirmation and the associated costs of maintaining additional infrastructure and network traffic are considerations. A survey participant noted: *"Downtime is a critical concern, but many tools like Spark and Flink handle it with a replica mechanism, allowing us to trust in their ability to manage"*. This feedback underscores the relevance of item (1) on the Fault Tolerance Recommendations board. We expanded this item to include further insights on the impacts of latency and the financial implications of implementing replicas.

(viii) **Data loss strategy:** A focus group participant pointed out that synchronizing all incoming messages in a data lake is a strategy to prevent data loss. However, other participants cautioned that this approach might not be suitable for high-volume data scenarios. Therefore, we introduced the item (9) specifically to address data loss tolerance strategies.

Key evaluation insights included the difficulty in replicating time-related conditions in tests and the non-determinism associated with stateful operations. Features of Great Expectations support data variation thresholds. Strategies for horizontally scaling up a broker cluster and maintaining service availability through broker replicas were discussed. Additionally, data loss

prevention strategies were suggested, such as synchronizing incoming messages in a data lake. These changes are summarized in Table 31.

Table 31 – Summary of promoted changes regarding #G7 evaluation

Feedback	**Promoted Changes**
(i) Reproducing Temporal Characteristics in Tests	A note was included in item #G7-A regarding clock control features present in stream processing platforms.
(ii) Stateful Operation and Non-Deterministic Behavior	No changes. The feedback validates the role of stateful operations as a cause of non-determinism in the DSP context, as highlighted in item #G7-B.
(iii) Divergent Opinions on Non-Determinism Role in the DSP context	Item (5) was included in the non-determinism testing strategies, containing supplementary information detailing how stream processing platforms help to minimise non-determinism.
(iv) Test Oracle and Non-Determinism	Item (1) in the Non-determinism Testing Strategies board was complemented to include information on the Great Expectations capability to set data variation thresholds.
(v) Fault Tolerance Infrastructure Testing	Item (8) was included in the Non-determinism Testing Strategies board as a list of tools used in fault tolerance testing.
in the Non-determinism Testing Strategies (vi) Horizontally Scaling up a Broker Cluster	Item (2) of the Fault Tolerance Recommendations table has been enhanced to encompass insights on aligning broker and consumer capabilities with increased demands.
(vii) Broker Replicas	Item (1) on the Fault Tolerance Recommendations board was expanded to include further insights on the impacts of latency and the financial implications of implementing replicas.
(viii) Data Loss Strategy	Item (9) was included to address strategies for data loss tolerance.

6.2.10 Evaluation of General Feedbacks

After individually evaluating each guideline, we gathered general feedback. The following subsections highlight the key points discussed.

6.2.10.1 RQ2.1 Perceived Strengths

Feedback gathered from both focus groups and surveys enclosed various aspects that were well-received by participants. Key strengths noted include consolidating dispersed information into a single document containing relevant content that is present in everyday testing

routines. Participants specifically praised the recommendations on synthetic data and fault tolerance strategies. Additionally, the colour layout and design elements have been recognized for improving user engagement and readability. The following comments exemplify the positive reception of these guidelines:

- *"I consider this guideline important because all this information was dispersed, and summarizing it in a document already represents something very useful"*. [FG1-01]
- *"Definitely, it references a lot of content that frequently appears in testing scenarios"*.[FG2-03]
- *"The section on generating synthetic data was impressive. I look forward for more details and references"*.[FG2-01]
- *"The guide regarding fault tolerance is very relevant nowadays, especially in applications in mission-critical systems"*.[survey feedback]
- *"Colour-coding to separate the different topics makes perfect sense. Even the dashed and solid lines between topics contribute to that"*.[FG3-03]
- *"I do like these simple-to-read one/two pagers or info-graphics. They provide a quick overview, and if the reader is interested in more details, there is enough information to start a detailed analysis"*.[survey feedback]

6.2.10.2 RQ2.2 Weaknesses and Areas for Improvement

Despite the positive feedback, practitioners also identified some negative points and suggested improvements. There is a noticeable demand for practical examples, visual aids, and references to tools. Below are some comments that evidence these needs:

- *"Perhaps a detailed example demonstrating the guidelines' application would clarify things considerably"*.[FG1-02]
- *"Some items need examples, tool suggestions, and more detailed explanations. For instance, which algorithms and tools are recommended for synthetic data generation? And references?"*[FG3-02]

- *"A visual summary, like a mind map, guiding the guidelines would help quickly understand the information."*.[FG3-03]

6.2.10.3 RQ2.3 Perceived Applicability in Industrial Context

Regarding the guidelines' industrial applicability, practitioners generally found them helpful, but they noted the need for adaptations based on company culture, development process and project context. The document was considered an excellent starting point for newcomers and valuable in training, and many recommendations in the guidelines are already standard practice in industry testing. Here are some representative comments:

- *"I would use parts of it but would need to omit things irrelevant to our development team, like the resources planning parts"*.[FG2-02]
- *"I believe this document would be very helpful in guiding someone with little experience in the field. It could also be part of training"*.[FG1-02]
- *"Many of these recommendations, like synthetic data generation with property-based testing and data quality monitoring, are already practised in my company"*.[FG2-03]

6.2.10.4 Additional Feedback: Suggestions on Guidelines Format

There were varied considerations regarding the format of the guidelines. Some practitioners felt they appeared as rigid steps, comparable to a waterfall model. There were calls for a flexible format compatible with agile methodologies, adaptable to enhance usability and applicability. For example, information from items #G1 and #G2 might not be collectable at once but instead iteratively across sprints.

Regarding the document's format, participants reported that the PDF document is difficult to access and browsing it to look for information within it is not practical. A web format with navigable hyperlinks, interactive elements, and options to display or hide information was suggested. This format can be used to provide different levels of detail, with examples, related links, and in-depth explanations. Participants also wanted lighter or more detailed versions of the guidelines, taking inspiration from resources like the "The Twelve-Factor App" [1] that associates content with a list of principles. Some representative comments include:

[1] <https://12factor.net>

- *"The testing structure proposed here is well thought out. I work in more agile environments without many compliance issues. I can imagine a compliance-heavy environment, like health care or financial services, would especially benefit from this structure"*.[survey feedback]
- *"It would be great if the guidelines could adapt to a company's capabilities. Startups might consider some parts of the guidelines hard to follow"*.[survey feedback]
- *"Creating an entry-level version would help provide a quick introduction before delving into more detailed versions"*.[FG1-03]
- *"Exploring the relationships between guideline items would be useful, as many topics recur throughout"*.[FG1-02]

6.3 SUMMARY

This chapter presents the proposed guidelines V2, which include improvements from the evaluation stage, along with their results and discussions. These guidelines are the main contribution of this doctoral thesis, resulting from extensive research that included an exploratory study, a grey literature review, and feedback from practitioners. They are designed to tackle the complexities and challenges inherent in DSP applications, offering structured guidance to support testing practices in this area.

The evaluation methods comprised three focal group discussions with nine participants and a survey involving 22 participants. Participants provided feedback, emphasizing the guidelines' perceived relevance, areas for improvement, and applicability. The evaluation suggests that the guidelines are both relevant and practical in an industrial context.

7 CONCLUSION

This chapter presents the final considerations of this thesis, including the contributions achieved and indications for future works.

7.1 FINAL CONSIDERATIONS

This thesis' initial research direction and motivation came from the realization that software testing in the context of DSP is an emerging topic in the industry and little explored by research, needing more studies and consolidated documentation. No literature reviews or comprehensive studies encapsulating the main aspects of this field were found. The subject had received attention in IT forums, blogs, and events, and the emergence of several tools in the DSP context was noted. However, despite the abundance of information and technical content available in informal sources, this content was spread across the internet and was not cohesive or consolidated. Finally, there was a lack of unified documentation, such as guidelines or roadmaps, to guide testing activities within the DSP context.

To delve into this research field, we employed qualitative methods that validated the industry's interest in the topic, improved our understanding of the practical aspect of the field, identified challenges, and mapped the state-of-practice in DSP application testing. The exploratory study gathered significant insights revealing recurring issues and testing approaches employed by companies. It also indicated areas that deserve further exploration in future investigation phases.

The GLR study curated and synthesised intrinsic knowledge about DSP practices from 154 informal sources into a formal publication. This study revealed the industry's progress in testing DSP applications, spotlighting the in-house techniques to meet their demands. It also highlighted industry-developed approaches, processes, and tools that are missing in the academic literature. The study collected a substantial dataset that supported the development of the guidelines in the subsequent phase.

The primary contribution of this thesis is the development of guidelines for testing DSP applications derived from technical insights into testing practices extracted by the exploratory study and the GLR. We evaluated the guidelines, consulting specialists with hands-on experience to assess their perceived strengths, limitations, and industrial applicability. The evalu-

ation employed two methods—focus groups and surveys—whose results were triangulated to enhance the validity of our conclusions. The feedback suggests the proposed guidelines are helpful and applicable in assisting testing activities in the industrial setting, also promoting updates and refinements to the guidelines.

After evaluating the research conducted and the methods applied, the results substantiate the proposed thesis hypothesis: "It is feasible to formulate valuable guidelines for the industry grounded on diverse pieces of knowledge and experiences from practitioners, which have been systematically collected, selected, and synthesised through appropriate research methods." This confirmation of the hypothesis affirms the effectiveness of the research methods in compiling practitioner insights. It emphasises the significance of systematic and rigorous approaches in software engineering research, demonstrating their ability to produce outcomes that were considered relevant by practitioners. Additionally, the results highlight the role of academic research in addressing industry-specific challenges, potentially fostering increased collaboration between academics and industry professionals. This collaboration could lead to the validation and refinement of academic theories and frameworks through their practical application, bridging the gap between theoretical research and industry needs.

7.2 SUMMARY OF MAIN CONTRIBUTIONS

Here, we pinpoint this thesis's contributions:

- **Consolidation of practical knowledge:** This includes the selection and synthesis of practical insights, some of which may not have been previously documented, obtained through interviews and surveys in the exploratory study. Additionally, the knowledge dispersed across various informal publications was collected through the GLR. The publications resulting from these studies serve as a significant resource for both practitioners and researchers in the field. We highlight that, to our knowledge, no equivalent studies exist. Therefore, our work contributes to bridging the identified knowledge gap between academia and industry.

- **Provisioning of testing guidelines for the DSP community:** We have developed comprehensive testing guidelines perceived as useful and applicable by practitioners. These guidelines have been made accessible in a web-based format at <http://datastreamtesting.space>, representing a tangible research contribution to the DSP community.

- **Employing sound methodologies for investigating emerging industry topics:** This thesis also represents a case of using research methodologies to deeply investigate practical knowledge related to emerging topics in the industrial context.

7.3 PUBLICATIONS

- **VIANNA, A. S.**; CRUZ, M. O.; BARBOSA, L.; GAMA, K. Análise do impacto de chuvas na velocidade média do transporte público coletivo de ônibus em recife. In:SBC.Anais do I Workshop Brasileiro de Cidades Inteligentes. [S.l.], 2018.
- **VIANNA, A. S.**; FERREIRA, W.; GAMA, K. An exploratory study of how specialists deal with testing in data stream processing applications. In: IEEE.2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).[S.l.], p. 1–6. 2019.
- **VIANNA, A. S.**; KAMEI, F. K.; GAMA; K., ZIMERLE, C.; NETO, J. A. A Grey Literature Review on Data Stream Processing Applications Testing. In: Journal of Systems and Software, p. 111744, 2023.

7.4 FUTURE WORK

As an evolution of this research, potential research directions in future work:

- **Practical case study on the guidelines application:** Carry out a practical case study using the proposed guidelines in a real industrial context. The primary objective could be to validate whether the perceptions collected from evaluation participants will remain in real-world settings. Additionally, this study will evaluate the effectiveness of the guidelines in real-world projects and identify any unique challenges or considerations that only emerge in the practical environment.
- **Investigating the impact of architectural decisions on testing in DSP context:** Future studies may investigate how architectural choices within the DSP context, such as Lambda and Kappa, influence testing strategies and planning. Recognizing that architectural configurations inherently affect testability, research may aim to uncover the relationship between specific architectural patterns and the ease or complexity of testing

processes. This connection was reported in focus group discussions regarding the need for architectural validation.

- **Privacy and security in DSP testing environments:** Explore technical privacy and data security solutions within test environments in the DSP context. Recognizing the constraints imposed on the use of real data by regulations like GDPR and LGPD, the investigation could study the efficacy of current industry practices and explore innovative approaches to data handling in testing scenarios. The aim is to develop methodologies that ensure data protection in testing environments.

- **Advancements in al-driven synthetic data creation for DSP testing:** Investigate the use of state-of-the-art artificial intelligence methods in synthetic testing data generation tools. The study may aim to develop and validate tools that emulate complex real-world data scenarios, promoting more accurate and comprehensive testing outcomes. This future research responds to insights from the GLR, which identified the creation of realistic synthetic data as a challenge within the DSP domain.

REFERENCES

ADAMS, J.; HILLIER-BROWN, F. C.; MOORE, H. J.; LAKE, A. A.; ARAUJO-SOARES, V.; WHITE, M.; SUMMERBELL, C. Searching and synthesising 'grey literature'and 'grey information'in public health: critical reflections on three case studies. *Systematic reviews*, BioMed Central, v. 5, n. 1, p. 1–11, 2016.

ADAMS, R. J.; SMART, P.; HUFF, A. S. Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies. *International Journal of Management Reviews*, Wiley Online Library, v. 19, n. 4, p. 432–454, 2017.

ADZIC, G. *Specification by example: how successful teams deliver the right software*. [S.l.]: Simon and Schuster, 2011.

AHMAD, T.; IQBAL, J.; ASHRAF, A.; TRUSCAN, D.; PORRES, I. Model-based testing using uml activity diagrams: A systematic mapping study. *Computer Science Review*, Elsevier, v. 33, p. 98–112, 2019.

AKBER, S. M. A.; CHEN, H.; JIN, H. Fatm: A failure-aware adaptive fault tolerance model for distributed stream processing systems. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 33, n. 10, p. e6167, 2021.

AKIDAU, T.; BEGOLI, E.; CHERNYAK, S.; HUESKE, F.; KNIGHT, K.; KNOWLES, K.; MILLS, D.; SOTOLONGO, D. *Watermarks in stream processing systems: Semantics and comparative analysis of apache flink and google cloud dataflow*. [S.l.], 2021.

AKIDAU, T.; BRADSHAW, R.; CHAMBERS, C.; CHERNYAK, S.; FERNÁNDEZ-MOCTEZUMA, R. J.; LAX, R.; MCVEETY, S.; MILLS, D.; PERRY, F.; SCHMIDT, E. et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. 2015.

ALADEV, R. *How to Do Kafka Testing With JMeter*. 2021. Accessed: 2022-07-19. Disponível em: <https://www.blazemeter.com/blog/kafka-testing>.

ALQUIZA, J. *Sangrenel*. 2021. <https://github.com/jamiealquiza/sangrenel>.

ALSAQAF, W.; DANEVA, M.; WIERINGA, R. Quality requirements challenges in the context of large-scale distributed agile: An empirical study. *Information and software technology*, Elsevier, v. 110, p. 39–55, 2019.

ALSHAMRANI, S.; WASEEM, Q.; ALHARBI, A.; ALOSAIMI, W.; TURABIEH, H.; ALYAMI, H. An efficient approach for storage of big data streams in distributed stream processing systems. *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 11, n. 5, 2020.

Amazon Kinesis. *Test Your Streaming Data Solution with the New Amazon Kinesis Data Generator*. 2017. [Accessed: 2021-02-20]. Disponível em: <https://aws.amazon.com/pt/blogs/big-data/test-your-streaming-data-solution-with-the-new-amazon-kinesis-data-generator/>.

AMMENWERTH, E.; ILLER, C.; MANSMANN, U. Can evaluation studies benefit from triangulation? a case study. *International journal of medical informatics*, Elsevier, v. 70, n. 2-3, p. 237–248, 2003.

AMPATZOGLOU, A.; BIBI, S.; AVGERIOU, P.; CHATZIGEORGIOU, A. Guidelines for managing threats to validity of secondary studies in software engineering. In: *Contemporary Empirical Methods in Software Engineering*. Springer, 2020. p. 415–441. ISBN 978-3-030-32489-6. Disponível em: <https://doi.org/10.1007/978-3-030-32489-6_15>.

ANDREWS, A.; FRANCE, R.; GHOSH, S.; CRAIG, G. Test adequacy criteria for uml design models. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 13, n. 2, p. 95–127, 2003.

Apache Flink. *Apache Flink 1.12 Documentation: Testing*. 2021. [Accessed: 2021-02-20]. Disponível em: <https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/stream/testing.html>.

APACHE, M. E. *Kcat*. 2021. <https://github.com/edenhill/kcat>.

ARORA, V.; BHATIA, R.; SINGH, M. A systematic review of approaches for testing concurrent programs. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 28, n. 5, p. 1572–1611, 2016.

ASSUNCAO, M. D. de; VEITH, A. da S. et al. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, Elsevier, 2018.

ASTORGA, D. del R.; DOLZ, M. F.; FERNÁNDEZ, J.; GARCÍA, J. D. Paving the way towards high-level parallel pattern interfaces for data stream processing. *Future Generation Computer Systems*, Elsevier, v. 87, p. 228–241, 2018.

AUGUST, T.; CHEN, W.; ZHU, K. Competition among proprietary and open-source software firms: the role of licensing in strategic contribution. *Management Science*, INFORMS, v. 67, n. 5, p. 3041–3066, 2021.

AUTHORJAPPS. *Kafka Testing Hello World examples*. 2019. Disponível em: <https://github.com/authorjapps/hello-kafka-stream-testing>.

BABAEI, M.; DINGEL, J. Efficient replay-based regression testing for distributed reactive systems in the context of model-driven development. In: IEEE. *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. [S.l.], 2021. p. 89–100.

BABCOCK, B.; BABU, S.; DATAR, M. et al. Models and issues in data stream systems. In: ACM. *Proceedings of the twenty-first SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. [S.l.], 2002.

BABU, S.; WIDOM, J. Continuous queries over data streams. *ACM Sigmod Record*, ACM, v. 30, n. 3, p. 109–120, 2001.

BACKDATA. *Fluent Kafka Streams Tests*. 2021. <https://github.com/bakdata/fluent-kafka-streams-tests>.

BALAZINSKA, M.; HWANG, J.-H.; SHAH, M. A. Fault tolerance and high availability in data stream management systems. *Encyclopedia of Database Systems*, v. 11, p. 57, 2009.

BAQASAH, A.; PARDEDE, E.; RAHAYU, W. Maintaining schema versions compatibility in cloud applications collaborative framework. *World Wide Web*, Springer, v. 18, n. 6, p. 1541–1577, 2015.

BARCHENKOV, I. *Ecto Stream Factory*. 2019. <https://github.com/ibarchenkov/ecto_stream_factory>.

BARESI, L.; PEZZE, M. An introduction to software testing. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 148, n. 1, p. 89–111, 2006.

BARR, E. T.; HARMAN, M.; MCMINN, P.; SHAHBAZ, M.; YOO, S. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, IEEE, v. 41, n. 5, p. 507–525, 2014.

BARROS, M. D.; SHIAU, J.; SHANG, C.; GIDEWALL, K.; SHI, H.; FORSMANN, J. Web services wind tunnel: On performance testing large-scale stateful web services. In: IEEE. *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. [S.l.], 2007. p. 612–617.

BASIRI, A.; BEHNAM, N.; ROOIJ, R. D.; HOCHSTEIN, L.; KOSEWSKI, L.; REYNOLDS, J.; ROSENTHAL, C. Chaos engineering. *IEEE Software*, IEEE, v. 33, n. 3, p. 35–41, 2016.

BATH, G. The next generation tester: Meeting the challenges of a changing it world. *The Future of Software Quality Assurance*, Springer International Publishing, p. 15–26, 2020.

BATTINA, D. S. Devops, a new approach to cloud development & testing. *International Journal of Emerging Technologies and Innovative Research (www. jetir. org), ISSN*, p. 2349–5162, 2020.

BEGUM, S. H.; NAUSHEEN, F. A comparative analysis of differential privacy vs other privacy mechanisms for big data. *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, p. 512–516, 2018.

BEHJATI, R.; ARISHOLM, E.; BEDREGAL, M.; TAN, C. Synthetic test data generation using recurrent neural networks: a position paper. In: IEEE. *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. [S.l.], 2019. p. 22–27.

BENKHELIFA, E.; HANI, A. B.; WELSH, T.; MTHUNZI, S.; GUEGAN, C. G. Virtual environments testing as a cloud service: a methodology for protecting and securing virtual infrastructures. *IEEE Access*, IEEE, v. 7, p. 108660–108676, 2019.

BERNARDO, A.; VALLE, E. D. Vfc-smote: very fast continuous synthetic minority oversampling for evolving data streams. *Data Mining and Knowledge Discovery*, Springer, v. 35, n. 6, p. 2679–2713, 2021.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KIRKBY, R.; GAVALDÀ, R. New ensemble methods for evolving data streams. In: ACM. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2009. p. 139–148.

BISWAS, S.; MALL, R.; SATPATHY, M.; SUKUMARAN, S. Regression test selection techniques: A survey. *Informatica*, v. 35, n. 3, 2011.

BLANCHI, C.; PETRONE, J. Distributed interoperable metadata registry. *D-Lib Magazine*, v. 7, n. 12, p. 1082–9873, 2001.

BORODAY, S.; PETRENKO, A.; GROZ, R. Can a model checker generate tests for non-deterministic systems? *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 190, n. 2, p. 3–19, 2007.

BOURQUE, P.; FAIRLEY, R. E. (Ed.). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0. IEEE Computer Society, 2014. ISBN 978-0-7695-5166-1. Disponível em: <http://www.swebok.org/>.

BREEN, R. L. A practical guide to focus-group research. *Journal of geography in higher education*, Taylor & Francis, v. 30, n. 3, p. 463–475, 2006.

BU, Y.; CHEN, L.; FU, A. W.-C.; LIU, D. Efficient anomaly monitoring over moving object trajectory streams. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2009. p. 159–168.

BUCHGEHER, G.; FISCHER, S.; MOSER, M.; PICHLER, J. An early investigation of unit testing practices of component-based software systems. In: IEEE. *2020 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST)*. [S.l.], 2020. p. 12–15.

BUCHMANN, A.; KOLDEHOFE, B. Complex event processing. *it-Information Technology*, v. 51, n. 5, p. 241, 2009.

BUSTAMANTE, R.; GARCÉS, K. Managing evolution of api-driven iot devices through adaptation chains. In: *CIbSE*. [S.l.: s.n.], 2020. p. 85–95.

CAPPELLARI, P.; CHUN, S. A.; ROANTREE, M. Ise: A high performance system for processing data streams. In: *DATA*. [S.l.: s.n.], 2016. p. 13–24.

CARBONE, P.; FRAGKOULIS, M.; KALAVRI, V.; KATSIFODIMOS, A. Beyond analytics: The evolution of stream processing systems. In: *Proceedings of the 2020 ACM SIGMOD international conference on Management of data*. [S.l.: s.n.], 2020. p. 2651–2658.

CARBONE, P.; KATSIFODIMOS, A.; EWEN, S.; MARKL, V.; HARIDI, S.; TZOUMAS, K. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 38, n. 4, 2015.

CARBONE, P.; TRAUB, J.; KATSIFODIMOS, A.; HARIDI, S.; MARKL, V. Cutty: Aggregate sharing for user-defined windows. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. [S.l.: s.n.], 2016. p. 1201–1210.

CARDELLINI, V.; PRESTI, F. L.; NARDELLI, M.; RUSSO, G. R. Runtime adaptation of data stream processing systems: The state of the art. *ACM Computing Surveys*, ACM New York, NY, v. 54, n. 11s, p. 1–36, 2022.

CARZANIGA, A.; GOFFI, A.; GORLA, A.; MATTAVELLI, A.; PEZZÈ, M. Cross-checking oracles from intrinsic software redundancy. In: *Proceedings of the 36th International Conference on Software Engineering*. [S.l.: s.n.], 2014. p. 931–942.

CHARAF, M. E. H.; AZZOUZI, S. Timed distributed testing rules for the distributed test architecture. In: IEEE. *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*. [S.l.], 2016. p. 314–319.

CHARMAZ, K.; BELGRAVE, L. L. Grounded theory. *The Blackwell encyclopedia of sociology*, Wiley Online Library, 2007.

CHATTERJEE, S.; MORIN, C. Experimental study on the performance and resource utilization of data streaming frameworks. In: IEEE. *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. [S.l.], 2018. p. 143–152.

CHEN, G.; BAI, G.; ZHANG, C.; WANG, J.; NI, K.; CHEN, Z. Big data system testing method based on chaos engineering. In: IEEE. *2022 IEEE 12th International Conference on Electronics Information and Emergency Communication (ICEIEC)*. [S.l.], 2022. p. 210–215.

CHEN, T.-H.; SYER, M. D.; SHANG, W.; JIANG, Z. M.; HASSAN, A. E.; NASSER, M.; FLORA, P. Analytics-driven load testing: An industrial experience report on load testing of large-scale systems. In: IEEE. *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. [S.l.], 2017. p. 243–252.

CHEN, W.; PAIK, I.; LI, Z. Cost-aware streaming workflow allocation on geo-distributed data centers. *IEEE Transactions on Computers*, IEEE, v. 66, n. 2, p. 256–271, 2016.

CHEN, Y.; ZHANG, S.; GUO, Q.; LI, L.; WU, R.; CHEN, T. Deterministic replay: A survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 48, n. 2, p. 1–47, 2015.

CHEN, Y. T.; GOPINATH, R.; TADAKAMALLA, A.; ERNST, M. D.; HOLMES, R.; FRASER, G.; AMMANN, P.; JUST, R. Revisiting the relationship between fault detection, test adequacy criteria, and test set size. In: *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. [S.l.: s.n.], 2020. p. 237–249.

CHERNIACK, M.; BALAKRISHNAN, H.; BALAZINSKA, M.; CARNEY, D.; CETINTEMEL, U.; XING, Y.; ZDONIK, S. B. Scalable distributed stream processing. In: *CIDR*. [S.l.: s.n.], 2003. v. 3, p. 257–268.

CHINTAPALLI, S.; DAGIT, D.; EVANS, B.; FARIVAR, R.; GRAVES, T.; HOLDERBAUGH, M.; LIU, Z.; NUSBAUM, K.; PATIL, K.; PENG, B. J. et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In: IEEE. *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. [S.l.], 2016. p. 1789–1792.

CHOI, C.; ESPOSITO, C.; WANG, H.; LIU, Z.; CHOI, J. Intelligent power equipment management based on distributed context-aware inference in smart cities. *IEEE Communications Magazine*, IEEE, v. 56, n. 7, p. 212–217, 2018.

CIRCLE, F. *Jackdaw - Test Machine*. 2021. <https://github.com/FundingCircle/jackdaw>.

COMMUNITY. *Embedded Kafka*. 2021. <https://github.com/embeddedkafka/embedded-kafka>.

COMMUNITY. *ScalaTest*. 2021. <https://github.com/scalatest/scalatest>.

CONDIE, T.; CONWAY, N.; ALVARO, P.; HELLERSTEIN, J. M.; ELMELEEGY, K.; SEARS, R. Mapreduce online. In: *Nsdi*. [S.l.: s.n.], 2010. v. 10, n. 4, p. 20.

COOPER, G. H.; KRISHNAMURTHI, S. Embedding dynamic dataflow in a call-by-value language. In: SPRINGER. *European Symposium on Programming*. [S.l.], 2006.

CORRAL-PLAZA, D.; MEDINA-BULO, I.; ORTIZ, G.; BOUBETA-PUIG, J. A stream processing architecture for heterogeneous data sources in the internet of things. *Computer Standards & Interfaces*, Elsevier, v. 70, p. 103426, 2020.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, ACM, v. 44, n. 3, p. 15, 2012.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 44, n. 3, p. 1–62, 2012.

DÁVID, I.; RÁTH, I.; VARRÓ, D. Foundations for streaming model transformations by complex event processing. *Software & Systems Modeling*, Springer, v. 17, p. 135–162, 2018.

DEAK, A.; STÅLHANE, T.; SINDRE, G. Challenges and strategies for motivating software testing personnel. *Information and software Technology*, Elsevier, v. 73, p. 1–15, 2016.

DEHAAN, M.; INC., R. H. *Ansible*. 2021. <https://github.com/ansible/ansible>.

DEMŠAR, J.; BOSNIĆ, Z. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, Elsevier, v. 92, p. 546–559, 2018.

DENZIN, N.; BRYANT, A.; CHARMAZ, K. Grounded theory and the politics of interpretation. *The Sage Handbook of Grounded Theory (London: Sage, 2007)*, p. 454–471, 2007.

DIAZ, S. M.; SOUZA, P. S.; SOUZA, S. R. Structural testing for communication events into loops of message-passing parallel programs. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 33, n. 18, p. e6082, 2021.

DILSHAD, R. M.; LATIF, M. I. Focus group interview as a tool for qualitative research: An analysis. *Pakistan Journal of Social Sciences*, v. 33, n. 1, p. 191–198, 2013.

DINGSØYR, T.; DYBÅ, T.; GJERTSEN, M.; JACOBSEN, A. O.; MATHISEN, T.-E.; NORDFJORD, J. O.; RØE, K.; STRAND, K. Key lessons from tailoring agile methods for large-scale software development. *IT Professional*, IEEE, v. 21, n. 1, p. 34–41, 2019.

DIVÁN, M. J.; REYNOSO, M. L. S. Relocating the load-shedding strategy in the data stream processing architecture. In: IEEE. *2020 IEEE Congreso Bienal de Argentina (ARGENCON)*. [S.l.], 2020. p. 1–8.

DUMITRESCU, C.; RAICU, I.; RIPEANU, M.; FOSTER, I. Diperf: An automated distributed performance testing framework. In: IEEE. *Fifth IEEE/ACM International Workshop on Grid Computing*. [S.l.], 2004. p. 289–296.

EBERT, C.; RAY, R. Test-driven requirements engineering. *IEEE Software*, IEEE, v. 38, n. 1, p. 16–24, 2020.

EISMANN, S.; BEZEMER, C.-P.; SHANG, W.; OKANOVIĆ, D.; HOORN, A. van. Microservices: A performance tester's dream or nightmare? In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. [S.l.: s.n.], 2020. p. 138–149.

ESPINOSA, C. V.; MARTIN-MARTIN, E.; RIESCO, A.; RODRÍGUEZ-HORTALÁ, J. Flinkcheck: property-based testing for apache flink. *IEEE Access*, IEEE, v. 7, p. 150369–150382, 2019.

EVERETT, G. D.; JR, R. M. Software testing. *Testing Across the Entire*, 2007.

FARAGLIA, D. *Faker (Python)*. 2021. <https://github.com/joke2k/faker>.

FAWCETT, T.; PROVOST, F. Activity monitoring: Noticing interesting changes in behavior. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 1999. p. 53–62.

FEILER, P. H. Model-based validation of safety-critical embedded systems. In: IEEE. *2010 IEEE Aerospace Conference*. [S.l.], 2010. p. 1–10.

FELDERER, M.; RUSSO, B.; AUER, F. On testing data-intensive software systems. In: *Security and Quality in Cyber-Physical Systems Engineering*. [S.l.]: Springer, 2019. p. 129–148.

FIGUEIRAS, P.; HERGA, Z.; GUERREIRO, G.; ROSA, A.; COSTA, R.; JARDIM-GONÇALVES, R. Real-time monitoring of road traffic using data stream mining. In: IEEE. *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. [S.l.], 2018. p. 1–8.

FILIP, I.-D.; POSTOACA, A. V.; STOCHITOIU, R.-D.; NEATU, D.-F.; NEGRU, C.; POP, F. Data capsule: Representation of heterogeneous data in cloud-edge computing. *IEEE Access*, IEEE, v. 7, p. 49558–49567, 2019.

FLOREA, R.; STRAY, V. A qualitative study of the background, skill acquisition, and learning preferences of software testers. In: *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.: s.n.], 2020. p. 299–305.

FORBES. *Forbes Top 100 Digital Companies*. 2021. Accessed: 2021-02-10. Disponível em: <https://www.forbes.com/top-digital-companies/list>.

FORTUNE. *Fortune 500*. 2021. Accessed: 2021-02-10. Disponível em: <http://fortune.com/fortune500>.

FOUNDATION, A. *Flink Testing Utilities - Apache Flink*. 2021. <https://mvnrepository.com/artifact/org.apache.flink/flink-test-utils>.

FOUNDATION, A. *JMeter*. 2021. <https://github.com/apache/jmeter>.

FOUNDATION, A. *Kafka Streams Testing Utilities*. 2021. <https://mvnrepository.com/artifact/org.apache.kafka/kafka-streams-test-utils>.

FOUNDATION, A. S. *Apache Flink mail archive*. 2022. Accessed: 2022-07-19. Disponível em: <https://lists.apache.org/list?dev@flink.apache.org:lte=1M:tests>.

FRAGKOULIS, M.; CARBONE, P.; KALAVRI, V.; KATSIFODIMOS, A. A survey on the evolution of stream processing systems. *The VLDB Journal*, Springer, p. 1–35, 2023.

FU, H.-H.; LIN, D. K.; TSAI, H.-T. Damping factor in google page ranking. *Applied Stochastic Models in Business and Industry*, Wiley Online Library, v. 22, n. 5-6, p. 431–444, 2006.

GAMOV, V. *I Don't Always Test My Streams, But When I Do, I Do it in Production*. 2020. Disponível em: <https://www.confluent.io/resources/kafka-summit-2020/i-dont-always-test-my-streams-but-when-i-do-i-do-it-in-production>.

GARCIA, A. M.; GRIEBLER, D.; SCHEPKE, C.; FERNANDES, L. G. L. Evaluating micro-batch and data frequency for stream processing applications on multi-cores. In: IEEE. *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. [S.l.], 2022. p. 10–17.

GARCIA, A. M.; GRIEBLER, D.; SCHEPKE, C.; FERNANDES, L. G. Spbench: a framework for creating benchmarks of stream processing applications. *Computing*, Springer, p. 1–23, 2022.

GAROFALAKIS, M.; GEHRKE, J.; RASTOGI, R. Data stream management: A brave new world. In: *Data Stream Management: Processing High-Speed Data Streams*. [S.l.]: Springer, 2016. p. 1–9.

GAROUSI, V.; ELBERZHAGER, F. Test automation: not just for test execution. *IEEE Software*, IEEE, v. 34, n. 2, p. 90–96, 2017.

GAROUSI, V.; FELDERER, M.; FERNANDES, J. M.; PFAHL, D.; MÄNTYLÄ, M. V. Industry-academia collaborations in software engineering: An empirical analysis of challenges, patterns and anti-patterns in research projects. In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. [S.l.: s.n.], 2017. p. 224–229.

GAROUSI, V.; FELDERER, M.; KUHRMANN, M.; HERKILOĞLU, K.; ELDH, S. Exploring the industry's challenges in software testing: An empirical study. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 32, n. 8, p. e2251, 2020.

GAROUSI, V.; FELDERER, M.; MäNTYLä, M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, v. 106, p. 101 – 121, 2019. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584918301939>.

GAROUSI, V.; MÄNTYLÄ, M. V. When and what to automate in software testing? a multi-vocal literature review. *Information and Software Technology*, Elsevier, v. 76, p. 92–117, 2016.

GÄRTNER, M. *ATDD by example: a practical guide to acceptance test-driven development*. [S.l.]: Addison-Wesley, 2012.

GAZZOLA, L.; GOLDSTEIN, M.; MARIANI, L.; MOBILIO, M.; SEGALL, I.; TUNDO, A.; USSI, L. Exvivomicrotest: Exvivo testing of microservices. *Journal of Software: Evolution and Process*, Wiley Online Library, p. e2452, 2022.

GELDENHUYS, M. K.; PFISTER, B. J.; SCHEINERT, D.; KAO, O.; THAMSEN, L. Khaos: Dynamically optimizing checkpointing for dependable distributed stream processing. *arXiv preprint arXiv:2109.02340*, 2021.

GHAZI, A. N.; PETERSEN, K.; REDDY, S. S. V. R.; NEKKANTI, H. Survey research in software engineering: Problems and mitigation strategies. *IEEE Access*, IEEE, v. 7, p. 24703–24718, 2018.

GLASER, B. G. *Basics of grounded theory analysis: Emergence vs forcing*. [S.l.]: Sociology press, 1992.

GLASER, B. G.; STRAUSS, A. L. *Discovery of grounded theory: Strategies for qualitative research*. [S.l.]: Routledge, 2017.

GMBH, I. *Imixs Workflow - Open Source Workflow with BPMN 2.0*. 2023. Disponível em: <https://www.imixs.org/>. Acesso em: 20 nov. 2023.

GODEFROID, P.; LEHMANN, D.; POLISHCHUK, M. Differential regression testing for rest apis. In: *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. [S.l.: s.n.], 2020. p. 312–323.

GODIN, K.; STAPLETON, J.; KIRKPATRICK, S. I.; HANNING, R. M.; LEATHERDALE, S. T. Applying systematic review search methods to the grey literature: a case study examining guidelines for school-based breakfast programs in canada. *Systematic reviews*, BioMed Central, v. 4, n. 1, p. 1–10, 2015.

GOLAB, L.; ÖZSU, M. T. Issues in data stream management. *ACM Sigmod Record*, ACM New York, NY, USA, v. 32, n. 2, p. 5–14, 2003.

GOLAB, L.; OZSU, M. T. *Data stream management*. [S.l.]: Springer Nature, 2022.

GONG, L.; LI, Y.; CHEN, H.; XU, X.; LIU, F. Scenario-based reliability testing methods for smart grid dispatching and control system. In: IEEE. *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. [S.l.], 2021. p. 656–663.

GOOGLE. *Google's Search Algorithm and Ranking System*. 2021. Accessed: 2021-02-13. Disponível em: <https://www.google.com/search/howsearchworks/algorithms/>.

GOULDING, C. Grounded theory: the missing methodology on the interpretivist agenda. *Qualitative Market Research: an international journal*, MCB UP Ltd, v. 1, n. 1, p. 50–57, 1998.

GREGORY, J.; CRISPIN, L. *More agile testing: learning journeys for the whole team*. [S.l.]: Addison-Wesley Professional, 2014.

GRULICH, P. M.; TRAUB, J.; BRESS, S.; KATSIFODIMOS, A.; MARKL, V.; RABL, T. Generating reproducible out-of-order data streams. In: *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*. [S.l.: s.n.], 2019. p. 256–257.

GU, R.; ZHOU, Y.; WANG, Z.; YUAN, C.; HUANG, Y. Penguin: Efficient query-based framework for replaying large scale historical data. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 29, n. 10, p. 2333–2345, 2018.

GULZAR, M. A.; ZHU, Y.; HAN, X. Perception and practices of differential testing. In: IEEE. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. [S.l.], 2019. p. 71–80.

GUNAWI, H. S.; HAO, M.; LEESATAPORNWONGSA, T.; PATANA-ANAKE, T.; DO, T.; ADITYATAMA, J.; ELIAZAR, K. J.; LAKSONO, A.; LUKMAN, J. F.; MARTIN, V. et al. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In: *Proceedings of the ACM symposium on cloud computing*. [S.l.: s.n.], 2014. p. 1–14.

GÜRCAN, F.; BERIGEL, M. Real-time processing of big data streams: Lifecycle, tools, tasks, and challenges. In: IEEE. *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. [S.l.], 2018. p. 1–6.

GUTIÉRREZ-MADROÑAL, L.; GARCÍA-DOMÍNGUEZ, A.; MEDINA-BULO, I. Evolutionary mutation testing for iot with recorded and generated events. *Software: Practice and Experience*, Wiley Online Library, v. 49, n. 4, p. 640–672, 2019.

GUTIÉRREZ-MADROÑAL, L.; MEDINA-BULO, I.; DOMÍNGUEZ-JIMÉNEZ, J. J. Iot–teg: Test event generator system. *Journal of Systems and Software*, Elsevier, v. 137, p. 784–803, 2018.

GüNTHER, M. *Kafka for Junit*. 2021. <https://github.com/mguenther/kafka-junit>.

HALEBY, J.; COMMUNITY. *Awaitility Library*. 2021. <https://github.com/awaitility/awaitility>.

HANAMANTHRAO, R.; THEJASWINI, S. Real-time clickstream data analytics and visualization. In: IEEE. *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. [S.l.], 2017. p. 2139–2144.

HANAWA, T.; BANZAI, T.; KOIZUMI, H.; KANBAYASHI, R.; IMADA, T.; SATO, M. Large-scale software testing environment using cloud computing technology for dependable parallel and distributed systems. In: IEEE. *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. [S.l.], 2010. p. 428–433.

HARSH, P.; LASZKOWSKI, J. F. R.; EDMONDS, A.; THANH, T. Q.; PAULS, M.; VLASKOVSKI, R.; AVILA-GARCÍA, O.; PAGES, E.; BELLAS, F. G.; CARRILLO, M. G. Cloud enablers for testing large-scale distributed applications. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. [S.l.: s.n.], 2019. p. 35–42.

HASAN, M.; ORGUN, M. A. et al. A survey on real-time event detection from the twitter data stream. *Journal of Information Science*, SAGE Publications Sage UK: London, England, 2018.

HASHICORP. *Terraform*. 2021. <https://github.com/hashicorp/terraform>.

HASHIMOV, E. *Qualitative Data Analysis: A Methods Sourcebook and The Coding Manual for Qualitative Researchers: Matthew B. Miles, A. Michael Huberman, and Johnny Saldaña. Thousand Oaks, CA: SAGE, 2014. 381 pp. Johnny Saldaña. Thousand Oaks, CA: SAGE, 2013. 303 pp.* [S.l.]: Taylor & Francis, 2015.

HERZIG, K.; GREILER, M.; CZERWONKA, J.; MURPHY, B. The art of testing less without sacrificing quality. In: IEEE. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. [S.l.], 2015. v. 1, p. 483–493.

HIERONS, R. M.; URAL, H. Checking sequences for distributed test architectures. *Distributed Computing*, Springer, v. 21, n. 3, p. 223–238, 2008.

HILL, J. H.; TURNER, H. A.; EDMONDSON, J. R.; SCHMIDT, D. C. Unit testing non-functional concerns of component-based distributed systems. In: IEEE. *2009 International Conference on Software Testing Verification and Validation*. [S.l.], 2009. p. 406–415.

HIRZEL, M.; SOULÉ, R.; SCHNEIDER, S.; GEDIK, B.; GRIMM, R. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 46, n. 4, p. 1–34, 2014.

HOQUE, S.; MIRANSKYY, A. Architecture for analysis of streaming data. In: IEEE. *2018 IEEE International Conference on Cloud Engineering (IC2E)*. [S.l.], 2018. p. 263–269.

HOSSAIN, S. B.; DWYER, M. B.; ELBAUM, S.; NGUYEN-TUONG, A. Measuring and mitigating gaps in structural testing. In: IEEE. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. [S.l.], 2023. p. 1712–1723.

HOSSAYNI, H.; KHAN, I.; CRESPI, N. Data anonymization for maintenance knowledge sharing. *IT Professional*, IEEE, v. 23, n. 5, p. 23–30, 2021.

HUMMEL, O.; EICHELBERGER, H.; GILOJ, A.; WERLE, D.; SCHMID, K. A collection of software engineering challenges for big data system development. In: IEEE. *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.], 2018. p. 362–369.

HYNNINEN, T.; KASURINEN, J.; KNUTAS, A.; TAIPALE, O. Software testing: Survey of the industry practices. In: IEEE. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.], 2018. p. 1449–1454.

IGLESIAS, F.; OJDANIC, D.; HARTL, A.; ZSEBY, T. Mdcstream: Stream data generator for testing analysis algorithms. In: *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools*. [S.l.: s.n.], 2020. p. 56–63.

IGLESIAS, G.; TALAVERA, E.; GONZÁLEZ-PRIETO, Á.; MOZO, A.; GÓMEZ-CANAVAL, S. Data augmentation techniques in time series domain: a survey and taxonomy. *Neural Computing and Applications*, Springer, v. 35, n. 14, p. 10123–10145, 2023.

IKONOMOVSKA, E.; LOSKOVSKA, S.; GJORGJEVIK, D. A survey of stream data mining. In: *Proceedings of 8th National Conference with International participation, ETAI*. [S.l.: s.n.], 2007. p. 19–21.

IMTIAZ, J.; SHERIN, S.; KHAN, M. U.; IQBAL, M. Z. A systematic literature review of test breakage prevention and repair techniques. *Information and Software Technology*, Elsevier, v. 113, p. 1–19, 2019.

INC., A. *Kinesis Data Generator*. 2021. <https://github.com/awslabs/amazon-kinesis-data-generator>.

INC., C. *Ksql-datagen*. 2017. <https://docs.confluent.io/5.4.0/ksql/docs/tutorials/generate-custom-test-data.html>.

INC., C. *Avro Random Generator*. 2021. <https://github.com/confluentinc/avro-random-generator>.

INC., C. *Confluent CLI*. 2021. <https://github.com/confluentinc/confluent-cli>.

INC., C. *Ducktape*. 2021. <https://github.com/confluentinc/ducktape>.

INC., C. *Kafka Datagen Connector*. 2021. <https://github.com/confluentinc/kafka-connect-datagen>.

ISAH, H.; ZULKERNINE, F. A scalable and robust framework for data stream ingestion. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 2900–2905.

ISO/IEC 25010. *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. 2011. Disponível em: <https://www.bibsonomy.org/bibtex/25951b0998b7eaea346d826fd77110a48/bcoldewey>.

ISO/IEC 25010:2011: Systems And Software Engineering – Systems And SoftWare Quality Requirements And Evaluation (Square) – System And Software Quality Models. 2011. Disponível em: <https://www.iso.org/standard/35733.html>.

JAFFARI, A.; YOO, C.-J.; LEE, J. Automatic test data generation using the activity diagram and search-based technique. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 10, n. 10, p. 3397, 2020.

JEDLITSCHKA, A.; CIOLKOWSKI, M.; PFAHL, D. Reporting experiments in software engineering. *Guide to advanced empirical software engineering*, Springer, p. 201–228, 2008.

JEDLITSCHKA, A.; PFAHL, D. Reporting guidelines for controlled experiments in software engineering. In: IEEE. *2005 International Symposium on Empirical Software Engineering, 2005.* [S.l.], 2005. p. 10–pp.

JENKINS. *Jenkins*. 2021. <https://github.com/jenkinsci/jenkins>.

JIANG, Z. M.; HASSAN, A. E. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, IEEE, v. 41, n. 11, p. 1091–1118, 2015.

JONES, C.; BONSIGNOUR, O. *The economics of software quality*. [S.l.]: Addison-Wesley Professional, 2011.

JONSEN, K.; JEHN, K. A. Using triangulation to validate themes in qualitative studies. *Qualitative Research in Organizations and Management: An International Journal*, Emerald Group Publishing Limited, v. 4, n. 2, p. 123–150, 2009.

JÚNIOR, E.; FARIAS, K.; SILVA, B. A survey on the use of uml in the brazilian industry. In: *Proceedings of the XXXV Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2021. p. 275–284.

KAISLER, S.; ARMOUR, F.; ESPINOSA, J. A.; MONEY, W. Big data: Issues and challenges moving forward. In: IEEE. *2013 46th Hawaii International Conference on System Sciences*. [S.l.], 2013. p. 995–1004.

KALLAS, K.; NIKSIC, F.; STANFORD, C.; ALUR, R. Diffstream: differential output testing for stream processing programs. *Proceedings of the ACM on Programming Languages*, ACM New York, NY, USA, v. 4, n. OOPSLA, p. 1–29, 2020.

KAMEI, F.; WIESE, I.; LIMA, C.; POLATO, I.; NEPOMUCENO, V.; FERREIRA, W.; RIBEIRO, M.; PENA, C.; CARTAXO, B.; PINTO, G.; SOARES, S. Grey literature in software engineering: A critical review. *Information and Software Technology*, p. 106609, 2021. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584921000860>.

KAMEI, F.; WIESE, I.; PINTO, G.; RIBEIRO, M.; SOARES, S. On the use of grey literature: A survey with the brazilian software engineering research community. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2020. p. 183–192.

KAPUR, P.; SHRIVASTAVA, A.; SINGH, O. When to release and stop testing of a software. *Journal of the Indian Society for Probability and Statistics*, Springer, v. 18, p. 19–37, 2017.

KARAU, H. *Spark Testing Base*. 2016. Disponível em: <https://github.com/holdenk/spark-testing-base>.

KARAU, H. *Spark Testing Base*. 2021. <https://github.com/holdenk/spark-testing-base>.

KARGAR, M. J.; HANIFIZADE, A. Automation of regression test in microservice architecture. In: IEEE. *2018 4th International Conference on Web Research (ICWR)*. [S.l.], 2018. p. 133–137.

KARIMOV, J.; RABL, T.; KATSIFODIMOS, A.; SAMAREV, R.; HEISKANEN, H.; MARKL, V. Benchmarking distributed stream data processing systems. In: IEEE. *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. [S.l.], 2018. p. 1507–1518.

KARLSSON, S.; ČAUŠEVIĆ, A.; SUNDMARK, D. Quickrest: Property-based test generation of openapi-described restful apis. In: IEEE. *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. [S.l.], 2020. p. 131–141.

KARLSSON, S.; ČAUŠEVIĆ, A.; SUNDMARK, D. Automatic property-based testing of graphql apis. In: IEEE. *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. [S.l.], 2021. p. 1–10.

KASAULI, R.; KNAUSS, E.; HORKOFF, J.; LIEBEL, G.; NETO, F. G. de O. Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software*, Elsevier, v. 172, p. 110851, 2021.

KASSAB, M.; LAPLANTE, P.; DEFRANCO, J.; NETO, V. V. G.; DESTEFANIS, G. Exploring the profiles of software testing jobs in the united states. *IEEE Access*, IEEE, v. 9, p. 68905–68916, 2021.

KENTHAPADI, K.; TRAN, T. T. L. Pripearl: A framework for privacy-preserving analytics and reporting at linkedin. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018.

KHAMASSI, I.; SAYED-MOUCHAWEH, M.; HAMMAMI, M.; GHÉDIRA, K. Discussion and review on evolving data streams and concept drift adapting. *Evolving systems*, Springer, v. 9, n. 1, p. 1–23, 2018.

KHANDAI, S.; ACHARYA, A. A.; MOHAPATRA, D. P. Prioritizing test cases using business criticalitytest value. *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 3, n. 5, 2012.

KHARE, K. *A Guide for Unit Testing in Apache Flink*. 2020. [Accessed: 2021-02-20]. Disponível em: <https://flink.apache.org/news/2020/02/07/a-guide-for-unit-testing-in-apache-flink.html>.

KHINE, A. A.; KHIN, H. W. Credit card fraud detection using online boosting with extremely fast decision tree. In: IEEE. *2020 IEEE Conference on Computer Applications (ICCA)*. [S.l.], 2020. p. 1–4.

KIM, S.; PARK, J.; KIM, K. H.; SHON, J. G. A test data generation for performance testing in massive data processing systems. In: *Advanced Multimedia and Ubiquitous Engineering*. [S.l.]: Springer, 2018. p. 207–213.

KIRAN, M.; MURPHY, P.; MONGA, I.; DUGAN, J.; BAVEJA, S. S. Lambda architecture for cost-effective batch and speed big data processing. In: IEEE. *2015 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2015. p. 2785–2792.

KITCHENHAM, B.; AL-KHILIDAR, H.; BABAR, M. A.; BERRY, M.; COX, K.; KEUNG, J.; KURNIAWATI, F.; STAPLES, M.; ZHANG, H.; ZHU, L. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, Springer, v. 13, n. 1, p. 97–121, 2008.

KOLAJO, T.; DARAMOLA, O.; ADEBIYI, A. Big data stream analysis: a systematic literature review. *Journal of Big Data*, Springer, v. 6, n. 1, p. 1–30, 2019.

KOMORNICZAK, J.; KSIENIEWICZ, P. Data stream generation through real concept's interpolation. In: *30th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*. [S.l.: s.n.], 2022. p. 5–7.

KOMORNICZAK, J.; ZYBLEWSKI, P.; KSIENIEWICZ, P. Statistical drift detection ensemble for batch processing of data streams. *Knowledge-Based Systems*, Elsevier, v. 252, p. 109380, 2022.

KONSTANTINOU, C.; STERGIOPOULOS, G.; PARVANIA, M.; ESTEVES-VERISSIMO, P. Chaos engineering for enhanced resilience of cyber-physical systems. In: IEEE. *2021 Resilience Week (RWS)*. [S.l.], 2021. p. 1–10.

KOUKOUVIS, K.; CUBERO, R. A.; PELLICCIONE, P. A/b testing in e-commerce sales processes. In: SPRINGER. *International Workshop on Software Engineering for Resilient Systems*. [S.l.], 2016. p. 133–148.

KRÄMER, J.; SEEGER, B. *A temporal foundation for continuous queries over data streams*. [S.l.]: Univ., 2004.

KRAWCZYK, B. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, Springer, v. 5, n. 4, p. 221–232, 2016.

KREMPL, G.; ŽLIOBAITE, I.; BRZEZIŃSKI, D.; HÜLLERMEIER, E.; LAST, M.; LEMAIRE, V.; NOACK, T.; SHAKER, A.; SIEVI, S.; SPILIOPOULOU, M. et al. Open challenges for data stream mining research. *ACM SIGKDD explorations newsletter*, ACM, v. 16, n. 1, p. 1–10, 2014.

KRUGER, J.; EVANS, M. If you don't want to be late, enumerate: Unpacking reduces the planning fallacy. *Journal of Experimental Social Psychology*, Elsevier, v. 40, n. 5, p. 586–598, 2004.

KULESOVS, I. ios applications testing. In: *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*. [S.l.: s.n.], 2015. v. 3, p. 138–150.

KUUTILA, M.; MÄNTYLÄ, M.; FAROOQ, U.; CLAES, M. Time pressure in software engineering: A systematic review. *Information and Software Technology*, Elsevier, v. 121, p. 106257, 2020.

LAFI, M.; ALRAWASHED, T.; HAMMAD, A. M. Automated test cases generation from requirements specification. In: IEEE. *2021 International Conference on Information Technology (ICIT)*. [S.l.], 2021. p. 852–857.

LAHARIYA, M.; BENOIT, D. F.; DEVELDER, C. Synthetic data generator for electric vehicle charging sessions: Modeling and evaluation using real-world data. *Energies*, Mdpi, v. 13, n. 16, p. 4211, 2020.

LANGHI, S.; TOMMASINI, R.; VALLE, E. D. Extending kafka streams for complex event recognition. In: IEEE. *2020 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2020. p. 2190–2197.

LEESATAPORNWONGSA, T.; LUKMAN, J. F.; LU, S.; GUNAWI, H. S. Taxdc: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. [S.l.: s.n.], 2016. p. 517–530.

LEOPARDI, A. *StreamData*. 2017. Disponível em: <https://github.com/whatyouhide/stream_data>.

LEOPARDI, A.; VALIM, J. *StreamData*. 2021. <https://github.com/whatyouhide/stream_data>.

LI, J.; CHEN, Z.; CHENG, L.; LIU, X. Energy data generation with wasserstein deep convolutional generative adversarial networks. *Energy*, Elsevier, v. 257, p. 124694, 2022.

LI, N.; LEI, Y.; KHAN, H. R.; LIU, J.; GUO, Y. Applying combinatorial test data generation to big data applications. In: IEEE. *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.], 2016. p. 637–647.

LI, Y.; DONG, T.; ZHANG, X.; SONG, Y.-d.; YUAN, X. Large-scale software unit testing on the grid. In: *GrC*. [S.l.: s.n.], 2006. p. 596–599.

LIMA, B. Automated scenario-based integration testing of time-constrained distributed systems. In: IEEE. *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. [S.l.], 2019. p. 486–488.

LIMA, B.; FARIA, J. P.; HIERONS, R. Local observability and controllability analysis and enforcement in distributed testing with time constraints. *IEEE Access*, IEEE, v. 8, p. 167172–167191, 2020.

LIMA, B. C.; FARIA, J. Conformance checking in integration testing of time-constrained distributed systems based on uml sequence diagrams. In: *Proceedings of the 12th International Conference on Software Technologies - ICSOFT*. [S.l.: s.n.], 2017. p. 459–466.

LITT, G.; HARDENBERG, P. v.; HENRY, O. Cambria: schema evolution in distributed systems with edit lenses. In: *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*. [S.l.: s.n.], 2021. p. 1–9.

LIU, P.; XU, H.; SILVA, D. D.; WANG, Q.; AHMED, S. T.; HU, L. Fp4s: Fragment-based parallel state recovery for stateful stream applications. In: IEEE. *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. [S.l.], 2020. p. 1102–1111.

LIU, X.; IFTIKHAR, N.; XIE, X. Survey of real-time processing systems for big data. In: ACM. *Proceedings of the 18th International Database Engineering & Applications Symposium*. [S.l.], 2014. p. 356–361.

LLC, M. *Mockaroo*. 2021. <https://www.mockaroo.com/>.

LUZ, W. P.; PINTO, G.; BONIFÁCIO, R. Building a collaborative culture: a grounded theory of well succeeded devops adoption in practice. In: ACM. *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. [S.l.], 2018. p. 6.

LV, J.; YIN, B.-B.; CAI, K.-Y. On the asymptotic behavior of adaptive testing strategy for software reliability assessment. *IEEE transactions on Software Engineering*, IEEE, v. 40, n. 4, p. 396–412, 2014.

LV, Y.; LIU, R.; JIN, P. Water-wheel: Real-time storage with high throughput and scalability for big data streams. In: *Proceedings of the 33rd International Conference on Software Engineering and Knowledge Engineering*. KSI Research Inc., 2021. p. 634–635. Disponível em: <https://doi.org/10.18293\%2Fseke2021-204>.

MADAN, S.; GOSWAMI, P. A privacy preserving scheme for big data publishing in the cloud using k-anonymization and hybridized optimization algorithm. In: IEEE. *2018 international conference on circuits and systems in digital enterprise technology (ICCSDET)*. [S.l.], 2018. p. 1–7.

MAHOOD, Q.; EERD, D. V.; IRVIN, E. Searching for grey literature for systematic reviews: challenges and benefits. *Research synthesis methods*, Wiley Online Library, v. 5, n. 3, p. 221–234, 2014.

MAILEWA, A.; HERATH, J.; HERATH, S. A survey of effective and efficient software testing. In: *The Midwest Instruction and Computing Symposium.(MICS), Grand Forks, ND*. [S.l.: s.n.], 2015.

MAJEED, A.; LEE, S. Anonymization techniques for privacy preserving data publishing: A comprehensive survey. *IEEE access*, IEEE, v. 9, p. 8512–8545, 2020.

MALASKA, T. *Mastering Spark Unit Testing*. 2019. Disponível em: <https://databricks.com/session/mastering-spark-unit-testing>.

MALENSEK, M.; PALLICKARA, S. L.; PALLICKARA, S. Galileo: A framework for distributed storage of high-throughput data streams. In: IEEE. *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. [S.l.], 2011. p. 17–24.

MANCO, G.; RITACCO, E.; RULLO, A.; SACCÀ, D.; SERRA, E. Machine learning methods for generating high dimensional discrete datasets. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley Online Library, v. 12, n. 2, p. e1450, 2022.

MANNINO, M.; ABOUZIED, A. Is this real? generating synthetic data that looks real. In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. [S.l.: s.n.], 2019. p. 549–561.

MÄNTYLÄ, M. V.; SMOLANDER, K. Gamification of software testing-an mlr. In: SPRINGER. *International conference on product-focused software process improvement*. [S.l.], 2016. p. 611–614.

MARAK. *Fake.js*. 2021. <https://github.com/marak/Faker.js/>.

MCKEEMAN, W. M. Differential testing for software. *Digital Technical Journal*, v. 10, n. 1, p. 100–107, 1998.

MEEHAN, J.; ASLANTAS, C.; ZDONIK, S.; TATBUL, N.; DU, J. Data ingestion for the connected world. In: *CIDR*. [S.l.: s.n.], 2017.

MEHMOOD, E.; ANEES, T. Challenges and solutions for processing real-time big data stream: a systematic literature review. *IEEE Access*, IEEE, v. 8, p. 119123–119143, 2020.

MISHRA, L.; VARMA, S. et al. Performance evaluation of real-time stream processing systems for internet of things applications. *Future Generation Computer Systems*, Elsevier, v. 113, p. 207–217, 2020.

Mocked Streams. *Mocked Streams*. 2016. [Accessed: 2021-02-20]. Disponível em: <https://github.com/jpzk/mockedstreams>.

MOLLÉRI, J. S.; PETERSEN, K.; MENDES, E. Survey guidelines in software engineering: An annotated review. In: *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*. [S.l.: s.n.], 2016. p. 1–6.

MONTE, B. D.; ZEUCH, S.; RABL, T.; MARKL, V. Rhino: Efficient management of very large distributed state for stream processing engines. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. [S.l.: s.n.], 2020. p. 2471–2486.

MORALES, G. D. F.; BIFET, A.; KHAN, L.; GAMA, J.; FAN, W. Iot big data stream mining. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. [S.l.: s.n.], 2016. p. 2119–2120.

MORGAN, D. L.; KRUEGER, R. A.; KING, J. A. *The focus group kit, Vols. 1–6*. [S.l.]: Sage Publications, Inc, 1998.

MOUTAI, F.-z.; HSAINI, S.; AZZOUZI, S.; CHARAF, M. E. H. Testing distributed cloud: A case study. In: IEEE. *2019 International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*. [S.l.], 2019. p. 1–5.

NAMIOT, D. On big data stream processing. *International Journal of Open Information Technologies*, v. 3, n. 8, 2015.

NARDELLI, M.; CARDELLINI, V.; GRASSI, V.; PRESTI, F. L. Efficient operator placement for distributed data stream processing applications. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 30, n. 8, p. 1753–1767, 2019.

NETO, J. B. de S.; MOREIRA, A. M.; VARGAS-SOLAR, G.; MUSICANTE, M. A. Transmut-spark: Transformation mutation for apache spark. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 32, n. 8, p. e1809, 2022.

NEWTON, I. *Carta a Robert Hooke*. 1675. Disponível em: <https://discover.hsp.org/Record/dc-9792/Description#tabnav>. Acesso em: 7 nov. 2024.

NOKLEBY, M.; RAJA, H.; BAJWA, W. U. Scaling-up distributed processing of data streams for machine learning. *Proceedings of the IEEE*, IEEE, v. 108, n. 11, p. 1984–2012, 2020.

NYUMBA, T. O.; WILSON, K.; DERRICK, C. J.; MUKHERJEE, N. The use of focus group discussion methodology: Insights from two decades of application in conservation. *Methods in Ecology and evolution*, Wiley Online Library, v. 9, n. 1, p. 20–32, 2018.

OLSTHOORN, M.; PANICHELLA, A. Multi-objective test case selection through linkage learning-based crossover. In: SPRINGER. *Search-Based Software Engineering: 13th International Symposium, SSBSE 2021, Bari, Italy, October 11–12, 2021, Proceedings 13*. [S.l.], 2021. p. 87–102.

ORSO, A.; SHI, N.; HARROLD, M. J. Scaling regression testing to large software systems. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 29, n. 6, p. 241–251, 2004.

OSWILL, D. *5 Challenges to Deploying Real-Time Data Streaming Platforms*. 2019. [Accessed: 2021-02-20]. Disponível em: <https://www.computer.org/publications/tech-news/trends/5-challenges-to-deploying-real-time-data-streaming-platforms>.

OTTOGROUP. *Flink Spector*. 2019. <https://github.com/ottogroup/flink-spector>.

PÄÄKKÖNEN, P. Feasibility analysis of asterixdb and spark streaming with cassandra for stream-based processing. *Journal of Big Data*, Springer, v. 3, n. 1, p. 1–25, 2016.

PAGLIARI, A.; HUET, F.; URVOY-KELLER, G. Namb: A quick and flexible stream processing application prototype generator. In: IEEE. *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. [S.l.], 2020. p. 61–70.

PANDEY, R.; SINGH, A.; KASHYAP, A.; ANAND, A. Comparative study on realtime data processing system. In: IEEE. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*. [S.l.], 2019. p. 1–7.

PARGAONKAR, S. Synergizing requirements engineering and quality assurance: A comprehensive exploration in software quality engineering. *International Journal of Science and Research (IJSR)*, v. 12, n. 8, p. 2003–2007, 2023.

PATEL, A. R.; TYAGI, S. The state of test automation in devops: A systematic literature review. In: *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing*. [S.l.: s.n.], 2022. p. 689–695.

PAWLAK, M.; PONISZEWSKA-MARAŃDA, A. Software testing management process for agile approach projects. *Data-Centric Business and Applications: Evolvements in Business Information Processing and Management (Volume 2)*, Springer, p. 63–84, 2020.

Philip Chen, C. L.; ZHANG, C. Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, v. 275, p. 314–347, 2014. ISSN 00200255.

PIZONKA, S.; KEHRER, T.; WEIDLICH, M. Domain model-based data stream validation for internet of things applications. In: *MODELS Workshops*. [S.l.: s.n.], 2018. p. 503–508.

POPIĆ, S.; PAVKOVIĆ, B.; VELIKIĆ, I.; TESLIĆ, N. Data generators: a short survey of techniques and use cases with focus on testing. In: IEEE. *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*. [S.l.], 2019. p. 189–194.

PUNN, N. S.; AGARWAL, S.; SYAFRULLAH, M.; ADIYARTA, K. Testing big data application. In: IEEE. *2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. [S.l.], 2019. p. 159–162.

QIN, C.; EICHELBERGER, H.; SCHMID, K. Enactment of adaptation in data stream processing with latency implications—a systematic literature review. *Information and Software Technology*, Elsevier, v. 111, p. 1–21, 2019.

RAFI, D. M.; MOSES, K. R. K.; PETERSEN, K.; MÄNTYLÄ, M. V. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: IEEE. *2012 7th International Workshop on Automation of Software Test (AST)*. [S.l.], 2012. p. 36–42.

RAHMAN, M. S.; REZA, H. Systematic mapping study of non-functional requirements in big data system. In: IEEE. *2020 IEEE International Conference on Electro Information Technology (EIT)*. [S.l.], 2020. p. 025–031.

RAHUTOMO, R.; LIE, Y.; PERBANGSA, A. S.; PARDAMEAN, B. Improving conversion rates for fashion e-commerce with a/b testing. In: IEEE. *2020 International Conference on Information Management and Technology (ICIMTech)*. [S.l.], 2020. p. 266–270.

RAIZMAN, A.; ANANTHANARAYAN, A.; KIRILOV, A.; CHANDRAMOULI, B.; ALI, M. H. An extensible test framework for the microsoft streaminsight query processor. In: *DBTest*. [S.l.: s.n.], 2010.

REALTIMEBOARD, I. *Miro online whiteboard (no version provided)*. 2023. [Accessed: 2023-11-14]. Disponível em: <www.miro.com>.

RICHMAN, W. L.; KIESLER, S.; WEISBAND, S.; DRASGOW, F. A meta-analytic study of social desirability distortion in computer-administered questionnaires, traditional questionnaires, and interviews. *Journal of applied psychology*, American Psychological Association, v. 84, n. 5, p. 754, 1999.

RIEHLE, D. The open source distributor business model. *Computer*, IEEE, v. 54, n. 12, p. 99–103, 2021.

RIESCO, A.; RODRÍGUEZ-HORTALÁ, J. Property-based testing for spark streaming. *Theory and Practice of Logic Programming*, Cambridge University Press, v. 19, n. 4, p. 574–602, 2019.

RODRÍGUEZ, P.; KUVAJA, P.; OIVO, M. Lessons learned on applying design science for bridging the collaboration gap between industry and academia in empirical software engineering. In: *Proceedings of the 2nd International Workshop on Conducting Empirical Studies in Industry*. [S.l.: s.n.], 2014. p. 9–14.

ROMANO, S.; FUCCI, D.; SCANNIELLO, G.; BALDASSARRE, M. T.; TURHAN, B.; JURISTO, N. On researcher bias in software engineering experiments. *Journal of Systems and Software*, Elsevier, v. 182, p. 111068, 2021.

RUSSO, D. The agile success model: a mixed-methods study of a large-scale agile transformation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM New York, NY, USA, v. 30, n. 4, p. 1–46, 2021.

RUTHERFORD, M. J.; CARZANIGA, A.; WOLF, A. L. Evaluating test suites and adequacy criteria using simulation-based models of distributed systems. *IEEE Transactions on Software Engineering*, IEEE, v. 34, n. 4, p. 452–470, 2008.

SABLIS, A.; SMITE, D.; MOE, N. Team-external coordination in large-scale software development projects. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 33, n. 3, p. e2297, 2021.

SADRI-MOSHKENANI, Z.; BRADLEY, J.; ROTHERMEL, G. Survey on test case generation, selection and prioritization for cyber-physical systems. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 32, n. 1, p. e1794, 2022.

SAHAL, R.; BRESLIN, J. G.; ALI, M. I. Big data and stream processing platforms for industry 4.0 requirements mapping for a predictive maintenance use case. *Journal of manufacturing systems*, Elsevier, v. 54, p. 138–151, 2020.

SALDAÑA, J. *The coding manual for qualitative researchers*. [S.l.]: Sage, 2015.

SALEEM, H.; UDDIN, M. K. S.; REHMAN, S. Habib-ur; SALEEM, S.; ASLAM, A. M. Strategic data driven approach to improve conversion rates and sales performance of e-commerce websites. *International Journal of Scientific & Engineering Research (IJSER)*, 2019.

SAMOSIR, J.; INDRAWAN-SANTIAGO, M.; HAGHIGHI, P. D. An evaluation of data stream processing systems for data driven applications. *Procedia Computer Science*, Elsevier, v. 80, p. 439–449, 2016.

SANTOS, I. de S.; ANDRADE, R. M. de C.; ROCHA, L. S.; MATALONGA, S.; OLIVEIRA, K. M. de; TRAVASSOS, G. H. Test case design for context-aware applications: Are we there yet? *Information and Software Technology*, Elsevier, v. 88, p. 1–16, 2017.

SAYED-MOUCHAWEH, M. Handling concept drift. In: *Learning from Data Streams in Dynamic Environments*. [S.l.]: Springer, 2016. p. 33–59.

SCHIEFERDECKER, I.; HOFFMANN, A. Model-based testing. *IEEE software*, Institute of Electrical and Electronics Engineers, Inc., 345 E. 47 th St. NY . . . , v. 29, n. 1, p. 14–18, 2012.

SCHLEIER-SMITH, J.; KROGEN, E. T.; HELLERSTEIN, J. M. Restream: Accelerating backtesting and stream replay with serial-equivalent parallel processing. In: ACM. *Proceedings of the Seventh ACM Symposium on Cloud Computing*. [S.l.], 2016. p. 334–347.

SEAMAN, C. B. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, IEEE, v. 25, n. 4, p. 557–572, 1999.

SEGURA, S.; BENAVIDES, D.; CORTÉS, A. R. Functional testing of feature model analysis tools. a first step. In: *SPLC (2)*. [S.l.: s.n.], 2008. p. 179.

SHAH, M. A.; HELLERSTEIN, J. M.; BREWER, E. Highly available, fault-tolerant, parallel dataflows. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. [S.l.: s.n.], 2004. p. 827–838.

SHAHRIVAR, S.; ELAHI, S.; HASSANZADEH, A.; MONTAZER, G. A business model for commercial open source software: A systematic literature review. *Information and Software Technology*, Elsevier, v. 103, p. 202–214, 2018.

SHAHVERDI, E.; AWAD, A.; SAKR, S. Big stream processing systems: an experimental evaluation. In: IEEE. *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. [S.l.], 2019. p. 53–60.

SHARMA, A.; SINGH, G.; REHMAN, S. A review of big data challenges and preserving privacy in big data. *Advances in Data and Information Sciences*, Springer, p. 57–65, 2020.

SHIROLE, M.; KUMAR, R. Concurrency coverage criteria for activity diagrams. *IET Software*, Wiley Online Library, v. 15, n. 1, p. 43–54, 2021.

SHIROLE, M.; KUMAR, R. Concurrent behavioral coverage criteria for sequence diagrams. *Innovations in Systems and Software Engineering*, Springer, p. 1–20, 2023.

SHREE, A. R.; KIRAN, P.; MOHITH, N.; KAVYA, M. Sensitivity context aware privacy preserving disease prediction. In: *Expert Clouds and Applications*. [S.l.]: Springer, 2022. p. 11–20.

SILVA, P.; PAIVA, A. C.; RESTIVO, A.; GARCIA, J. E. Automatic test case generation from usage information. In: IEEE. *2018 11Th International Conference on the Quality of Information and Communications Technology (QUATIC)*. [S.l.], 2018. p. 268–271.

SIMONSSON, J.; ZHANG, L.; MORIN, B.; BAUDRY, B.; MONPERRUS, M. Observability and chaos engineering on system calls for containerized applications in docker. *Future Generation Computer Systems*, Elsevier, v. 122, p. 117–129, 2021.

Software Engineering Daily. *Great Expectations: Data Pipeline Testing with Abe Gong*. 2020. [Accessed: 2021-02-20]. Disponível em: <https://podcasts.podinstall.com/software-engineering-daily-software-engineering-daily/202002171000-great-expectations-data-pipeline-testing-abe-gong.html>.

SOUZA, F. R. de; VEITH, A. da S.; ASSUNÇÃO, M. Dias de; CARON, E. Scalable joint optimization of placement and parallelism of data stream processing applications on cloud-edge infrastructure. In: SPRINGER. *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings 18*. [S.l.], 2020. p. 149–164.

SRIKANTH, H.; HETTIARACHCHI, C.; DO, H. Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology*, Elsevier, v. 69, p. 71–83, 2016.

Stack Exchange. *Test Strategies for Stream Processing/Event Processing*. 2017. [Accessed: 2021-02-20]. Disponível em: <https://sqa.stackexchange.com/questions/25915/test-strategies-for-stream-processing-event-processing>.

Stack Overflow. *Test Kafka Streams topology*. 2017. [Accessed: 2021-02-20]. Disponível em: <https://stackoverflow.com/questions/41825753/test-kafka-streams-topology>.

STEINDL, G.; KASTNER, W. Semantic microservice framework for digital twins. *Applied Sciences*, MDPI, v. 11, n. 12, p. 5633, 2021.

STEPHENS, R. A survey of stream processing. *Acta Informatica*, Springer, v. 34, n. 7, p. 491–541, 1997.

STEPIEN, B.; PEYTON, L. Test coordination and dynamic test oracles for testing concurrent systems. In: *SOFTENG 2020 : The Sixth International Conference on Advances and Trends in Software Engineering*. [S.l.: s.n.], 2020. p. 22–27.

STEWART, D. W.; SHAMDASANI, P. Online focus groups. *Journal of Advertising*, Taylor & Francis, v. 46, n. 1, p. 48–60, 2017.

STOL, K.-J.; RALPH, P.; FITZGERALD, B. Grounded theory in software engineering research: a critical review and guidelines. In: IEEE. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. [S.l.], 2016. p. 120–131.

STONEBRAKER, M.; ÇETINTEMEL, U.; ZDONIK, S. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, ACM, 2005.

SUHADA, T. A.; FORD, J. A.; VERREYNNE, M.-L.; INDULSKA, M. Motivating individuals to contribute to firms' non-pecuniary open innovation goals. *Technovation*, Elsevier, v. 102, p. 102233, 2021.

SULEIMAN, D.; ALIAN, M.; HUDAIB, A. A survey on prioritization regression testing test case. In: IEEE. *2017 8th International Conference on Information Technology (ICIT)*. [S.l.], 2017. p. 854–862.

SUN, D.; YAN, H.; GAO, S.; ZHOU, Z. Performance evaluation and analysis of multiple scenarios of big data stream computing on storm platform. *KSII Transactions on Internet and Information Systems (TIIS)*, Korean Society for Internet Information, v. 12, n. 7, p. 2977–2997, 2018.

TALEY, D. S.; PATHAK, B. Comprehensive study of software testing techniques and strategies: a review. *Int. J. Eng. Res*, v. 9, n. 08, p. 817–822, 2020.

TAN, C.; BEHJATI, R.; ARISHOLM, E. A model-based approach to generate dynamic synthetic test data: A conceptual model. In: IEEE. *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.], 2019. p. 11–14.

TANTALAKI, N.; SOURAVLAS, S.; ROUMELIOTIS, M. A review on big data real-time stream processing and its scheduling techniques. *International Journal of Parallel, Emergent and Distributed Systems*, Taylor & Francis, v. 35, n. 5, p. 571–601, 2020.

TANTALAKI, N.; SOURAVLAS, S.; ROUMELIOTIS, M.; KATSAVOUNIS, S. Pipeline-based linear scheduling of big data streams in the cloud. *IEEE Access*, IEEE, v. 8, p. 117182–117202, 2020.

THURMOND, V. A. The point of triangulation. *Journal of nursing scholarship*, Wiley Online Library, v. 33, n. 3, p. 253–258, 2001.

TIWARI, R. G.; SRIVASTAVA, A. P.; BHARDWAJ, G.; KUMAR, V. Exploiting uml diagrams for test case generation: a review. In: IEEE. *2021 2nd international conference on intelligent engineering and management (ICIEM)*. [S.l.], 2021. p. 457–460.

TOM, E.; AURUM, A.; VIDGEN, R. An exploration of technical debt. *Journal of Systems and Software*, Elsevier, v. 86, n. 6, p. 1498–1516, 2013.

TÖNJES, R.; BARNAGHI, P.; ALI, M.; MILEO, A.; HAUSWIRTH, M.; GANZ, F.; GANEA, S.; KJÆRGAARD, B.; KUEMPER, D.; NECHIFOR, S. et al. Real time iot stream processing and large-scale data analytics for smart city applications. In: SN. *poster session, European Conference on Networks and Communications*. [S.l.], 2014.

TORKURA, K. A.; SUKMANA, M. I.; CHENG, F.; MEINEL, C. Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access*, IEEE, v. 8, p. 123044–123060, 2020.

TUCKER, H.; HOCHSTEIN, L.; JONES, N.; BASIRI, A.; ROSENTHAL, C. The business case for chaos engineering. *IEEE Cloud Computing*, IEEE, v. 5, n. 3, p. 45–54, 2018.

TUN, M. T.; NYAUNG, D. E.; PHYU, M. P. Performance evaluation of intrusion detection streaming transactions using apache kafka and spark streaming. In: IEEE. *2019 international conference on advanced information technologies (ICAIT)*. [S.l.], 2019. p. 25–30.

TYNDALL, J. *AACODS checklist for appraising grey literature*. Flinders University, 2010. Accessed: 2021-02-13. Disponível em: <https://dspace.flinders.edu.au/xmlui/bitstream/handle/2328/3326/AACODS_Checklist.pdf>.

VALSAMIS, A.; TSERPES, K.; ZISSIS, D.; ANAGNOSTOPOULOS, D.; VARVARIGOU, T. Employing traditional machine learning algorithms for big data streams analysis: The case of object trajectory prediction. *Journal of Systems and Software*, Elsevier, v. 127, p. 249–257, 2017.

VASA, J.; THAKKAR, A. Deep learning: Differential privacy preservation in the era of big data. *Journal of Computer Information Systems*, Taylor & Francis, p. 1–24, 2022.

VEITH, A. da S.; ASSUNCAO, M. D. de; LEFEVRE, L. Latency-aware strategies for deploying data stream processing applications on large cloud-edge infrastructure. *IEEE transactions on cloud computing*, IEEE, 2021.

VIANELLO, V.; PATIÑO-MARTÍNEZ, M.; AZQUETA-ALZÚAZ, A.; JIMENEZ-PÉRIS, R. Cost of fault-tolerance on data stream processing. In: SPRINGER. *European Conference on Parallel Processing*. [S.l.], 2018. p. 17–27.

VIANNA, A.; FERREIRA, W.; GAMA, K. An exploratory study of how specialists deal with testing in data stream processing applications. In: IEEE. *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.], 2019. p. 1–6.

VIANNA, A.; KAMEI, F.; GAMA, K.; ZIMMERLE, C.; NETO, J. *Research Data*. 2022. <https://doi.org/10.6084/m9.figshare.22259539>. [Accessed: 2023-03-12].

VIANNA, A.; KAMEI, F. K.; GAMA, K.; ZIMMERLE, C.; NETO, J. A. A grey literature review on data stream processing applications testing. *Journal of Systems and Software*, Elsevier, p. 111744, 2023.

VILAS, M. *Google search from Python*. Flinders University, 2020. Accessed: 2021-02-13. Disponível em: <https://python-googlesearch.readthedocs.io/en/latest/>.

VOIGT, P.; BUSSCHE, A. Von dem. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, Springer, 2017.

VRIES, G. K. D. D.; SOMEREN, M. V. Machine learning for vessel trajectories using compression, alignments and domain knowledge. *Expert Systems with Applications*, Elsevier, v. 39, n. 18, p. 13426–13439, 2012.

WADGE, W. W.; ASHCROFT, E. A. *LUCID, the dataflow programming language*. [S.l.]: Academic Press London, 1985. v. 303.

WAEHNER, K. *Testing your Apache Kafka Data with Confidence*. 2022. Accessed: 2022-07-19. Disponível em: <https://www.linkedin.com/posts/kaiwaehner_testing-your-apache-kafka-data-with-confidence-activity-6922507026493284352-C_7N?utm_source=linkedin_share&utm_medium=member_desktop_web>.

WANG, J.; YANG, Y.; WANG, T.; SHERRATT, R. S.; ZHANG, J. Big data service architecture: a survey. *Journal of Internet Technology*, v. 21, n. 2, p. 393–405, 2020.

WANG, X.; ZHANG, C.; FANG, J.; ZHANG, R.; QIAN, W.; ZHOU, A. A comprehensive study on fault tolerance in stream processing systems. *Frontiers of Computer Science*, Springer, v. 16, n. 2, p. 1–18, 2022.

WANG, Z.; MYLES, P.; JAIN, A.; KEIDEL, J. L.; LIDDI, R.; MACKILLOP, L.; VELARDO, C.; TUCKER, A. Evaluating a longitudinal synthetic data generator using real world data. In: IEEE. *2021 IEEE 34th International Symposium on Computer-Based Medical Systems (CBMS)*. [S.l.], 2021. p. 259–264.

WASEEM, M.; LIANG, P.; SHAHIN, M.; SALLE, A. D.; MÁRQUEZ, G. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, Elsevier, v. 182, p. 111061, 2021.

WEYUKER, E. J. On testing non-testable programs. *The Computer Journal*, The British Computer Society, v. 25, n. 4, p. 465–470, 1982.

WHITING, M. A.; HAACK, J. et al. Creating realistic, scenario-based synthetic data for test and evaluation of information analytics software. In: ACM. *Proceedings Workshop on BEyond time and errors*. [S.l.], 2008.

WIESMAN, S. *Testing Stateful Streaming Applications*. 2018. Disponível em: <https://sf-2018.flink-forward.org/index.html\%3Fp=4222.html>.

WINGERATH, W.; WOLLMER, B.; BESTEHORN, M.; SUCCO, S.; FERRLEIN, S.; BüCKLERS, F.; DOMNIK, J.; PANSE, F.; WITT, E.; SENER, A.; GESSERT, F.; RITTER, N. Beaconnect: Continuous web performance a/b testing at scale. *Proc. VLDB Endow.*, VLDB Endowment, v. 15, n. 12, p. 3425–3431, sep 2022. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/3554821.3554833>.

WOHLIN, C. Empirical software engineering research with industry: Top 10 challenges. In: IEEE. *2013 1st international workshop on conducting empirical studies in industry (CESI)*. [S.l.], 2013. p. 43–46.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. [S.l.: s.n.], 2014. p. 1–10.

WU, S.; LIU, M.; IBRAHIM, S.; JIN, H.; GU, L.; CHEN, F.; LIU, Z. Turbostream: Towards low-latency data stream processing. In: IEEE. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. [S.l.], 2018. p. 983–993.

XU, C.; WEDLUND, D.; HELGOSON, M.; RISCH, T. Model-based validation of streaming data:(industry article). In: ACM. *Proceedings of the 7th ACM international conference on Distributed event-based systems*. [S.l.], 2013. p. 107–114.

XU, H.; LIU, P.; CRUZ-DIAZ, S.; SILVA, D. D.; HU, L. Sr3: customizable recovery for stateful stream processing systems. In: *Proceedings of the 21st International Middleware Conference*. [S.l.: s.n.], 2020. p. 251–264.

YAMATO, Y. Automatic verification technology of software patches for user virtual environments on iaas cloud. *Journal of Cloud Computing*, SpringerOpen, v. 4, n. 1, p. 1–14, 2015.

YANG, L.; ZHANG, H.; SHEN, H.; HUANG, X.; ZHOU, X.; RONG, G.; SHAO, D. Quality assessment in systematic literature reviews: A software engineering perspective. *Information and Software Technology*, Elsevier, v. 130, p. 106397, 2021.

YASMIN, J.; TIAN, Y.; YANG, J. A first look at the deprecation of restful apis: An empirical study. In: IEEE. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.], 2020. p. 151–161.

YE, Q.; LU, M. Spot: Testing stream processing programs with symbolic execution and stream synthesizing. *Applied Sciences*, MDPI, v. 11, n. 17, p. 8057, 2021.

YOO, S.; HARMAN, M. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability*, Wiley Online Library, v. 22, n. 2, p. 67–120, 2012.

YU, T.; SRISA-AN, W.; ROTHERMEL, G. An automated framework to support testing for process-level race conditions. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 27, n. 4-5, p. e1634, 2017.

ZEUCH, S.; MONTE, B. D.; KARIMOV, J.; LUTZ, C.; RENZ, M.; TRAUB, J.; BRESS, S.; RABL, T.; MARKL, V. Analyzing efficient stream processing on modern hardware. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 12, n. 5, p. 516–530, 2019.

ZHANG, C.; KUPPANNAGARI, S. R.; KANNAN, R.; PRASANNA, V. K. Generative adversarial network for synthetic time series data generation in smart grids. In: IEEE. *2018 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm)*. [S.l.], 2018. p. 1–6.

ZHAO, X.; GARG, S.; QUEIROZ, C.; BUYYA, R. A taxonomy and survey of stream processing systems. In: *Software Architecture for Big Data and the Cloud*. [S.l.]: Elsevier, 2017. p. 183–206.

ZHOU, J.; LI, S.; ZHANG, Z.; YE, Z. Position paper: Cloud-based performance testing: Issues and challenges. In: *Proceedings of the 2013 international workshop on Hot topics in cloud services*. [S.l.: s.n.], 2013. p. 55–62.

ZHU, H.; HALL, P. A.; MAY, J. H. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, Acm New York, NY, USA, v. 29, n. 4, p. 366–427, 1997.

ZIMMERLE, C.; GAMA, K. A web-based approach using reactive programming for complex event processing in internet of things applications. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2018. (SAC '18), p. 2167–2174. ISBN 9781450351911. Disponível em: <https://doi.org/10.1145/3167132.3167365>.

ZVARA, Z.; SZABÓ, P. G.; BALÁZS, B.; BENCZÚR, A. Optimizing distributed data stream processing by tracing. *Future Generation Computer Systems*, Elsevier, v. 90, p. 578–591, 2019.

ZVARA, Z.; SZABÓ, P. G.; HERMANN, G.; BENCZÚR, A. Tracing distributed data stream processing systems. In: IEEE. *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. [S.l.], 2017. p. 235–242.

APPENDIX A – QUESTIONNAIRES

1) What is your gender? (closed-ended question)

- Male
- Female
- Other

2) Nationality (open-ended)

***3) Country where you currently work?** (open-ended)

***4) What is your age?** (open-ended)

5) Education: What is the highest degree or level of school you have completed? (closed-ended question)

- Less than high school degree
- High school degree or equivalent
- Some college credit, no degree
- Trade/technical/vocational training
- Associate/Bachelor/Graduate degree
- Master degree
- Doctorate degree

- **5.1) If you have an Associate/Bachelor/Graduate degree, please specify the name of the course.**

***6) Occupation: Which of the following category is your current occupation most related to?** (closed-ended question)

- Academy. (Professor / Student / Researcher)
- Industry (Developer/ Software Engineer / Tester Engineer ...)
- Other (specify)

***7) Experience: How many years of IT professional experience do you have?** (closed-ended question)

- at least 1 year
- 2-4 years
- 5-7 years
- 8-10 years
- 11-13 years
- 14-16 years
- 17-19 years
- 20 years or more

***8) Context Experience: How many years of work experience with data stream processing do you have?** (closed-ended question)

- at least 1 year
- 2-4 years
- 5-7 years
- 8-10 years
- 11-13 years

- 14-16 years
- 17-19 years
- 20 years or more

9) OPTIONAL Company's/Institution's Information (open-ended)

10) Could you rate how valuable data stream processing is to your company's business? (Likert scale question)
Likert Scale: [Extremely Valuable, Valuable, Somewhat Valuable, Slightly Valuable, Not valuable at all]

11) Which of the following uses of a data stream is best suited for your institution/-company? (closed-ended question)

- Monitor business data in information systems: user access, user actions, website requests, transactions, interest variables, e-commerce sales conversions and others.
- Monitor network infrastructure: packet monitoring, throughput, improper access, latency, timeout and others.
- Monitor equipment: sensor networks, industrial machines, general hardware and others.
- Other (specify)

***12) Please, check which of the following business contexts your data stream work is related to.** (multiple-choice)

- Industry
- Financial Systems (credit card, financial market, stock exchange)
- E-commerce
- Urban Transportation Systems
- Road Transport Monitoring Systems (Freight)

- Digital marketing
- Research and Development of Software and Technologies for processing data stream.
- Business Intelligence
- Social Networking Monitoring
- Air Traffic Control Systems
- Maritime Traffic Control Systems
- Monitoring of equipment and infrastructures (oil pumps, oil pipelines, oil rigs, mining equipment, nuclear power plants and others)
- Power Distribution Systems (monitoring of transmission networks, electric stations, power plants and others ...)
- Telecommunications
- Other

***13) Could you provide more details on the use of data stream in your company/institution?** (open-ended)

14) Does your job involve identifying patterns in data stream? Example: Monitoring systems that trigger alerts for specific events or complex event processing? (open-ended)

- No
- Yes. Can you exemplify a case like this?

***15) Check which of the following technologies are being used by you or your company/institution to process data stream:** (multiple-choice)

- Apache Flink
- 16.12 54

- Apache Storm
- Apache Kafka
- Apache Spark
- Apache Samza
- Apache SAMOA (Scalable Advanced
- Massive Online Analysis)
- Apache Beam
- Apache Ignite
- Apache Apex
- Apache NiFi
- MOA (Massive Online Analysis)
- DDS (Data Distribution Service)
- AMQP (Advanced Message Queuing
- Protocol)
- Microsoft StreamInsight
- Esper
- Amazon Kinesis
- Azure Stream
- Confluent Platform
- Other

***16) About testing and validating applications that process data stream, can you explain details about the solution you apply to this test? For example, do you have some testing environment, use a testing framework, simulate streams to test**

the application, or test in the production environment? If you have designed your solution, could you provide us with an overview? (open-ended)

***17) How do you generate test data? (replay of historical data, sampling production data, generators, and others.)** (open-ended)

***18) What are the most common bugs found in applications that process stream? Can you give us an example that you have already experienced?** (open-ended)

***19) Computational resources: processing power, storage, and memory needed to perform data stream tests and simulation.** (Likert scale question)
Likert Scale: [Extremely Important, Important, Somewhat Important, Slightly Important, Not Important at all]

***20) The need for handling a large volume of data.** (Likert scale question)
Likert Scale: [Extremely Important, Important, Somewhat Important, Slightly Important, Not Important at all]

***21)Methodology: lack of techniques, processes, guidelines and good practices to test software based on data streams.** (Likert scale question)
Likert Scale: [Extremely Important, Important, Somewhat Important, Slightly Important, Not Important at all]

***22) Tooling: lack of testing frameworks/tools for data stream software.** (Likert scale question)
Likert Scale: [Extremely Important, Important, Somewhat Important, Slightly Important, Not Important at all]

***23) Do you think it would be useful to have an appropriate tool for automated software testing targeting data stream processing? (YES/NO). Please justify your answer below.** (open-ended)

***24) Window configuration options: time-based windows, elements batches sized**

windows, sliding windows and others. (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

***25) Historical Data Replay / Backtesting** (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

***26) Generation of a random data stream which is based on historical data models** (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

***27) Option to replay data stream in an accelerated way. (Keeping the time sync.)** (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

***28) Easily integrated with other infrastructures and tools, such as database systems and platforms for stream processing.** (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

***29)Simulation of fault tolerance scenarios: guaranteed delivery, hardware failures, network failures and others.** (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

***30) Options to configure infrastructure latency (random or fixed).** (Likert scale question)
Likert Scale: [Extremely Helpful, Helpful, Somewhat Helpful, Slightly Helpful, Not Helpful at all]

31) In addition to the features of the previous questions, could you provide other features that you consider essential in a tool for testing data stream? (open-ended)

***32) Could you cite some hot spots or new challenges in the stream processing realm that could be considered in further research?** (open-ended)

33) OPTIONAL PERSONAL CONTACT Would you mind to provide your name and email address so, in case we need some extra information pertinent to the study, we can reach you? Providing such information is entirely optional, and your anonymity will be assured. (open-ended)

APPENDIX B – INTERVIEW SCRIPT

It is a semi-structured interview, so the points in this script are to address critical issues. The interviewer is free to focus on and explore issues that he deems most relevant.

B.1 SECTION 1: INTRODUCTION (ABOUT 3 MINUTES)

The interview begins with a brief conversation explaining the study, the anonymity of the interviewees and elucidation about the consent term briefly.

Examples:

- First of all, I want to thank you for making some time to help us.
- We are researching data stream processing, and the primary objective is to investigate how people develop and test data stream applications.

B.2 SECTION 2: PROFILE AND BACKGROUND OF THE INTERVIEWEE. (ABOUT 6 MINUTES)

This section aims to gather information about the interviewee's profile: training, IT experiences, data stream experience, professional career and others.

Examples:

- Maybe you could start by telling us about your background?
- Education Level? Do you have a degree? What is the name of your undergraduate degree?
- Wich companies you've worked before?
- How many years of IT professional experience do you have?

B.3 SECTION 3: PROFESSIONAL EXPERIENCE WITH DATA STREAM DEVELOPMENT (ABOUT 10 MINUTES)

This section is about the data stream interviewee experience. If data stream activity is past experience, then we redirect questions to that previous experience.

Examples:

- How many years of work experience with data stream processing do you have?
- In which companies have you worked on data stream applications. What is your current occupation in the company?
- How many IT professionals are there in the company? How many are working with data stream applications?
- What is the business context of your company? (Products, services..l)
- Could you rate how valuable data stream processing is to your company's business?
- Could you provide more details on the use of data stream in your company/institution?
- May you give an example of a use case that involves monitoring data stream? Does your job include identifying patterns in the data stream? Standard: Monitoring systems that trigger alerts for specific events or complex event processing?
- What tools does your company/institution use to process the data stream?

B.4 SECTION 4: DATA STREAM APPLICATIONS TESTING (ABOUT 8 MINUTES)

In this section, we try to understand the data stream applications testing practices.

Examples:

- There is a test culture in your company?
- May you explain how are you testing your stream software?
- What levels of testing do you have in data stream applications? Unit tests? Integration tests? Systems tests?

- Can you explain more details about the solution you apply to this test?
- Do you use any specific testing frameworks, testing tools, or testing environments?
- Are your tests embedded in a continuous integration system?
- What are your strategies for generating test data? (A replay of historical data? Sampling production data? Use some generators?)
- What are the most common bugs found in data stream applications? Can you give us an example that you have already experienced?

B.5 SECTION 5: ABOUT TOOLS TO TEST DATA STREAM APPLICATIONS. (ABOUT 8 MINUTES)

This section focuses on specific questions about data stream applications testing tools features.

- Do you think it would be useful to have an appropriate tool for automated testing data stream software?
- Now I will present some features of a tool to test data stream applications. Could you say if you find it useful?
  - Window configuration options: like time-based windows, elements batches sized windows, sliding windows and others.
  - Historical Data Replay / Backtesting
  - Generation of a random data stream based on historical data models.
  - Option to replay data stream in an accelerated way. (Keeping the time sync.)
  - Easily integrated with other infrastructures and tools, such as database systems and platforms for stream processing.
  - Simulation of fault tolerance scenarios: guaranteed delivery, hardware failures, network failures and others.
  - Options to configure infrastructure latency (random or fixed).

- In addition to the features of the previous questions, could you provide other features that you consider essential in a tool for testing data stream?
- Could you cite some hot spots or new challenges in the stream processing realm that could be considered in further research?

APPENDIX C – FOCUS GROUP DISCUSSION GUIDE

Below is the discussion guide prepared for the guideline evaluation focus group, which is presented and explained in Section 3.4.

1. **Introduction (5 minutes)**

   1.1. Welcome participants and thank them for their participation.

   1.2. Introduce the facilitator and explain the purpose of the focus group.

   1.3. Provide a brief overview of the script and establish ground rules for the discussion, including respecting others' opinions, one person speaking at a time, and confidentiality.

   1.4. Allow participants to ask questions and request clarification.

2. **Guidelines Overview (25 minutes)**

   2.1. Go through each guideline, encouraging participants to give feedback.

3. **Research Question 1: Perceived Effectiveness (10 minutes)**

   3.1. Ask participants for their overall impressions of the guidelines in supporting practitioners to make decisions related to the development of testing processes for DSP applications.

   3.2. Encourage participants to share specific examples of how the guidelines would be helpful or not in their work.

4. **Research Question 1.1: Gaps and Areas for Improvement (5 minutes)**

   4.1. Facilitate a discussion on any gaps or areas for improvement within the guidelines.

   4.2. Ask participants to suggest recommendations to address the identified gaps or areas for improvement.

5. **Research Question 1.2: Strengths and Weaknesses (5 minutes)**

   5.1. Invite participants to discuss the strengths and weaknesses of the guidelines.

   5.2. Encourage participants to provide examples or explanations for the strengths and weaknesses they identify.

6. **Research Question 1.3: Applicability in Industrial Context (5 minutes)**

   6.1. Facilitate a conversation about the applicability of the guidelines within an industrial context.

   6.2. Encourage participants to share their experiences or thoughts on how the guidelines would fit into their work environment or industry practices.

7. **Conclusion and Next Steps (5 minutes)**

   7.1. Summarize the main points of the discussion, including insights on gaps, strengths, weaknesses, and applicability.

   7.2. Thank participants for their contributions and explain the next steps in the evaluation process (e.g., analyzing the feedback and refining the guidelines).

APPENDIX D – QUESTIONNAIRES

Below we present the questions from the guidelines evaluation survey.

D.1 PARTICIPANT PROFILE

In this section, we have provided a few brief questions to help us better understand the profile of our participants. (Only in this section do we have mandatory questions)

***1) What is your gender?** (closed-ended question)

- Male
- Female
- Others (inform)
- I prefer not to inform.

***2) In which country do you currently work?** (open-ended)

***3) Education: What is the highest degree or level of school you have completed?** (closed-ended question)

- Less than high school degree
- High school degree or equivalent
- Some college credit, no degree
- Trade/technical/vocational training
- Associate/Bachelor/Graduate degree
- Master degree
- Doctorate degree

***4) Experience: How many years of IT professional experience do you have?** (closed-ended question)

- at least 1 year
- 2-4 years
- 5-7 years
- 8-10 years
- 11-13 years
- 14-16 years
- 17-19 years
- 20 years or more

***5) Context Experience: How many years of work experience with data processing do you have? (stream processing, micro-batch, high-performance ETL systems and similar.)** (closed-ended question)

- at least 1 year
- 2-4 years
- 5-7 years
- 8-10 years
- 11-13 years
- 14-16 years
- 17-19 years
- 20 years or more

D.2 #G1 COLLECT INFORMATION

6) How relevant do you find #G1 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Relevant at all]

7) Could you share the reasons for your answer regarding #G1 relevance? We welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.3 #G2 ESTABLISH TEST OBJECTIVES

8) How relevant do you find #G2 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Relevant at all]

9) Could you share the reasons for your answer regarding #G2 relevance? We welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.4 #G3 SYNC TEAM SKILLS WITH TESTING STRATEGY

10) How relevant do you find #G3 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Relevant at all]

11) Could you share the reasons for your answer regarding #G3 relevance? We

welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.5 #G4 PLAN TIME ALLOCATION

12) How relevant do you find #G4 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Relevant at all]

13) Could you share the reasons for your answer regarding #G4 relevance? We welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.6 #G5 PLAN FINANCIAL RESOURCE ALLOCATION

14) How relevant do you find #G5 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Relevant at all]

15) Could you share the reasons for your answer regarding #G5 relevance? We welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.7 #G6 DEVELOP A TEST DATA STRATEGY

16) How relevant do you find #G6 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Rel-

evant at all]

17) Could you share the reasons for your answer regarding #G6 relevance? We welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.8 #G7 PARTICULAR ISSUES OF DSP APPLICATIONS TESTING

18) How relevant do you find #G7 in supporting test planning in the context of data stream processing? (Likert scale question)
Likert Scale: [Extremely Relevant, Relevant, Somewhat Relevant, Slightly Relevant, Not Relevant at all]

19) Could you share the reasons for your answer regarding #G7 relevance? We welcome opinions, suggestions, real-world examples, constructive criticism or any noted inaccuracies. (open-ended)

D.9 FINAL CONSIDERATIONS

20) Do you have any additional comments, opinions, criticisms, or feedback to share? (open-ended)

APPENDIX E – LIST OF TESTING TOOLS

Table 32 – Tools List

Tool Name	Author	License	Main Testing Approaches Support	Features and Specific Uses	Focused on data stream context?	Documents Citation
Alooma Plataform	Alooma Inc.	Proprietary	Contract/Schema Test	Data Validation, Data Replay	Yes	1
Ansible	Michael DeHaan and Red Hat Inc.	GNU General Public License v3.0	System Test	Test Infrastructure Automation	No	2
Apache Griffin	Apache Foundation	Apache License 2.0	Contract/Schema Test; Regression Test	Data Validation	No	3
Artillery	Artillery Software Incorporation	Open Source MPLv2 license	Performance Test	Load generator, Testing Utilities	No	1
Avro Random Generator	Confluent Inc.	Apache License 2.0	Unit Test; Integration Test; Performance Test; Contract/Schema Test	Data Generator	Yes	2
Awaitility Library	Johan Haleby and Community	Apache License 2.0	Integration Test	Asynchronous System Test, Time Issues	No	1
AWS Deequ	Amazon Inc.	Apache License 2.0	Contract/Schema Test	Data Validation	No	3
Chaos Monkey	NetFlix	Apache License 2.0	Chaos Test	Chaos Generation	No	1
Confluent CLI	Confluent Inc.	Confluent Community License Agreement Version 1.0	System Test	Test Infrastructure Automation	Yes	1
Ducktape	Confluent Inc.	Apache License 2.0	Uni Test; Integration Test; System Test; Performance Test	Testing Utilities	Yes	1
Ecto Stream Factory	Igor Barchenkov	MIT License	Uni Test; Integration Test; Performance Test; Property-based Test	Data Generator	Yes	1
Embedded Kafka	Community	MIT License	Unit Test; Integration Test	Mock Stream Processing Topologies	Yes	6
Fake.js	Marak	Copyright Marak Squires	Unit Test	Data Generator	No	2

Faker (Python)	Daniele Faraglia	MIT License	Unit Test	Data Generator	No	1
Flink Spector	ottogroup	Apache License 2.0	Unit Tests; Regression Test	Framework to define unit tests for Apache Flink	Yes	6
Flink Testing Utilities	Apache Foundation	Apache License 2.0	Unit Tests; Regression Test	Testing Utilities, Time Issues, Clock Control	Yes	6
Fluent Kafka Streams Tests	Backdata	MIT License	Unit Tests; Regression Test	Unit Testing Utilities for Kafka	Yes	5
Gatling	Gatling Corp.	Apache License 2.0	Performance Test	Load Test	No	1
Grafana	Grafana Labs	AGPL-3.0 License	Performance Test	Log and Monitoring	No	4
Great Expectations	Superconductive	Apache License 2.0	Contract/Schema Test	Data Validation	No	3
Intel Platform Analysis Technology	Intel Corp.	Proprietary	Performance Test	Log and monitoring of hardware and operation systems metrics	No	1
Jackdaw - Test Machine	Funding Circle	BSD-3-Clause License	Integration Tests	Testing Utilities	Yes	1
Jenkins	Jenkins	MIT License	System Test	Test Infrastructure Automation	No	5
Jepsen	Jepsen	Apache License 2.0	Chaos Test	Chaos Generation	No	1
JMeter	Apache Foundation	Apache License 2.0	Performance Test	Load Test	No	7
Kafka Datagen Connector	Confluent Inc.	Apache-2.0 License	System Test; Performance Test	Data Generator	Yes	3
Kafka for Junit	Markus Günther	Apache License 2.0	Unit Test; Regression Test	Unit Testing Utilities, Junit Support fo Kafka	Yes	5
Kafka Streams Testing Utilities	Apache Foundation	Apache License 2.0	Unit Tests; Regression Test	Testing Utilities, Time Issues	Yes	8
Kafkameter	Signal	Apache License 2.0	Performance Test	Load Generator	Yes	2
Kcat	Magnus Edenhill - Apache	Copyright (c) 2014-2021 Magnus Edenhill	Unit Test; Integration Test	Testing Utilities	Yes	4
Kinesis Data Generator	Amazon Inc.	Apache License 2.0	System Test; Performance Test	Data Generator, Load Generator	Yes	4
Ksql-datagen	Confluent Inc.	Confluent Community License Agreement Version 1.0	Performance Test; System Test	Data Generator	Yes	4

MemoryStream - Apache Spark	Apache Foundation	Apache License 2.0	Unit Test; Regression Test	Replay data stream values stored in memory	Yes	4
Mockaroo	Mockaroo LLC	Proprietary	Unit Test; Integration Test; System Test	Data Generator	No	2
Mockedstreams	Jendrik Poloczek	Apache License 2.0	Unit Tests	Mock Stream Processing Topologies	Yes	3
Mockito	Community	MIT License	Unit Test	Mocking framework for unit test	No	5
Nifi Script Tester	Matt Burges	Apache License 2.0	Unit Test	Infrastructure Mock	Yes	2
Passert - Apache Beam	Apache Foundation	Apache License 2.0	Unit Tests; Regression Test	Assertion for data collections	Yes	1
Pepper-Box	Great Software Laboratory	Apache License 2.0	Performance Test	Load Generator	No	5
Sangrenel	Jamie Alquiza	Apache License 2.0	Performance Test	Load Generator	Yes	2
SBT Coverage	Community	Apache License 2.0	Integration Test	Tracking code coverage	No	1
Scalacheck	Typelevel.scala	BSD-3-Clause license	Unit Test; Property-based test	Automated property-based testing	No	2
ScalaTest	Community	Apache License 2.0	Unit Tests; Integration Tests	Testing Utilities	No	16
Spark Testing Base	Holden Karau	Apache License 2.0	Unit Test; Integration Test; Regression Test	Base classes to write tests for Spark, Time Issues	Yes	8
Spring Cloud Stream Test Support	Spring	Apache License 2.0	Unit Tests; Integration Tests; Regression Test	Testing Utilities	Yes	3
StreamData	Andrea Leopardi and José Valim	Apache License 2.0	Property-based Test	Data Generator	Yes	3
Terraform	HashiCorp	MPL-2.0 License	System Test	Test Infrastructure Automation	No	4
Thundra	Thundra	Proprietary	Chaos Test	Chaos Generation	No	2
WireMock	Tom Akehurst and Community	Apache License 2.0	Unit Test; Integration Test	Mock APIs and External Services	No	3
ZeroCode Samurai	Zerocode	Apache License 2.0	Performance Test	Testing Utilities, Automated support to tests	Yes	4

APPENDIX F – GLR SELECTED SOURCES

S1 Alexandre Vicenzi; "Test and document your data pipeline" <https://www.alexandrevicenzi.com/posts/test-and-document-your-data-pipeline> PDF File QA Score: 0.35

S2 Artillery; "Load Testing AWS Kinesis With Artillery" <https://artillery.io/blog/load-testing-aws-kinesis> PDF File QA Score: 0.64

S3 Allan MacInnis, Jared Warren, Amazon; "Test Your Streaming Data Solution with the New Amazon Kinesis Data Generator" <https://aws.amazon.com/blogs/big-data/test-your-streaming-data-solution-wi PDF File QA Score: 0.73

S4 Apache Foundation; "Apache Flink 1.11 Documentation: Testing" <https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/testing.html> PDF File QA Score: 0.86

S5 Amazon; "Testing Your Delivery Stream Using Sample Data" <https://docs.aws.amazon.com/firehose/latest/dev/test-drive-firehose.html> PDF File QA Score: 0.61

S6 Cloudera; "Test and validation" <https://docs.cloudera.com/csa/1.4.0/development/topics/csa-test-validatio html> PDF File QA Score: 0.60

S7 Microsoft Corporation; "Test an Azure Stream Analytics job with sample data" <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-test-query> PDF File QA Score: 0.66

S8 Aeldung; "Introduction to Apache Flink with Java" <https://www.baeldung.com/apache-flink> PDF File QA Score: 0.71

S9 Confluent; "Generate Custom Test Data by Using the ksql-datagen Tool" <https://github.com/confluentinc/kafka-connect-datagen> PDF File QA Score: 0.89

S10 Dell EMC; "Confluent Kafka Performance Characterization" <https://infohub.delltechnologies.com/static/media/48e14554-a272-4cb0-8996-80ac6dd016b8.pdf> PDF File QA Score: 0.56

S11 Javier Ramos, ITNEXT; "Big Data Quality Assurance. Introduction" <https://medium.com/@javier.ramos1/big-data-quality-assurance-635c368a3e28> PDF File QA Score: 0.78

S12 Wei Huang; "Build a Real-time Data Pipeline during the weekend in Go-Part 1" <https://medium.com/@jayhuang75/build-a-real-time-data-pipeline-during-the-weekend-in-go-30f9c63e207a> PDF File QA Score: 0.59

S13 Wei Huang; "Build a Real-time Data Pipeline during the weekend in Go-Part 3" <https://medium.com/@jayhuang75/build-a-real-time-data-pipeline-during-the-weekend-in-go-part-3-3aa944d10caf> PDF File QA Score: 0.58

S14 Michael Gendy; "The Power of Quality Assurance — Designing Robust QA Processes for SQL Data Analysis Pipelines" <https://medium.com/@michaelgendy/the-power-of-quality-assurance-designing-ro PDF File QA Score: 0.42

S15 Ashish Mrig; "Design & Strategies for Building Big Data Pipelines" <https://medium.com/@mrashish/design-strategies-for-building-big-data-pipelines-4c11affd47f3> PDF File QA Score: 0.67

S16 Milan Sahu; "Data Quality testing with AWS Deequ" <https://medium.com/@sahu.milan1988/data-quality-testing-with-aws-deequ-53b6f765de08> PDF File QA Score: 0.38

S17 Vivek N; "Quality Engineering our Data Pipelines" <https://medium.com/@vivekn.19/quality-engineering-our PDF File QA Score: 0.50

S18 Zachary Ennenga, Airbnb Engineering & Data Science; "Scaling a Mature Data Pipeline — Managing Overhead" <https://medium.com/airbnb-engineering/scaling-a-mature-data-pipeline-managing-overhe PDF File QA Score: 0.65

S19 Irina Pashkova, GreenM; "Data Warehouse Testing. It's rare luck when you are given a. . . " <https://medium.com/greenm/data-warehouse-testing-3c0fb955da1d> PDF File QA Score: 0.52

S20 Frank Dekervel, Kapernikov; "Writing a high quality data pipeline for master data with Apache Spark — Part 3" <https://medium.com/kapernikov/writing-a-high-quality-data-pipeline-for-master-data-wit PDF File QA Score: 0.50

S21 Talha Malik, The Startup; "Structuring a Robust Data Pipeline" <https://medium.com/swlh/structuring-a-robust-data-pipeline-24ff67783782> PDF File QA Score: 0.61

S22 Nikita Zhevnitskiy; "Testing Kafka based applications." <https://medium.com/test-kafka-based-applications/https-medium-com-testing-kafka-based-applications-85d8951cec43> PDF File QA Score: 0.62

S23 Pavan Kumar S, TestVagrant; "Key Points To Remember While Testing Data Streams" <https://medium.com/testvagrant/what-is-data-streaming-and-key-points-to-know-before-testing-it-7a16fe861e05> PDF File QA Score: 0.35

S24 Anton Bakalets; "Flink Job Unit Testing. Write a unit test ensuring your Flink. . . " <https://medium.com/@anton.bakalets/flink-job-unit-testing-df4f618d07a6> PDF File QA Score: 0.45

S25 Scott Haines; "Testing Spark Structured Streaming Applications:" <https://medium.com/@newfrontcreative/testing-spark-structured-streaming-applications-like-a-boss-95a1b261cd35> PDF File QA Score: 0.69

S26 Jacobus Herman, Big Data Republic; "AWS Kinesis Data Analytics: a cautionary review" <https://medium.com/bigdatarepublic/kinesis-data-analytics-sql-a-cautionary-review-fb9ddd06e5d9> PDF File QA Score: 0.66

S27 Nikita Gupta, Henrique Ribeiro Rezende, Coding Stories; “How to test Kinesis with LocalStack” <https://medium.com/coding-stories/how-to-test-kinesis-with-localstack-356b55c69e2> PDF File QA Score: 0.51

S28 Alibaba Cloud, DataSeries, Digoal; “Using PostgreSQL for Real-Time IoT Stream Processing Applications” <https://medium.com/dataseries/using-postgresql-for-real-time-iot-stream-processing-applicatio PDF File QA Score: 0.53

S29 Alibaba Tech, HackerNoon.com; “From Code Quality to Integration: Optimizing Alibaba’s Blink Testing Framework” <https://medium.com/hackernoon/from-code-quality-to-integration-optimizing-alibaba PDF File QA Score: 0.66

S30 Brandon Stanley, Slalom Data & Analytics; “Amazon Kinesis Data Streams: Auto-scaling the number of shards” <https://medium.com/slalom-data-analytics/amazon-kinesis-data-streams-auto-scaling-t PDF File QA Score: 0.67

S31 Shash, Andy LoPresto, daggett; “Faster way of developing and Testing new Nifi Processor” <https://stackoverflow.com/questions/44748548/faster-way-of-developing-and-testing-new-nifi-processor> PDF File QA Score: 0.61

S32 zavalit, Diego Reico; “apache flink - Failing to trigger StreamingMultipleProgramsTestBase Test in Scala” <https://stackoverflow.com/questions/49155762/failing-to-trigger-streamingmultipleprogramstestbas PDF File QA Score: 0.39

S33 Salvador Vigo, AbhishekN; “Java - Generate fake" stream data. Kafka - Flink"” <https://stackoverflow.com/questions/51934554/generate-fake-stream-data-kafka-flink> PDF File QA Score: 0.31

S34 user3139545, David Anderson; “Java - Is there a notion of virtual time in Apache Flink tests like there is in Reactor and RxJava” <https://stackoverflow.com/questions/54855358/is-there-a-notion-of-virtual PDF File QA Score: 0.55

S35 Tilak, Oleg Zhurakousky; “Spring Cloud Stream - Integration testing” <https://stackoverflow.com/questions/55739806/spring-cloud-stream-integration-testing> PDF File QA Score: 0.59

S36 jerrypeng, Valdon Mesh (KIC); “Java - Test apache pulsar functions in an embedded standalone environment” <https://stackoverflow.com/questions/56515083/test-apache-pulsar-functions-in-an-embedde PDF File QA Score: 0.34

S37 SunilS, Andy LoPresto; “How Can Apache NiFi Flow Be Tested?” <https://stackoverflow.com/questions/57062240/how-can-apache-nifi-flow-be-tested> PDF File QA Score: 0.56

S38 Pavan, Chesnay Schepler; “Java - Unit-testing flink application with streaming data” <https://stackoverflow.com/questions/40845683/unit-testing-flink-application-with-streaming-data> PDF File QA Score: 0.35

S39 Till Rohrmann, Mike; "Junit - How to stop a flink streaming job from program" <https://stackoverflow.com/questions/44441153/how-to-stop-a-flink-streaming-job-from-program> PDF File QA Score: 0.58

S40 Timo Walther, user8298342; "Scala - mock object for flinks DataStream" <https://stackoverflow.com/questions/45067426/mock-object-for-flinks-datastream> PDF File QA Score: 0.41

S41 Diego Reiriz Cores, Yohei Kishimoto, UberHans; "Java - Why is Apache Flink droping the event from datastream?" <https://stackoverflow.com/questions/49278579/why-is-apache-flink-droping-the-event- PDF File QA Score: 0.47

S42 Piotr Nowojski, William Speirs; "Unit Testing Flink Streams with Multiple Event Streams" <https://stackoverflow.com/questions/50454231/unit-testing-flink-streams-with-multiple-event-streams> PDF File QA Score: 0.53

S43 David Anderson, Richard Deurwaarder; "Apache Flink - End to End testing how to terminate input source" <https://stackoverflow.com/questions/51242886/apache-flink-end-to-end-testing-how-to-terminate- PDF File QA Score: 0.61

S44 Felder, David Anderson; "Testing Flink window" <https://stackoverflow.com/questions/60418113/testing-flink-window> PDF File QA Score: 0.55

S45 Felder, David Anderson; "Testing Flink with embedded Kafka" <https://stackoverflow.com/questions/60476733/testing-flink-with-embedded-kafka> PDF File QA Score: 0.50

S46 YRQ; "Junit - How to test a Datastream with jsonobject in Apache Flink" <https://stackoverflow.com/questions/61866226/how-to-test-a-datastream-with-jsonobject-in-apache-flink> PDF File QA Score: 0.34

S47 user13906258, Yeis Gallegos; "Junit - How to test keyedbroadcastprocessfunction in flink?" <https://stackoverflow.com/questions/62919920/how-to-test-keyedbroadcastprocessfunction-in-flink> PDF File QA Score: 0.41

S48 Dennis Layton; "DataOps: Building Trust in Data through Automated Testing" <https://www.linkedin.com/pulse/dataops-building-trust-data-through-automated-testing-dennis-layton> PDF File QA Score: 0.41

S49 Gopinath Mandala; "Applications are going Serverless. How will QA respond?" <https://www.linkedin.com/pulse/applications-going-serverless-how-qa-respond-gopinath-mandala> PDF File QA Score: 0.28

S50 Arseniy Tashoyan; "Developing Event-Driven Applications to Prevent Accidents" <https://www.linkedin.com/pulse/developing-event-driven-applications-prevent-arseniy-tashoyan> PDF File QA Score: 0.63

S51 Shachar Bar; “A million-to-one shot, Doc, million-to-one” <https://www.linkedin.com/pulse/million-to-one-shot-doc-shachar-bar-berezniski-> PDF File QA Score: 0.37

S52 Vijayendra Yadav, Niels Basjes, Arvid Heise, David Anderson; “Apache Flink User Mailing List archive. - [Flink Unit Tests] Unit test for Flink streaming codes” <http://apache-flink-user-mailing-list-archive.2336050.n4.nabble.com/Flink-Unit-Tests-Unit-test-for-Flink-streaming-codes-td37119.html> PDF File QA Score: 0.56

S53 Marcin Kuthan; “Spark and Spark Streaming Unit Testing - Passionate Developer” <http://mkuthan.github.io/blog/2015/03/01/spark-unit-testing> PDF File QA Score: 0.73

S54 Bartosz Gajda; “Unit Testing Apache Spark Structured Streaming using MemoryStream - Bartosz Gajda -” <https://bartoszgajda.com/2020/04/13/testing-spark-structured-streaming-using-memorystream> PDF File QA Score: 0.43

S55 Raphael Brugier, Ippon Technologies; “Testing strategy for Spark Streaming - Part 2 of 2” <https://blog.ippon.tech/testing-strategy-for-spark-streaming> PDF File QA Score: 0.73

S56 Anuj Saxena, Knoldus; “Spark Streaming: Unit Testing DStreams” <https://blog.knoldus.com/spark-streaming-unit-testing-dstreams> PDF File QA Score: 0.58

S57 Dipin Hora; “Performance testing a low-latency stream processing system” <https://blog.wallaroolabs.com/2018/03/performance-testing-a-low-latency-stream-processing-system> PDF File QA Score: 0.78

S58 Cloudflow; “Testing a Flink Streamlet” <https://cloudflow.io/docs/dev/develop/test-flink-streamlet.html> PDF File QA Score: 0.52

S59 Felipe Fernández, Codurance; “Testing Spark Streaming: Unit testing” <https://codurance.com/2016/08/02/testing-spark-streaming-unit-testing> PDF File QA Score: 0.74

S60 Kartik Khare; “Apache Flink: A Guide for Unit Testing in Apache Flink” <https://flink.apache.org/news/2020/02/07/a-guide-for-unit-testing-in-apache-flink.html> PDF File QA Score: 0.78

S61 Apache Foundation; “GitHub - Flink End to End Tests” <https://github.com/apache/flink/tree/master/flink-end-to-end-tests> PDF File QA Score: 0.54

S62 JAppsConsultants; “GitHub - Hello Kafka Stream Testing: The most simple way to test Kafka based applications or micro-services e.g. Read/Write during HBase/Hadoop or other Data Ingestion Pipe Lines” <https://github.com/authorjapps/hello-kafka-stream-testing> PDF File QA Score: 0.56

S63 Amazon; “GitHub - Amazon Kinesis Data Generator: A UI that simplifies testing with Amazon Kinesis Streams and Firehose. Create and save record templates, and easily send data to Amazon Kinesis.” <https://github.com/awslabs/amazon-kinesis-data-generator> PDF File QA Score: 0.68

S64 Igor Barchenkov; "GitHub - Ecto Stream Factory: Generate test data for regular and property-based tests and seed your database" <https://github.com/ibarchenkov/ecto_stream_factory> PDF File QA Score: 0.38

S65 Jendrik Poloczek; "GitHub - Mockedstreams: Scala DSL for Unit-Testing Processing Topologies in Kafka Streams" <https://github.com/jpzk/mockedstreams> PDF File QA Score: 0.66

S66 Ottogroup; "GitHub - Flink Spector: Framework for Apache Flink unit tests" <https://github.com/ottogroup/flink-spector> PDF File QA Score: 0.70

S67 Google Cloud Platform; "Testing the Flink Operator with Apache Kafka" <https://googlecloudplatform.github.io/flink-on-k8s-operator/docs/kafka_test_guide.html> PDF File QA Score: 0.60

S68 Apache Foundation; "Apache Kafka" <https://kafka.apache.org/documentation/streams/developer-guide/testing.html> PDF File QA Score: 0.65

S69 Filipe Correia, Stephan Ewen, Alexander Kolb; "Unit testing support for flink application?" <http://apache-flink-user-mailing-list-archive.2336050.n4.nabble.com/Unit-testing-support-for-flink-application-td4 html> PDF File QA Score: 0.61

S70 Anton Sitkovets; "Testing in Apache Beam Part 2: Stream" <https://medium.com/@asitkovets/testing-in-apache-beam-part-2-stream-2a9950ba2bc7> PDF File QA Score: 0.59

S71 Eugene Lopatkin; "Apache Spark Unit Testing Part 3" <https://medium.com/@eugene_lopatkin/apache-spark-unit-testing-part-3-streaming-79af91e5d4d1> PDF File QA Score: 0.41

S72 Ahsan Nabi Dar; "Testing Analytics Event Stream. Over the years we have made improvement. . . " <https://medium.com/@hsan_nabi_dar/testing-analytics-event-stream-ba1977b649c6> PDF File QA Score: 0.41

S73 Arvid Heise, Lawrence Benson, Bakdata Data Engineering Blog; "Fluent Kafka Streams Tests. A Java test DSL for Kafka Streams" <https://medium.com/bakdata/fluent-kafka-streams-tests-e641785171ec> PDF File QA Score: 0.82

S74 Lawrence Benson, Arvid Heise, Bakdata Data Engineering Blog; "Schema Registry mock for Kafka Streams Tests" <https://medium.com/bakdata/transparent-schema-registry-for-kafka-streams-6b43a3e7a15 PDF File QA Score: 0.73

S75 ProgrammerSought; "Flink unit test - Programmer Sought" <https://programmersought.com/article/53161299873> PDF File QA Score: 0.49

S76 Landon Robinson, Jack Chapa, Databricks, SpotX; "Headaches and Breakthroughs in Building Continuous Applications" <https://pt.slideshare.net/databricks/headaches-and-breakthroughs-in-building-co PDF File QA Score: 0.56

S77 Seth Wiesman, MediaMath; “Flink Forward San Francisco 2018: Seth Wiesman - Testing Stateful Streaming Applications” <https://pt.slideshare.net/FlinkForward/flink-forward-san-francisco-2018-seth-wiesn PDF File QA Score: 0.48

S78 Lars Albertsson; “Test strategies for data processing pipelines, v2.0” <https://pt.slideshare.net/lallea/test-strategies-for-data-processing-pipelines-v20?next_slideshow=1> PDF File QA Score: 0.59

S79 Johannes Haug; “Pystreamfs” <https://pypi.org/project/pystreamfs> PDF File QA Score: 0.45

S80 Artem Bilan, Spring; “How to test Spring Cloud Stream applications (Part I)” <https://spring.io/blog/2017/10/24/how-to-test-spring-cloud-stream-applications-part-i> PDF File QA Score: 0.85

S81 Noodle, Mutt; “Automated testing - Test Strategies for Stream Processing / Event Processing - Software Quality Assurance & Testing Stack Exchange” <https://sqa.stackexchange.com/questions/25915/test-strategies-for-stream-processing-event-processing> PDF File QA Score: 0.30

S82 Ian Jones, Ran Wasserman, Raj Saxena; “Java - Amazon Kinesis + Integration Tests” <https://stackoverflow.com/questions/30777368/amazon-kinesis-integration-tests> PDF File QA Score: 0.46

S83 Todd McGrath; “Spark Streaming Testing with Scala Example” <https://supergloo.com/spark-streaming/spark-streaming-testing-scala> PDF File QA Score: 0.45

S84 Roman Aladev, Blaze Meter; “Apache Kafka - How to Load Test with JMeter” <https://www.blazemeter.com/blog/apache-kafka-how-to-load-test-with-jmeter> PDF File QA Score: 0.77

S85 Yeva Byzek, Confluent; “Stream Processing Tutorial Part 2: Testing Your Streaming Application” <https://www.confluent.io/blog/stream-processing-part-2-testing-your-streaming-application> PDF File QA Score: 0.92

S86 Raghunandan Gupta; “Spring Kafka Integration: Unit Testing using Embedded Kafka” <https://www.linkedin.com/pulse/spring-kafka-integration-unit-testing-using-embedded-gupta> PDF File QA Score: 0.47

S87 Ram Ghadiyaram; “Test data generation using Spark by using simple Json data descriptor with Columns and DataTypes to load in dwh like Hive.” <https://www.linkedin.com/pulse/test-data-generation-usi PDF File QA Score: 0.46

S88 Vidhu Bhatnagar; “Load Testing Our Event Pipeline 2019” <https://klaviyo.tech/load-testing-our-event-pipel PDF File QA Score: 0.52

S89 Oren Raboy, Totango Engineering; “Testing Spark Data Processing In Production” <https://labs.totango.com/testing-spark-data-processing-in-production-8436e976c43> PDF File QA Score: 0.60

S90 Chip; “Mocking - How can I instantiate a Mock Kafka Topic for junit tests?” <https://stackoverflow.com/questions/33748328/how-can-i-instantiate-a-mock-kafka-topic-for-junit-tests/34009141#34009141> PDF File QA Score: 0.39

S91 Luciano Afranllie, CyanogenMod; “Performance - Datastream generator for Apache Kafka” <https://stackoverflow.com/questions/41550056/datastream-generator-for-apache-kafka> PDF File QA Score: 0.37

S92 imehl, Dmitry Minkovsky; “Testing - Test Kafka Streams topology” <https://stackoverflow.com/questions/41825753/test-kafka-streams-topology> PDF File QA Score: 0.55

S93 Brandon Barker, Eugene Lopatkin, Matt Powers, Shankar Koirala, Vidya Technology and Training; “Scala - How to write unit tests in Spark 2.0+?” <https://stackoverflow.com/questions/43729262/how-to-write-unit-tests-in-spark-2-0> PDF File QA Score: 0.73

S94 Matthias J. Sax, Jaker; “Apache kafka - Testing KafkaStreams applications” <https://stackoverflow.com/questions/48599228/testing-kafkastreams-applications> PDF File QA Score: 0.68

S95 David O, Matthias J. Sax; “Testing window aggregation with Kafka Streams” <https://stackoverflow.com/questions/52643391/testing-window-aggregation-with-kafka-streams> PDF File QA Score: 0.64

S96 Ramon Jansen Gomez, Jordan Moore, Matthias J. Sax, Levani Kokhreidze, Vassilis; “Java - How can do Functional tests for Kafka Streams with Avro (schemaRegistry)?” <https://stackoverflow.com/questions/52737242/how-can-do-functional-tests-for-kafka-streams-with-avro-schemaregistry> PDF File QA Score: 0.62

S97 Val Bonn, Vasyl Sarzhynskyi, Matthias J. Sax; “Java - How to test a Kafka Stream app without duplicating the topology? Use of TopologyTestDriver?” <https://stackoverflow.com/questions/52991065/how-to-test-a-kafka-stream-app-without-duplicating-the-topology-use-of-topology> PDF File QA Score: 0.58

S98 Kambo, Michael G. Noll, Sarwar Bhuiyan; “Java - Unit testing a kafka topology that’s using kstream joins” <https://stackoverflow.com/questions/55063686/unit-testing-a-kafka-topology-thats-using-k PDF File QA Score: 0.61

S99 Jeahyun Kim, perkss; “Can I test kafka-streams suppress logic?” <https://stackoverflow.com/questions/57686563/can-i-test-kafka-streams-suppress-logic> PDF File QA Score: 0.46

S100 Ronan Mahony, techburst; “ETL Testing Best Practices.” <https://techburst.io/etl-testing-best-practices-759 PDF File QA Score: 0.52

S101 Jyoti Dhiman; “How to do load testing of a real-time pipeline?” <https://towardsdatascience.com/load-testing-of-a-real-time-pipeline-d32475163285> PDF File QA Score: 0.43

S102 Holden Karau; “GitHub - Spark Testing Base: Base classes to use when writing tests with Spark” <https://github.com/holdenk/spark-testing-base> PDF File QA Score: 0.74

S103 Raphael Brugier, Ippon Technologies; “Testing strategy for Apache Spark jobs - Part 1 of 2” <https://blog.ippon.tech/testing-strategy-apache-spark-jobs> PDF File QA Score: 0.73

S104 Andrea Leopardi, Jose Valim; “StreamData — StreamData v0.5.0” <https://hexdocs.pm/stream_data/StreamData.html> PDF File QA Score: 0.74

S105 Superconductive; “Welcome to Great Expectations! — great_expectations documentation” <https://docs.greatexpectations.io/docs/why_use_ge> PDF File QA Score: 0.75

S106 Apache Foundation; “Griffin - Streaming Use Cases” <http://griffin.apache.org/docs/usecases.html> PDF File QA Score: 0.67

S107 Eugene Lopatkin; “Apache Spark Unit Testing Part 2 — Spark SQL” <https://medium.com/@eugene_lopatkin/apache-spark-unit-testing-part-2-spark-sql-35c76ed592b0> PDF File QA Score: 0.44

S108 Ran Silberman; “How to Unit Test Kafka – Ran Silberman” <https://ransilberman.com/2013/07/19/how-to-unit-test-kafka> PDF File QA Score: 0.47

S109 Stuart Perks; “The Perks of Computer Science” <https://perkss.github.io/#/DistributedSystems/Streaming> PDF File QA Score: 0.56

S110 Apache Foundation; “TopologyTestDriver (kafka 2.8.0 API)” <https://kafka.apache.org/28/javadoc/org/apache/kafka/streams/TopologyTestDriver.html> PDF File QA Score: 0.46

S111 Marcos Schroh, Konstantin Knauf, David Anderson, Felder; “testing - How to properly test a Flink window function?” <https://stackoverflow.com/questions/56755349/how-to-properly-test-a-flink-window-fu PDF File QA Score: 0.61

S112 Markus Günther; “User Guide to Kafka for JUnit” <https://mguenther.github.io/kafka-junit> PDF File QA Score: 0.58

S113 Konstantin Knauf, Apache Foundation; “GitHub - Flink Testing Pyramid: Example of a tested Apache Flink application.” <https://github.com/knaufk/flink-testing-pyramid> PDF File QA Score: 0.63

S114 Yan Cui, Thundra; “Chaos test your Lambda functions with Thundra” <https://blog.thundra.io/chaos-test-your-lambda-functions-with-thundra> PDF File QA Score: 0.78

S115 Nick Dimiduk, Rich Metzger, Stephan Ewen, Till Rohrmann, Alexander Kolb; “Apache Flink User Mailing List archive. - Published test artifacts for flink streaming” <http://apache-flink-user-mailing-list-archiv 2336050.n4.nabble.com/Published-test-artifacts-for-flink-streaming-td3379.html#a3560> PDF File QA Score: 0.53

S116 Todd McGrath; “Kafka Streams Testing with Scala Part 1” <https://supergloo.com/kafka-streams/kafka-streams-testing-scala-part-1> PDF File QA Score: 0.57

S117 Robin Moffatt; “Quick ‘n Easy Population of Realistic Test Data into Kafka” <https://rmoff.net/2018/05/10/quick-n-easy-population-of-realistic-test-data-into-kafka> PDF File QA Score: 0.40

S118 Yeva Byzek, Confluent; “Easy Ways to Generate Test Data in Kafka” <https://www.confluent.io/blog/easy-ways-generate-test-data-kafka> PDF File QA Score: 0.85

S119 Todd McGrath; “Kafka Test Data Generation Examples” <https://supergloo.com/kafka/kafka-test-data> PDF File QA Score: 0.64

S120 Jukka Karbanen, Confluent; “Easy Kafka Streams Testing with TopologyTestDriver - KIP-470” <https://www.confluent.io/blog/test-kafka-streams-with-topologytestdriver> PDF File QA Score: 0.84

S121 Matthias J. Sax; “KIP-247: Add public test utils for Kafka Streams - Apache Kafka - Apache Software Foundation” <https://cwiki.apache.org/confluence/display/KAFKA/KIP-247%3A+Add+public+test+utils+for+Kafka+Streams> PDF File QA Score: 0.55

S122 Bakdata Data Engineering Blog; “GitHub - Fluent Kafka Streams Tests: Fluent Kafka Streams Test with Java” <https://github.com/bakdata/fluent-kafka-streams-tests> PDF File QA Score: 0.71

S123 Zerocode, JAppsConsultants; “GitHub - Zerocode: A community-developed, free, open source, microservices API automation and load testing framework built using JUnit core runners for Http REST, SOAP, Security, Database, Kafka and much more. Zerocode Open Source enables you to create, change, orchestrate and maintain your automated test cases declaratively with absolute ease.” <https://github.com/authorjapps/zerocode> PDF File QA Score: 0.70

S124 LocalStack; “GitHub - Localstack: A fully functional local AWS cloud stack. Develop and test your cloud & Serverless apps offline!” <https://github.com/localstack/localstack> PDF File QA Score: 0.69

S125 Amazon; “Kinesis Data Generator” <https://awslabs.github.io/amazon-kinesis-data-generator/web/help.html> PDF File QA Score: 0.88

S126 Serkan Özal, Amazon; “How Thundra Decreased Data Processing Pipeline Delay By 3x on Average and 6x on P99” <https://aws.amazon.com/pt/blogs/apn/how-thundra-decreased-data-processing-pipelin PDF File QA Score: 0.77

S127 Zerocode; “Kafka Testing Introduction” <https://knowledge.zerocode.io/knowledge/kafka-testing-introductio PDF File QA Score: 0.74

S128 Markus Günther; “Using Kafka for JUnit with Spring Kafka” <https://mguenther.net/2021/03/using_kafka_for_junit_with_spring_kafka/index.html> PDF File QA Score: 0.49

S129 Markus Günther; “Writing system tests for a Kafka-enabled microservice” <https://mguenther.net/2021/02/writing_system_tests_for_kafka_microservices/index.html> PDF File QA Score: 0.44

S130 Markus Günther; “Writing component tests for Kafka consumers” <https://mguenther.net/2021/02/writing_component_tests_for_kafka_consumers/index.html> PDF File QA Score: 0.44

S131 Markus Günther; “Writing component tests for Kafka producers” <https://mguenther.net/2021/01/writing_component_tests_for_kafka_producers/index.html> PDF File QA Score: 0.45

S132 Andy Mac Giolla Fhionntog, Philipp, James; “Python - Data testing framework for data streaming (deequ vs Great Expectations)” <https://stackoverflow.com/questions/64711388/data-testing-framework-for PDF File QA Score: 0.37

S133 L Johnson, Dmitri T; “Apache kafka - Adding SSL parameters to pepper_box config in jmeter” <https://stackoverflow.com/questions/62783931/adding-ssl-parameters-to-pepper-box-config-in-jmeter> PDF File QA Score: 0.49

S134 Julian Harty; “Better Software Testing Blog - Seeking ways to improve the efficiency and effectiveness of our craft” <http://blog.bettersoftwaretesting.com> PDF File QA Score: 0.86

S135 Sysco Corporation; “GitHub - Kkafka Testing: Test examples of kafka-clients: unit, integration, end-to-end” <https://github.com/sysco-middleware/kafka-testing> PDF File QA Score: 0.68

S136 Scale Out Data; “Unit Testing Kafka Streams with Avro - Scalable Data Processing on the JVM Stack” <https://scaleoutdata.com/unit-testing-kafka-streams-with-avro-schemas> PDF File QA Score: 0.55

S137 JAppsConsultants; “What is Zerocode Testing” <https://github.com/authorjapps/zerocode/wiki/What-is-Zerocode-testing> PDF File QA Score: 0.67

S138 JAppsConsultants; “Kafka Load Testing Getting Started” <https://knowledge.zerocode.io/knowledge/kafka-load-testing-getting-started> PDF File QA Score: 0.43

S139 Satish Bhor; “Pepper-Box Kafka Load Generator” <https://dzone.com/articles/pepper-box-kafka-load-genera PDF File QA Score: 0.29

S140 Jay Kreps, Atlassian Corporation; “Performance testing - Apache Kafka - Apache Software Foundation” <https://cwiki.apache.org/confluence/display/KAFKA/Performance+testing> PDF File QA Score: 0.68

S141 Jamie Alquiza; “Load Testing Apache Kafka on AWS” <https://grey-boundary.io/load-testing-apache-kafka-o PDF File QA Score: 0.58

S142 Rami Amar, Alooma; “ETL Testing: The Future is Here” <https://www.alooma.com/blog/etl-testing-the-future-is-here> PDF File QA Score: 0.71

S143 Matthew Powers; “spark-fast-tests” <https://github.com/MrPowers/spark-fast-tests> PDF File QA Score: 0.73

S144 Andy Chambers, Confluent; “Testing Event-Driven Systems” <https://www.confluent.io/blog/testing-event-driven-systems> PDF File QA Score: 0.76

S145 Knoldus; “Writing Unit Test for Apache Spark using Memory Streams” <https://blog.knoldus.com/apache-sparks-memory-streams> PDF File QA Score: 0.56

S146 Amazon; “Serverless Streaming Architectures and Best Practices” <https://d1.awsstatic.com/whitepapers/Serverless_Streaming_Architecture_Best_Practices.pdf> PDF File QA Score: 0.75

S147 Francesco Tisiot, Aiven; “Create your own data stream for Kafka with Python and Faker” <https://aiven.io/blog/create-your-own-data-stream-for-kafka-with-python-and-faker> PDF File QA Score: 0.80

S148 Matthias J. Sax, Peyman, Gnos, thinktwice; “How to unit test a Kafka stream application that uses session window” <https://stackoverflow.com/questions/57480927/how-to-unit-test-a-kafka-stream-applicati PDF File QA Score: 0.49

S149 Andrea Leopardi; “StreamData: Property-based testing and data generation” <https://elixir-lang.org/blog/2017/10/31/stream-data-property-based-testing-and-data-generation-for-elixir> PDF File QA Score: 0.84

S150 Apache Foundation; “Apache Griffin” <https://github.com/apache/griffin> PDF File QA Score: 0.60

S151 Anupama Shetty, Neil Marshall; “Testing Spark: Best Practices” <https://docplayer.net/3780717-Testing-spar html> PDF File QA Score: 0.42

S152 Thomas Groh, Apache Foundation; “Testing Unbounded Pipelines in Apache Beam” <https://beam.apache.org/blog/test-stream> PDF File QA Score: 0.59

S153 Tom Seddon, Deliveroo; “Improving Stream Data Quality With Protobuf Schema Validation” <https://deliveroo.engineering/2019/02/05/improving-stream-data-quality-with-protobuf-schema-validation html> PDF File QA Score: 0.86

S154 Confluent, Jay Kreps; “Why Avro for Kafka Data?” <https://www.confluent.io/blog/avro-kafka-data> PDF File QA Score: 0.73

APPENDIX G – PYTHON SCRIPT: GOOGLE SEARCH

```python
'''
This script facilitates a Grey Literature Review (GLR) by automating the search process on Google. It executes predefined queries (search_strings) both in Google's general search and within selected target websites (target_websites),streamlining the collection of relevant literature.
Author: Alexandre Strapação Guedes Vianna / strapacao@gmail.com / alexandrevianna.net
'''

from googlesearch import search
import requests
from bs4 import BeautifulSoup as bs
import csv

def fetch_title(url, timeout=10):
    try:
        response = requests.get(url, timeout=timeout)
        response.raise_for_status()  # Raises error for bad responses (4XX, 5XX)
        soup = bs(response.content, 'lxml')
        return soup.select_one('title').text if soup.select_one('title') else "No title found"
    except requests.exceptions.RequestException:
        return "Error retrieving title"

def perform_search(search_strings, target_websites, writer):
    """Perform searches and write results to CSV."""
    count = 0
    for search_string in search_strings:
       for url in search(search_string, num=num, start=start, stop=stop, pause=pause):
            count += 1
            title = fetch_title(url)
            writer.writerow([count, title, url, "Google"])
            print(count, '\t', title, '\t', url, '\t', "Google")
        for name, site in target_websites:
            for url in search(search_string, num=num, start=start, stop=stop, pause=pause, extra_params={'as_sitesearch': site}):
                count += 1
```

```
                title = fetch_title(url)
                writer.writerow([count, title, url, name])
                print(count, '\t', title, '\t', url, '\t', name)

search_strings = [
    '(challenges OR difficulties) AND ("data stream" OR datastream) AND (test OR testing)',
    '(purposes OR objective OR goal) AND ("data stream" OR datastream) AND (test OR testing)',
    '( "testing approach" OR "testing strategy" OR "unit test" OR "integration test" OR "system test" OR "acceptance test" OR "functional test" OR "load test" OR "performance test") AND ("data stream" OR datastream) AND (test OR testing)',
    '( "testing data" OR "test data") AND ("data stream" OR datastream) AND (test OR testing)',
    '( "testing framework" OR "testing tool" OR "testing library") AND ("data stream" OR datastream) AND (test OR testing)'
]

target_websites = [('LinkedIn', 'https://www.linkedin.com/pulse/'), ('Medium', 'https://medium.com/'), ('StackOverflow', 'https://stackoverflow.com/questions/')]

# Configure the variables below to perform the pagination.
num = 4 # Number of results per page
start = 0 # First result to retrieve.
stop = 3 # Last result to retrieve. Use None to keep searching forever.

# Lapse to wait between HTTP requests, measured in seconds.
# A lapse too long will make the search slow and too short may block your IP.
pause = 5.0

# Open the CSV file for writing
with open('search_results.csv', mode='w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file, delimiter=';')
    writer.writerow(['id', 'title', 'url', 'source_site'])  # Write the header row
    perform_search(search_strings, target_websites, writer)
```

APPENDIX H – PYTHON SCRIPT: CONVERT WEBSITE TO PDF

```
'''
This script opens a CSV file containing a list of URLs and for each URL it generates PDF files of these webpages.
Ensure you have PyQt5 installed in your Python environment.
You can install it using: pip install PyQt5
Author: Alexandre Strapação Guedes Vianna / strapacao@gmail.com / alexandrevianna.net
'''

import csv
import os
import sys
from PyQt5 import QtCore, QtWidgets, QtWebEngineWidgets

# Initialize a QApplication
app = QtWidgets.QApplication(sys.argv)

def html_to_pdf(html, pdf):

    # Create a QWebEnginePage object, which is used to load and render web pages
    page = QtWebEngineWidgets.QWebEnginePage()

    def handle_print_finished(filename, status):
        print("finished", filename, status)
        # Exit the application once printing is done
        QtWidgets.QApplication.quit()

    def print_pdf():
        # Initiates printing of the currently loaded web page to a PDF file.
        page.printToPdf(pdf)

    def handle_load_finished(status):
        if status:
            # If the page loaded, execute any necessary JavaScript before printing
            execute_js()
        else:
            # If the page failed to load, print an error message and exit
```

```
            print("Failed")
            QtWidgets.QApplication.quit()

    def handle_run_js(status):
        print("-")
        if status:
            # If the JavaScript executed successfully, proceed to print the page to PDF
            QtCore.QTimer.singleShot(1000, print_pdf)
        else:
            # If JavaScript execution failed, attempt to execute it again
            QtCore.QTimer.singleShot(1000, execute_js)

    def execute_js():
        # Example JavaScript code that clicks elements with a specific class name
        page.runJavaScript(
            """
            (function () {
                var elements = document.getElementsByClassName('ClassName')
                for (i = 0; i < elements.length; i++) {
                  elements[i].click()
                }
                return true;
            })();
            """,
            handle_run_js,
        )

    # Connect signals to their corresponding callback functions
    page.pdfPrintingFinished.connect(handle_print_finished)
    page.loadFinished.connect(handle_load_finished)
    # Load the URL of the web page to convert
    page.setUrl(QtCore.QUrl(html))
    # Execute the application event loop
    app.exec_()

# Main execution block
if __name__ == "__main__":
    # Open the CSV file containing URLs
```

```
    with open('search_results.csv') as csv_file:
        # Create a CSV reader object to read the file
        csv_reader = csv.reader(csv_file, delimiter=';')
        # Iterate through each row in the CSV file
        for row in csv_reader:
            # Print the ID, title, and URL for each row
            print(f'{row[0]} | {row[1]} | {row[2]}')
            # Convert the webpage to PDF and save it with a specific filename
            html_to_pdf(row[2], "output/"+str(row[0]) + "_" + str(row[1]) + ".pdf")
```

APPENDIX I – GUIDELINES V1 (BEFORE EVALUATION)

Testing Guidelines for Data Stream Processing Applications

Authors: Alexandre Vianna (asgv@cin.ufpe.br) and Kiev Gama (kiev@cin.ufpe.br)

This guide is designed to assist professionals in planning testing of Data Stream Processing (DSP) applications. The insights will guide your decision-making and help shape testing strategies for your context. Discuss, adapt and apply these suggestions to best suit your needs and encourage fruitful discussions within your team. This compact version provides a quick and user-friendly reference to the most crucial points. For detailed information, refer to the full version.

#G1 COLLECT INFORMATION

A. Understand the application business context identifying the key characteristics to guide your testing decisions.

B. Gather parameters needed for test preparation, such as inputs, outputs, response times, and throughput rates needed for testing.

C. Identify potential issues like adverse conditions, fault tolerance scenarios, and testing obstacles like time issues and non-determinism.

D. Document your process with UML diagrams and technical documents highlighting the concurrency and operation states.

#G2 ESTABLISH TEST OBJECTIVES.

A. Evaluate software quality requirements concerning the application's characteristics. Discuss with stakeholders the series of questions to prioritise testing objectives according to quality categories.

B. Comprehend quality from the perspective of the business involved in the application.

C. Engage in process stakeholders—business analysts, developers, testers, clients, and users.

Guiding questions to establish test objectives

FUNCTIONAL SUITABILITY

Q-A Importance of results' correctness?
1) Define metrics for assessing output correctness.
2) Rank features based on their correctness criticality.
3) Focus on system-level tests considering factors like concurrency, asynchrony, and latency.

Q-B Importance of results' accuracy?
1) Establish metrics and acceptable thresholds for result accuracy.
2) Understand that non-determinism in DSP can affect accuracy. Implement measures to manage this.
3) Use monitoring tools like Grafana for performance analysis.

PERFORMANCE EFFICIENCY

Q-C Importance of time requirements?
1) Identify relevant time parameters and set thresholds.
2) Manage application's clock in test environments.
3) Simulate real-world conditions in the test environment.
4) Leverage DSP frameworks' test utilities to address time-related issues.

Q-D Importance of meeting resource usage requirements?
1) Estimate available resources for production.
2) Consider network resources required to run the application.
3) Conduct system and infrastructure-level testing. Employ monitoring tools to assess resource usage.

Q-E Importance of efficient resource usage?
1) Strategise dynamic hardware scaling.
2) Establish diverse scenarios involving hardware resource usage.
3) Optimise application code for efficient resource usage.
4) Use built-in DSP platform features for performance monitoring.

Q-F Importance of meeting maximum capacity limits?
1) Define parameters for the application's operation at maximum capacity.
2) Execute stress tests for maximum capacity scenarios.
3) Monitor application performance and hardware resource usage.

RELIABILITY

Q-G Importance of application's reliability?
1) Establish parameters to measure application reliability.
2) Define acceptable and unacceptable situations or issues.
3) Conduct system-level testing to verify reliability parameters.

Q-H Priority level for application's operational availability?
1) Determine the required application availability rates.
2) Perform fault tolerance tests.
3) Identify causes of application unavailability and propose mitigation strategies.

Q-I Importance of application's resilience to adverse conditions?
1) Define adverse conditions and specify functionalities that would continue or stop.
2) Specify potential performance degradation under adverse conditions.
3) Conduct fault tolerance testing and chaos engineering.

Q-J Priority level for data recovery and system restoration after interruptions?
1) Characterise potential service interruption scenarios.
2) Establish a disaster recovery plan.
3) Perform fault-tolerance tests to verify recovery mechanisms.

MAINTAINABILITY

Q-K Priority for application evolution without quality degradation?
1) Establish application evolution plans.
2) Perform regression testing before deploying new releases.
3) Test message contract integrity thoroughly.

Q-L Priority for minimising maintenance efforts?
1) Automate tests to reduce developers/testers workload.
2) Implement a CI/CD pipeline to catch bugs and errors early.
3) Maintain automatic test cases, focusing on stable components.

Resource Planning

#G3 SYNC TEAM SKILLS WITH TESTING STRATEGY

A. Skills: Align testing activities with the test team's abilities.
B. Training: Provide training and learning opportunities if needed to boost team skills.
C. Workload: Consider workload to decide on the volume and nature of test tasks.

#G4 PLAN TIME ALLOCATION

A. **Reserve time for key tasks** like test case development, environment setup, dataset creation, tests, and results analysis.
B. **Consider activity complexity**, test execution time, and number of test cases to estimate time for each testing activity.
C. **Rank activities** based on test objectives and project characteristics, as in Guideline #G2.
D. **Create the testing schedule** considering activity priority, estimated time, available time, and manageable workload.
E. **Get tests quickly** with automated test case generation, e.g., property-based tests.

F. **Start automating the testing infrastructure gradually** as the application gains stability, prioritising the most time-consuming activities.

#G5 PLAN FINANCIAL RESOURCE ALLOCATION

A. Align Investments: Ensure financial resource allocation matches the testing objectives from Guideline #G2, and prioritize high-impact investments on achieving these objectives.
B. Comprehensive Costing: Consider all potential costs related to the testing process, including infrastructure, personnel, consultancy contracts, service/tool contracts, and maintenance of the testing infrastructure.
C. Reduce Costs: Adopt strategies to cut costs, such as infrastructure automation, optimised use of paid resources, use of open-source tools, and using mock infrastructure/services. Evaluate the balance between cost-reduction and test effectiveness.

ISO/IEC 25012 Data Quality Characteristics

(1) Accuracy: Data accurately represents attribute values within the application context. DSP operations depend on the values precision, corresponding accurately to the concept of the variable.
(2) Credibility: This is about the authenticity of data and its believability within the application's usage context. Credibility becomes more complex in DSP due to factors like temporal data distribution, value frequency, message intervals, and the 4Vs of Big Data.
(3) Currentness: This refers to the data's relevancy based on its age. Data characteristics in DSP can evolve, making them ineffective for testing. Also, application updates might cause incompatibility with new versions.
(4) Compliance: This means data adheres to standards and conventions. DPS applications have multiple entities interacting through various message patterns, so test data must fit the used data structures.
(5) Confidentiality: This involves maintaining privacy and safeguarding sensitive data. DSP applications often deal with confidential data like personal, geolocation, and financial data.

#G6 DEVELOP A TEST DATA STRATEGY

A. Evaluate your test data set with the ISO/IEC 25012 data quality Characteristics.
B. Combine various test data sources and generation methods to increase data diversity and reduce biases.
C. Do not over-rely on historical data, as it may not contemplate never manifested defects, could become outdated, fail to assess new features, and may be incompatible with future application versions.
D. Improve historical data efficiency with semi-synthetic data generation strategies like mutation, machine learning, and manual customization.
E. Maintain vigilance over the data schema and utilize tools such as Avro for version management and compatibility to minimize issues throughout the schema's evolution.
F. Adhere to privacy regulations when managing test data to prevent legal issues. Adopt strategies to maintain confidentiality, like machine learning and shadow mode running when using production data.
G. Use property-based data generation for quick, easy, and resource-efficient test data creation.
H. Utilize high-quality documentation as a reference for synthetic data generation when real data isn't available, and employ natural language processing to quickly extract parameters from the documentation.

Historical Data: is real data extracted from a production application, including relevant metadata like timestamps and network delay.

(1) Test Oracle Generation: Historical data may not provide a reliable test oracle as the data may lack reliable expected outputs. Outputs need to be generated and validated, this relies on documentation detailing data characteristics and output examples.
(2) Coverage Limitations: Despite extensive historical data, it may not cover all potential future bugs due to application complexity and unmanifested bugs that could arise in production.
(3) Outdatedness: Historical data might not test new functionalities effectively and could become incompatible with updated message schemas. As conceptual data characteristics can change over time, their ability to simulate real-world conditions may decrease.

Mirroring Production Data: involves rerouting replicas of the input data stream from the production to the test environment. It aids in detecting critical failures and enables performance comparison across other versions. Privacy and data security can be preserved through mechanisms called 'shadow mode', which conducts automatic verification of parameters and execution results.

Synthetic Data Techniques: This type of data is generated automatically without using real data in the process.

(1) Property-based Data Generation: This method generates data based on the properties of the data stream, creating large volumes efficiently. Tools such as FlinkCheck, ScalaCheck, StreamData, and Ecto Stream Factory support this technique.

(2) Statistical Properties-Based Generation: This approach creates more accurate data by tailoring the statistical distribution of the generated data. It requires a strong understanding of math and statistics, and custom scripts can be made using libraries like Scipy.

(3) AI-Based Generation: This technique uses Natural Language Processing algorithms to extract information from documentation, providing valuable input for generating data that better represent real-world scenarios.

Semi-Synthetic Data: combines synthetic data generation with real-world data and may also involve customization steps.

(1) Mutation: Generates new data by subtly altering real-world data, enhancing diversity while maintaining core characteristics.

(2) Machine Learning: Extracts features from existing data to generate new data. It expands test datasets, creates data variants, and preserves privacy.

(3) Manual Customizations: Refines data to trigger specific conditions unmet by synthetic data. Achieved through iterative processes and manual adjustments, it requires professionals with a thorough understanding of the application business context and its documentation.

#G7 PARTICULAR ISSUES OF DSP APPLICATIONS TESTING

A. Time Issues: Message ordering, delays, and response time requirements are inherent to the business context. The production and test environments may differ in timing characteristics, potentially affecting test results.

B. Non-Deterministic Behavior: DSP applications may deliver different results across multiple executions due to their non-deterministic nature. It complicates result consistency and the establishment of accurate test oracles. Several characteristics, including stateful, window-based, and concurrent operations, make applications inherently non-deterministic.

C. Fault Tolerance: Businesses must keep running critical operations 24/7, can withstand adverse conditions and recover from failures. The main concerns are construction failures, glitches and interruptions of computational resources, networks, and third-party services.

Strategies for specific timing issues

(1) Clock Simulation: Control the system clock in the test environment to emulate production scenario timings. This is useful for testing temporal windows and other time-dependent functions.

(2) Speeding up the Clock: Minimize test duration by accelerating the clock, but balance speed with accuracy to avoid missing potential timing-related bugs.

(3) Adjusting Processing Time Interval: Longer intervals can yield inaccuracies, while shorter intervals provide precise results but increase computational overhead.

(4) Checkpointing Mechanisms: DSP platforms provide features to periodically store snapshots of states and timers of stateful operators, it would help in reproducing specific testing conditions.

(5) Testing Asynchronous Operations: It is particularly tricky to fire asynchronous event, it involve timeouts and internal operators states. Synchronize non-linear executions using appropriate tools to make process wait until pre-set conditions are met.

Non-determinism testing strategies

(1) Test Oracle Construction: Set acceptable result variation thresholds and validate observed variations with statistical methods.

(2) Deterministic Replay: Manage non-determinism variables for better test execution control.

(3) Chaos Engineering: To ensure result consistency, perform repeated tests under non-deterministic conditions, like out-of-order messages and network and data volume oscillation.

(4) Consider Non-Determinism Bugs: Watch for common bugs like race conditions, ordering issues, state inconsistencies, and timeout-associated problems.

Fault tolerance testing strategies

(1) Infrastructure Redundancy: Have backup servers for takeover in case of primary server failure. This requires budget considerations.

(2) Resource Scalability: Automatic adjustment of resources to meet increasing demand to maintain required performance. Elastic scalability requires planning and budget allocation.

(3) Service Redundancy: Prepare backup services to be automatically activated upon third-party service failure.

(4) Operation Downsizing: For inevitable failures, consider service interruption or functionality deactivation.

(5) Version Rollback: In case of instability after a new release, an easy rollback mechanism will save the day.

(6) Operations Rollback: If an operation yields incorrect results due to a bug, reprocess with a backup infrastructure. Legal aspects and business context must be considered.

(7) Contracts Compatibility: Maintain backward compatibility until contract updates propagate. Avro can assist in schema compatibility maintenance.

APPENDIX J – GUIDELINES CHEAT SHEET (BEFORE EVALUATION)

Testing Guidelines for Data Stream Processing Applications

Authors: Alexandre Vianna (asgv@cin.ufpe.br) and Kiev Gama (kiev@cin.ufpe.br)

#G1 COLLECT INFORMATION

A. Understand the application business context.
B. Gather parameters needed for test preparation.
C. Identify potential testing challenges such as adverse conditions, fault tolerance, time issues, and non-determinism.
D. Create test-relevant documentation, utilizing UML diagrams to represent concurrency and operational states.

#G2 ESTABLISH TEST OBJECTIVES.

A. Evaluate software quality requirements and discuss with stakeholders the questions to establish test objectives
B. Comprehend quality from the perspective of the business.
C. Engage in process stakeholders—business analysts, developers, testers, clients, and users.

Guiding questions to establish test objectives

Q-A Importance of results' correctness?
Q-B Importance of results' accuracy?
Q-C Importance of time requirements?
Q-D Importance of meeting resource usage requirements?
Q-E Importance of efficient resource usage?
Q-F Importance of meeting maximum capacity limits?
Q-G Importance of application's reliability?
Q-H Priority level for application's operational availability?
Q-I Importance of application's resilience to adverse conditions?
Q-J Priority level for data recovery and system restoration after interruptions?
Q-K Priority for application evolution without quality degradation?
Q-L Priority for minimising maintenance efforts?

#G3 SYNC TEAM SKILLS WITH TESTING STRATEGY

A. Align testing activities with the test team's abilities.
B. Provide training and learning opportunities.
C. Consider workload when planning tests.

#G4 PLAN TIME ALLOCATION

A. Reserve time for key tasks like test case development, environment setup, dataset creation, tests execution, and results analysis.
B. Consider activity complexity, test execution time, and number of test cases to estimate time for each testing activity.
C. Rank activities based on test objectives.
D. Create the testing schedule considering activity priority, estimated time, available time, and manageable workload.
E. Get tests quickly with automated test case generation, e.g., property-based tests.
F. Automate the testing infrastructure gradually as the application gains stability.

#G5 PLAN FINANCIAL RESOURCE ALLOCATION

A. Ensure budget allocation matches the testing objectives prioritizing.
B. Consider all potential testing process costs, including infrastructure, personnel, consultancy, service/tool, and maintenance.
C. Adopt cost-cutting strategies like infrastructure automation, efficient use of paid resources, open-source tools, and mock services.

#G6 DEVELOP A TEST DATA STRATEGY

A. Evaluate your test data set with the data quality characteristics.
B. Combine various test data sources and data generation methods.
C. Do not over-rely on historical data.
D. Improve historical data efficiency with semi-synthetic data.
E. Manage schema's evolution to minimize incompatibility issues.
F. Adhere to privacy regulations and adopt strategies to maintain confidentiality.
G. Use property-based data generation for quick test data creation.
H. Utilize high-quality documentation as a reference for synthetic data generation.

ISO/IEC 25012 Data Quality Characteristics

(1) Accuracy; **(2)** Credibility; **(3)** Currentness; **(4)** Compliance; **(5)** Confidentiality

#G7 PARTICULAR ISSUES OF DSP APPLICATIONS TESTING

A. Time Issues: Account for message order, delays, and response time requirements, timing discrepancies between production and test environments.

B. Non-Deterministic Behavior: DSP applications may deliver different results across multiple executions.

C. Fault Tolerance: Ensure resilience against adverse conditions, linke glitches, resource/network disruptions, and third-party service failures.

Particular Issues Testing strategies

Time issues: (1) Clock Simulation; (2) Speeding up the Clock; (3) Adjusting Processing Time Interval; (4) Checkpointing Mechanisms; (5) Testing Asynchronous Operations

Fault tolerance: (1) Infrastructure Redundancy; (2) Resource Scalability; (3) Service Redundancy; (4) Operation Downsizing; (5) Version Rollback; (6) Operations Rollback; (7) Contracts Compatibility

Non-determinism: (1) Test Oracle Construction; (2) Deterministic Replay; (3) Chaos Engineering; (4) Consider Non-Determinism Bugs

Test data sources and data generation strategies

Historical: is real data extracted from a production application.

Mirroring: involves rerouting replicas of the input data stream from the production to the test environment.

Synthetic: data is generated automatically without using real data in the process. Strategies: (1) Property-based Data Generation; (2) Statistical Properties-Based Generation; (3) AI-Based Generation.

Semi-Synthetic: combines synthetic data generation with real-world data and may also involve customization steps. Strategies: (1) Mutation; (2) Machine Learning; (3) Manual Customizations.