UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Amanda Maria da Penha Alves Santiago Silva

Rapid review: build verification tests

Recife
2025

Amanda Maria da Penha Alves Santiago Silva

Rapid review: build verification tests

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Breno Alexandro Ferreira de Miranda

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Amanda Maria da Penha Alves Santiago.
Rapid review: build verification tests / Amanda Maria da Penha Alves Santiago Silva. - Recife, 2025.
35 : il.

Orientador(a): Breno Alexandro Ferreira de Miranda
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
Inclui referências.

1. Revisão rápida. 2. Testes de verificação de build. 3. Testes de fumaça. 4. Testes de sanidade. I. Miranda, Breno Alexandro Ferreira de. (Orientação). II. Título.

000 CDD (22.ed.)

Amanda Maria da Penha Alves Santiago Silva

Rapid review: build verification tests

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 31/07/2025

BANCA EXAMINADORA

_____

Prof. Dr. Breno Miranda (Orientador)

Universidade Federal de Pernambuco

_____

Profa. Dra. Paola Accioly (Examinadora Interna)

Universidade Federal de Pernambuco

Recife

2025

RESUMO

Os testes de verificação de build (BVT), incluindo testes smoke e sanity, são essenciais para garantir a qualidade do software, verificando funcionalidades básicas e evitando falhas críticas que impactam os usuários. No entanto, os desafios e oportunidades nessa área ainda são pouco explorados na literatura. Esse estudo visa realizar uma revisão rápida da literatura sobre BVT com foco nos desafios, motivações e suporte de ferramentas existentes na área. Foi conduzida uma revisão rápida seguindo o protocolo de Cartaxo, Pinto e Soares [1], utilizando as bases ACM Digital Library, IEEE Xplore e SpringerLink, realizando uma síntese narrativa dos resultados obtidos. Iniciando com um conjunto de 1312 estudos e, após aplicação dos critérios de inclusão/exclusão, foram selecionados um conjunto final de 17 estudos primários. Provendo uma avaliação qualitativa acerca de BVT, na perspectiva dos desafios, motivações e suporte de metodologias, frameworks e ferramentas abordadas nos estudos selecionados. Com base na análise dos estudos primários selecionados, foram identificados cinco principais desafios enfrentados pelos testes de verificação de build (BVT): i) falta de estudos/ferramentas para testes de sanidade; ii) tempo de execução em BVT; iii) sistematização e boas práticas em BVT; iv) dificuldade em testar a interface gráfica do usuário (GUI). Apesar de ser um fator de grande importância para BVT, soluções que otimizam o tempo de execução foram o foco de apenas um estudo. Além disso, não há boas práticas difundidas acerca de BVT, deixando o desenvolvimento a cargo da experiência do profissional. Diante dos desafios identificados, há oportunidades no desenvolvimento de ferramentas e na exploração da área por meio de novas tecnologias, como a utilização de modelos de linguagem natural (LLMs).

Palavras-chave: revisão rápida; testes de verificação de build; testes de fumaça; testes de sanidade.

ABSTRACT

Build verification tests (BVT), including smoke and sanity tests, are essential to ensure software quality, by verifying basic functionalities and avoiding critical failures that impact users. However, the challenges and opportunities in this area are still scarcely explored in the literature. This study aims to conduct a rapid literature review on BVT with a focus on the challenges, motivations, and support from existing tools in the area. A rapid review was conducted following the protocol of Cartaxo, Pinto and Soares [1], using the ACM Digital Library, IEEE Xplore, and SpringerLink databases, performing a narrative synthesis of the results obtained. Starting with a set of 1312 studies and, after applying inclusion/exclusion criteria, a final set of 17 primary studies was selected. Providing a qualitative evaluation regarding BVT, from the perspective of challenges, motivations, and support of methodologies, frameworks, and tools addressed in the selected studies. Based on the analysis of the selected primary studies, five main challenges faced by build verification tests (BVT) were identified: i) lack of studies/tools for sanity tests; ii) execution time in BVT; iii) systematization and good practices in BVT; iv) difficulty in testing the graphical user interface (GUI). Despite being a factor of great importance for BVT, solutions that optimize execution time were the focus of only one study. Moreover, there are no widespread good practices regarding BVT, leaving development up to the professional's experience. In view of the identified challenges, there are opportunities in the development of tools and in the exploration of the area through new technologies, such as the use of large language models (LLMs).

Keywords: rapid review; build verification tests; smoke tests; sanity tests.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Os Testes de regressão têm o papel de verificar se as modificações feitas no software sob teste não inseriram novas falhas - ou reintroduziram falhas previamente conhecidas. Nesse sentido, testes smoke e sanity são subconjuntos dos testes de regressão [2][3]; E, por isso, são classificados como testes de verificação de build (BVT).

Acredita-se que a expressão “Smoke tests” tenha se originado a partir de testes de hardware eletrônico, em que a primeira verificação realizada era conectar o dispositivo à rede elétrica e, caso o hardware apresentasse fumaça, ele era desconectado, não sendo necessário realizar mais nenhum teste. Portanto, um conjunto de testes smoke tem como objetivo realizar verificações nas funcionalidades básicas de um software. Um exemplo lúdico desse tipo de testagem seria o envio de uma mensagem de texto em um smartphone que possui o sistema Android.

Testes smoke podem ser aplicados a cada build diária, sendo fundamentais para encontrar instabilidades ou fatores-chave faltantes ou quebrados no Software under test (SUT). Entretanto, se houver falha, não é necessário realizar qualquer teste adicional [2], pois, como o software não obteve sucesso em realizar tarefas estipuladas nos testes básicos, será um gasto de tempo e recursos promover essa build para testes mais complexos.

O glossário do International Software Testing Qualifications Board (2025), refere-se aos testes sanity como sinônimo dos testes smoke [4]. Apesar da similaridade no quesito de ambos serem aplicados como forma de promover ou não uma nova build, eles possuem diferenças nas circunstâncias em que são aplicados. Testes sanity são verificações feitas em situações em que novas features ou correções de bugs são inseridas na aplicação, ou seja, os testes são pontuais e focados nas mudanças descritas.

No contexto de execução de BVT destacam-se dois fatores principais: eficiência e efetividade. Como é argumentado por Kaner, Bach e Pettichord [2], testes smoke devem ser automatizados, com o intuito de gerar maior eficiência no tempo de execução, para que não acarrete na inviabilização dos demais testes. Isso pode ser estendido para os testes sanity, já que ambos são realizados em verificações básicas do software e como forma de promover uma build para testes mais complexos. Ademais, outro fator relevante é a efetividade, ou seja, os conjuntos de testes de

verificação de build devem ser eficazes em encontrar falhas, para incrementar a qualidade da aplicação.

Motivado pelo viés de eficiência e efetividade que os testes de verificação de build devem possuir, neste trabalho serão reportados os resultados de uma revisão rápida (RR), visando à análise dos desafios e oportunidades no contexto de BVT. Particularmente, serão respondidas três perguntas de pesquisa relacionadas aos i) principais desafios destacados no contexto de BVT; ii) as motivações para seu uso; iii) o suporte de ferramentas, metodologias ou frameworks para realizar BVT; Cada um desses quesitos originou questões de pesquisa para o estudo (detalhados na Seção III). As buscas foram realizadas nas bases de dados do IEEE Xplore, ACM Digital Library e SpringerLink com strings de pesquisa voltadas para testes de verificação de build, e, a partir disso, foi aplicado o protocolo de revisão rápida proposto por Cartaxo, Pinto e Soares [1]. O artigo é dividido da seguinte forma: Uma visão geral acerca de testes de smoke e sanity será apresentada na seção 2. Na seção 3, será discorrido sobre o protocolo adotado para executar a revisão rápida. Na seção 4, serão respondidas as questões de pesquisa definidas. A discussão dos principais desafios e oportunidades identificados em BVT será descrita na seção 5. E, por fim, a conclusão será apresentada na seção 6.

2 FUNDAMENTAÇÃO

Será provida, nesta seção, uma visão geral de como os testes de sanidade e fumaça são performados e as vantagens envolvidas na utilização de BVT.

Os testes de sanidade são descritos como verificações realizadas após menores mudanças no código ou funcionalidade. Esses testes são performados para assegurar que bugs foram corrigidos e outras falhas não foram inseridas por consequência das mudanças. O objetivo é determinar se a funcionalidade proposta opera aproximadamente como esperado, e, caso os testes de sanity falhem, a build é rejeitada para que, assim, o gasto em tempo e recursos destinados a testes rigorosos seja preservado. Para que esse tipo de teste seja executado, implica que o software foi sucedido de outras formas de verificações. Portanto, observa-se que o teste de sanidade não é demasiadamente aprofundado quanto outros tipos de testes. Por exemplo, após determinada correção de falha no envio de mensagens em um smartphone, uma verificação de sanity seria o envio de uma mensagem básica a partir desse aplicativo que sofreu a correção.

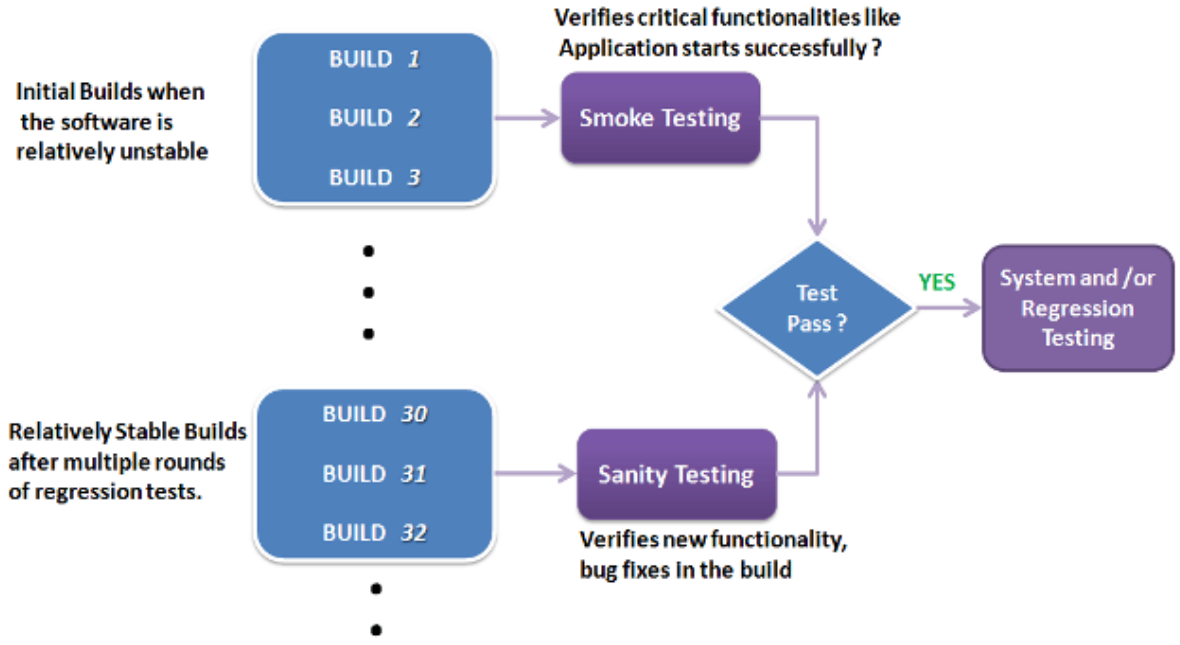
A importância de submeter a aplicação a testes de sanidade está no fato de que nem sempre se faz necessário executar uma análise aprofundada da funcionalidade do software. Nesse sentido, traz, além de praticidade, uma vantagem que está no sentido de que não haverá tendências ou pressuposições. O testador apenas verifica se o software está se comportando como o esperado, sem entrar em detalhamentos de aspectos internos ou desempenho [5].

O processo de testagem smoke consiste em identificar as funcionalidades básicas que a aplicação deve satisfazer; desenvolver casos de testes end-to-end que certifiquem que essas funções funcionam, formando o conjunto de testes smoke; garantir que, cada vez que o software sofra mudanças, a suite seja executada com sucesso antes que outros testes sejam executados; por fim, caso a suite falhe, os desenvolvedores devem trabalhar em correções, e realizar o conjunto de testes até que se obtenha sucesso. É uma prática realizada por algumas organizações o fato de o desenvolvedor executar essa verificação antes de inserir o código no repositório de gerenciamento de configuração, uma vez que podem ser inseridos erros por procedimentos relacionados ao processo de build. Uma pesquisa revelou que até 15% dos defeitos são inseridos por procedimentos relacionados à build [3].

Acerca do processo de testagem sanity, não é possível pré-definir todos os testes que serão executados, já que o fator premissa para execução são novas funcionalidades ou correções de bugs; logo, não há um roteiro padronizado, o foco é mais exploratório e no componente que sofreu mudanças. A figura 1 exemplifica o fluxo de verificação de builds para BVT, no qual os testes smoke são aplicados em builds mais instáveis, enquanto os testes de sanity são executados em versões mais estáveis.

Pode-se inferir que isso ocorre pelo fato de que, nas fases iniciais do desenvolvimento do software, é mais provável que ocorram falhas críticas, ocasionando maior utilidade dos testes smoke para verificar a viabilidade da build. Os testes de sanity, por sua vez, tendem a ser aplicados a builds mais estáveis, já que focam em verificar funcionalidades pontuais adicionadas ou correções de bugs.

Figura 1 – Fluxo de execução de builds em BVT



Fonte: Hamilton [6].

A prática do processo de Daily Build and Smoke test é utilizada por empresas como Microsoft. Essa atividade envolve que cada arquivo seja compilado, vinculado e combinado em um programa executável todo dia, e, em seguida, esse executável é posto em uma bateria de testes smoke. Pode-se explicitar que os benefícios mais significativos dos testes de verificação de build são:

- **Reduz riscos de integração:** A integração ocorre quando diferentes membros do time combinam os códigos que estavam trabalhando separadamente. Problemas ocorrem quando o código resultante não funciona como o esperado. Em cenários que a incompatibilidade é descoberta em estágios tardios no projeto, a depuração pode levar a gastos de tempo maiores; é provável que haja necessidade de realizar mudanças na interface ou reimplementações. Em casos extremos, erros de integração podem acarretar o cancelamento do projeto[7]. Ao utilizar a prática de testagens diárias, evita-se que problemas como esse fujam do controle, tornando-os pequenos e gerenciáveis.
- **Minimiza o risco de baixa qualidade:** Em decorrência de, minimamente, conjuntos de smoke e sanity serem executados em todo código diariamente, problemas de qualidade são prevenidos de infestar o projeto. Ao definir um ”bom estado” que o sistema deve estar e mantê-lo, não permite que problemáticas de qualidade com custo de tempo elevado ocorram.
- **Colabora com rapidez no diagnóstico de defeitos:** O caráter de execução diário do Daily Build facilita o apontamento do porquê de o produto está quebrado em qualquer dia. Por exemplo, se a aplicação funcionava corretamente no dia 10, e, no dia subsequente, apresentou um comportamento incorreto, logo, algo ocorreu entre essas duas builds, ocasionando essa reação [7].

Para consolidar o que foi discutido, o quadro 1 resume as principais diferenças entre testes de smoke e sanity.

Quadro 1 – Comparativo entre smoke e sanity

Smoke	Sanity
Verifica o comportamento das funcionalidades críticas do sistema	Verifica se as novas funcionalidades/bugs foram corrigidas
Executado por desenvolvedores e testadores	Performado por testadores
Testes são documentados e roteirizados	Testes não são documentados ou roteirizados
Testes exercitam o sistema inteiro (end-to-end)	Testes exercitam unicamente componentes particulares do sistema
Avaliação de saúde geral da aplicação	Avaliação de saúde especializada do sistema

Fonte: Hamilton [6].

3 REVISÃO RÁPIDA: PROTOCOLO

O protocolo utilizado para esta revisão rápida fundamenta-se no modelo proposto por Cartaxo et al. [1].

3.1 PROBLEMA DOS PRATICANTES

A crescente complexidade dos softwares torna necessária a aplicação de técnicas que poupam tempo e sejam efetivas na descoberta de falhas, principalmente em projetos que utilizam CI/CD (Integração contínua e entrega contínua). Os testes de verificação de build (BVT) desempenham um papel fundamental ao verificar se as funcionalidades básicas do software operam conforme o esperado. Além disso, esses testes têm como premissa serem executados no menor tempo possível e possuírem efetividade na descoberta de falhas, para permitir a subsequente verificação por testes mais complexos.

3.2 QUESTÕES DE PESQUISA

- QP1: Quais desafios são relatados no contexto de testes de verificação de build?
- QP2: Quais são as motivações reportadas para utilizar testes de verificação de build?
- QP3: Quais são as ferramentas, metodologias ou frameworks utilizados em testes de verificação de build?

3.3 ESTRATÉGIA DE BUSCA

Comumente, utiliza-se apenas uma base de dados para realizar a busca de publicações acadêmicas em revisões rápidas. Porém, a partir de pesquisas preliminares sobre a temática, notou-se que utilizar apenas uma base resultaria em poucas publicações relevantes. Nesse sentido, optou-se por utilizar três repositórios: ACM Digital Library, IEEE Xplore e SpringerLink. A string de busca foi criada a partir da divisão do tema em termos principais e em sinônimos na área de testes de

software. Esse processo visou garantir a maior abrangência possível na busca, resultando na seguinte string de busca:

Quadro 2 – String de busca

(“bvt” OR “build verification” OR “smoke tests” OR “smoke testing” OR “smoke test” OR “sanity tests” OR “sanity testing” OR “sanity tests”) AND (“software test” OR “software testing” OR “software tests” OR “software quality” OR “quality assurance” OR “automated testing” OR “automatic testing” OR “continuous integration” OR “CI”)

Fonte: A autora (2025).

3.4 PROCEDIMENTO DE SELEÇÃO

O processo de seleção foi baseado nos seguintes critérios para assegurar a relevância dos estudos para a temática:

1) O estudo deve estar no contexto de testes de verificação de build.

2) O estudo deve ser primário.

3) O estudo deve fornecer respostas a pelo menos uma das perguntas de pesquisa.

4) O estudo deve apresentar evidências baseadas em métodos empíricos científicos (pesquisas, estudos de caso, etc.). Estudos proposicionais ou anedóticos não foram considerados.

5) O estudo deve estar em inglês.

Figura 2 – Resultado do procedimento de seleção

Fonte: A autora (2025).

A figura 2 apresenta o resultado do processo de seleção. A busca nos três bancos de dados totalizou 1312 resultados. Na primeira etapa de seleção, analisou-se o título das publicações e excluíram-se os que evidentemente não cumpriam os critérios de aceitação, resultando no total de 274 estudos. Apesar dos termos abrangentes utilizados na string de busca resultarem em uma quantidade considerável de falsos positivos, optou-se por mantê-los, para que a pesquisa fosse a mais ampla possível. Na segunda etapa, os resumos das publicações foram examinados, resultando em 30 estudos. E, por fim, na terceira etapa, a leitura completa dos estudos foi realizada, e excluíram-se os que não responderam nenhuma questão de pesquisa ou os que abordavam superficialmente o tema. Assim, 17 estudos primários foram selecionados, conforme mostrado no quadro 3. O procedimento de seleção está disponível online[1] . A seleção em pares não foi considerada devido a restrições de recursos. O processo de seleção foi conduzido em colaboração com um professor e um grupo de alunos que possuem expertise na área de testes, e, a partir de encontros semanais, houve apresentação dos resultados do processo de seleção e discussão de dúvidas.

[1] O procedimento de seleção está disponível em http://bit.ly/4nrQmFs

Quadro 3 – Estudos primários incluídos

ID	Título	Ref.
PS1	Enhancing the Automotive Software Test Environment Using Continuous Integration and Validation Pipeline	[8]
PS2	BSA Tool: An Experience Report of Software Automation to Perform Sanity Tests in a Global Software Development Environment	[9]
PS3	Smoke Testing of Cloud Systems	[10]
PS4	Smoke Testing of UML Activity Diagrams: An Approach for Ensuring System Reliability	[11]
PS5	DART: a framework for regression testing “nightly/daily builds” of GUI applications	[12]
PS6	A Language-guided Acceleration Method for Smoke Testing of Game Quests	[13]
PS7	Web application’s performance testing using HP LoadRunner and CA Wily introscope tools	[14]
PS8	Using Decision Trees to Predict the Certification Result of a Build	[15]
PS9	Repairing Fragile GUI Test Cases Using Word and Layout Embedding	[16]
PS10	Smoke testing for machine learning: simple tests to discover severe bugs	[17]
PS11	Achieving Agility in Projects Through Hierarchical Divisive Clustering Algorithm	[18]
PS12	Automated testing of NFV orchestrators against carrier-grade multi-PoP scenarios using emulation-based smoke testing	[19]
PS13	Unity Application Testing Automation with Appium and Image Recognition	[20]
PS14	Automation of broad sanity test generation	[21]
PS15	Automatic generation of smoke test suites for kubernetes	[22]
PS16	The impact of test ownership and team structure on the reliability and effectiveness of quality test runs	[23]
PS17	AppFlow: using machine learning to synthesize robust, reusable UI tests	[24]

Fonte: A autora (2025).

3.5 PROCEDIMENTO DE EXTRAÇÃO

No processo de extração, foram extraídas todas as informações relevantes que poderiam auxiliar nas respostas para as perguntas de pesquisa. Essa etapa foi conduzida individualmente pelo pesquisador, assim como o procedimento de seleção.

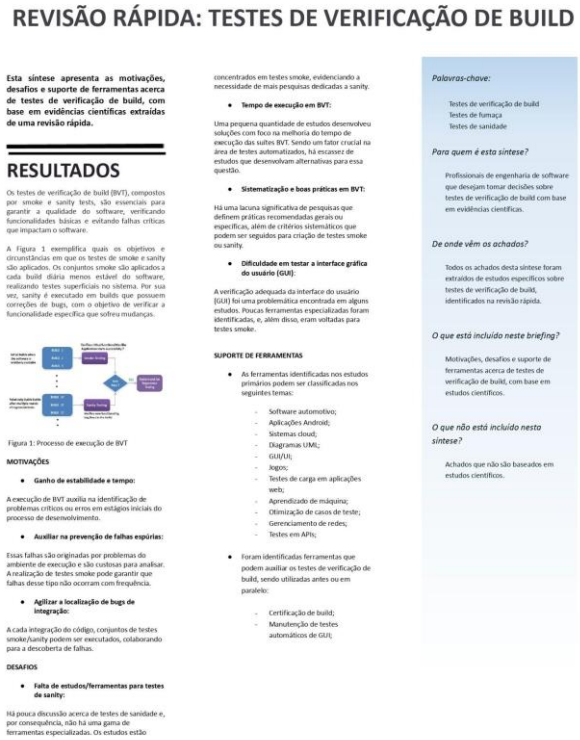
3.6 PROCEDIMENTO DE SÍNTESE

A síntese será conduzida da seguinte forma: a extração das informações será realizada por meio de um formulário padronizado para garantir a integridade dos dados em cada estudo incluído. Após isso, uma síntese narrativa será aplicada para englobar os dados, corroborando para uma melhor e mais abrangente compreensão das evidências, consequentemente facilitando na interpretação dos resultados.

3.7 RELATÓRIO DA REVISÃO RÁPIDA

Com o intuito de ampliar a leitura da revisão rápida (RR), os resultados foram apresentados por meio de um Evidence Briefing. Trata-se de um documento conciso de apenas uma página, com foco nas conclusões obtidas na pesquisa. A figura 3 exibe o documento elaborado para expor a RR. O Evidence Briefing também está disponível online em http://bit.ly/4eaC3Re.

Figura 3 – Evidence Briefing reportando os achados da revisão rápida



REVISÃO RÁPIDA: TESTES DE VERIFICAÇÃO DE BUILD

Esta síntese apresenta as motivações, desafios e suporte de ferramentas acerca de testes de verificação de build, com base em evidências científicas extraídas de uma revisão rápida.

RESULTADOS

Os testes de verificação de build (BVT), compostos por smoke e sanity tests, são essenciais para garantir a qualidade do software, verificando funcionalidades básicas e evitando falhas críticas que impactam o software.

A Figura 1 exemplifica quais os objetivos e circunstâncias em que os testes de smoke e sanity são aplicados. Os conjuntos smoke são aplicados a cada build diária menos estável do software, realizando testes superficiais no sistema. Por sua vez, sanity é executado em builds que possuem correções de bugs, com o objetivo de verificar a funcionalidade específica que sofreu mudanças.

Figura 1: Processo de execução de BVT

MOTIVAÇÕES

- **Ganho de estabilidade e tempo:**

A execução de BVT auxilia na identificação de problemas críticos ou erros em estágios iniciais do processo de desenvolvimento.

- **Auxiliar na prevenção de falhas espúrias:**

Essas falhas são originadas por problemas do ambiente de execução e são custosas para analisar. A realização de testes smoke pode garantir que falhas desse tipo não ocorram com frequência.

- **Agilizar a localização de bugs de integração:**

A cada integração do código, conjuntos de testes smoke/sanity podem ser executados, colaborando para a descoberta de falhas.

DESAFIOS

- **Falta de estudos/ferramentas para testes de sanity:**

Há pouca discussão acerca de testes de sanidade e, por consequência, não há uma gama de ferramentas especializadas. Os estudos estão concentrados em testes smoke, evidenciando a necessidade de mais pesquisas dedicadas a sanity.

- **Tempo de execução em BVT:**

Uma pequena quantidade de estudos desenvolveu soluções com foco na melhoria do tempo de execução das suítes BVT. Sendo um fator crucial na área de testes automatizados, há escassez de estudos que desenvolvam alternativas para essa questão.

- **Sistematização e boas práticas em BVT:**

Há uma lacuna significativa de pesquisas que definem práticas recomendadas gerais ou específicas, além de critérios sistemáticos que podem ser seguidos para criação de testes smoke ou sanity.

- **Dificuldade em testar a interface gráfica do usuário (GUI):**

A verificação adequada da interface do usuário (GUI) foi uma problemática encontrada em alguns estudos. Poucas ferramentas especializadas foram identificadas, e, além disso, eram voltadas para testes smoke.

SUPORTE DE FERRAMENTAS

- As ferramentas identificadas nos estudos primários podem ser classificadas nos seguintes temas:
  - Software automotivo;
  - Aplicações Android;
  - Sistemas cloud;
  - Diagramas UML;
  - GUI/UI;
  - Jogos;
  - Testes de carga em aplicações web;
  - Aprendizado de máquina;
  - Otimização de casos de teste;
  - Gerenciamento de redes;
  - Testes em APIs;
- Foram identificadas ferramentas que podem auxiliar os testes de verificação de build, sendo utilizadas antes ou em paralelo:
  - Certificação de build;
  - Manutenção de testes automáticos de GUI;

Palavras-chave:

Testes de verificação de build
Testes de fumaça
Testes de sanidade

Para quem é esta síntese?

Profissionais de engenharia de software que desejam tomar decisões sobre testes de verificação de build com base em evidências científicas.

De onde vêm os achados?

Todos os achados desta síntese foram extraídos de estudos específicos sobre testes de verificação de build, identificados na revisão rápida.

O que está incluído neste briefing?

Motivações, desafios e suporte de ferramentas acerca de testes de verificação de build, com base em estudos científicos.

O que não está incluído nesta síntese?

Achados que não são baseados em estudos científicos.

Fonte: A autora (2025).

3.8 LIMITAÇÕES E AMEAÇAS À VALIDADE

Devido à metodologia mais simplificada da revisão rápida, há algumas limitações nesta pesquisa. Todo o procedimento de apuração dos estudos foi efetuado por um único pesquisador, o que pode introduzir viés na seleção. Além disso, não houve uma análise de qualidade dos estudos primários incluídos; porém, foram realizadas rodadas de revisão para fortalecer a robustez das conclusões. Embora adotadas medidas de redução de possíveis tendências e formas de melhorar a confiabilidade nos estudos, é necessário considerar essas limitações na interpretação dos resultados.

4 RESULTADOS

Nessa seção, serão apresentadas as respostas para as questões de pesquisa propostas.

4.1 QP1: DESAFIOS

Os desafios relacionados aos testes de verificação de build, identificados nos estudos primários, podem ser agrupados da seguinte forma: i) Falta de sistematização e apoio metodológico em testes smoke; ii) Dificuldades relacionadas a testes smoke em interfaces gráficas do usuário (GUI); iii) Altos custos de tempo em testes smoke/sanity e certificação de build em projetos; iv) Desafios na execução de testes smoke/sanity manuais em produtos escaláveis; v) Dificuldade em testes de sanity manuais em APIs grandes.

O estudo primário [PS2] destacou que testes smoke têm sido, em grande parte, deixados à intuição dos testadores. Ressaltando a importância de definir rigorosamente os objetivos dos testes e realizar experimentos para que haja avanços na área e descoberta de boas práticas que possam ser adotadas pelos profissionais. Nessa perspectiva, o estudo [PS15] identificou a carência de ferramentas especializadas na geração de testes smoke automatizados; consequentemente, são implementados manualmente sem seguir métodos sistemáticos.

Dificuldades na verificação da interface gráfica do usuário (GUI) foram uma problemática abordada nos estudos primários [PS5, PS9]. Os testes smoke executados em builds noturnas sofrem limitações relacionadas à testagem inadequada em softwares que possuem GUI [PS5].

Outro desafio são as Falhas de Identificação de Visão (VIFs), que ocorrem quando mudanças no código-fonte da aplicação modificam identificadores de elementos da interface. Por exemplo, um botão com identificador btnConfirm é alterado para btnOk; por consequência, os testes que utilizam o identificador antigo falham na execução. O estudo primário [PS9] identificou que os scripts de teste smoke para GUI escritos por fornecedores são ainda mais propensos a apresentar VIFs do que os desenvolvidos pelos próprios desenvolvedores do aplicativo, pois os fornecedores são separados das atividades de desenvolvimento e, portanto,

quaisquer mudanças na interface gráfica não são diretamente visíveis ou rastreáveis por eles.

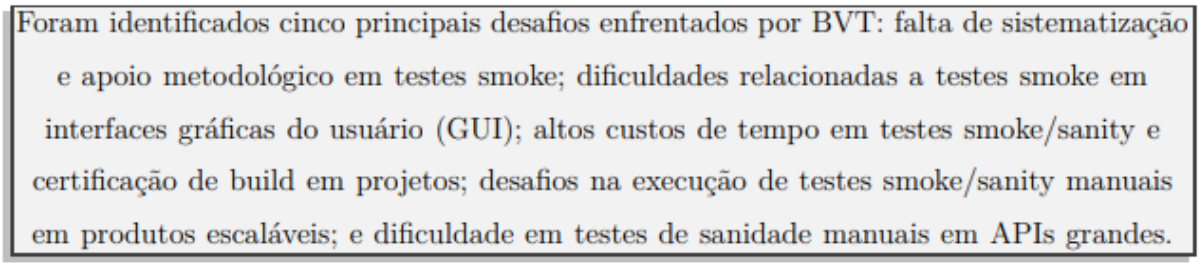
A certificação de build, por meio de testes smoke em projetos de médio porte, pode chegar a consumir oito horas, e esse montante aumenta para projetos de grande porte, sendo um tempo considerável. Enquanto a build está sendo certificada, o status do código mais recente compartilhado no projeto fica indefinido, culminando na redução da capacidade dos membros da equipe de desenvolvimento em paralelo, pois eles precisam aguardar até que uma build certificada esteja liberada para poder atuar [PS8].

Semelhante a essa questão, o estudo [PS2] destacou que a execução dos testes de sanity para cada versão liberada do software, mesmo que seja composta por uma suite pequena, por exemplo, 32 testes, pode gerar gargalo na atividade de testes em casos em que projetos são desenvolvidos em paralelo. Já os testes diários, que também incluem verificações smoke, podem diminuir a rapidez de desenvolvimento de um software, uma vez que verificar softwares grandes e complexos para cada alteração de código pode levar a consumo de tempo e recursos significativos [PS16].

Em relação à execução de testes smoke e sanity, o estudo primário [PS11] destaca que, à medida que o produto se desenvolve e consequentemente os casos de testes dos requisitos relevantes vão aumentando, a tarefa de execução manual desses testes vai se tornando cada vez mais difícil.

Em nível de complexidade, o estudo [PS13] salientou que os testes smoke são usualmente mais complexos quando comparados com testes de unidade regulares e, ao mesmo tempo, não são detalhados o suficiente para prover uma cobertura exaustiva dos possíveis cenários de uso. Por fim, o estudo [PS3] aponta que, em sistemas com um grande número de funções de interface e informações bem estruturadas sobre essas operações, as técnicas convencionais de elaboração de testes de sanity tornam-se excessivamente trabalhosas.

Figura 4 – QP1: desafios


Foram identificados cinco principais desafios enfrentados por BVT: falta de sistematização e apoio metodológico em testes smoke; dificuldades relacionadas a testes smoke em interfaces gráficas do usuário (GUI); altos custos de tempo em testes smoke/sanity e certificação de build em projetos; desafios na execução de testes smoke/sanity manuais em produtos escaláveis; e dificuldade em testes de sanidade manuais em APIs grandes.

Fonte: A autora (2025)

4.2 QP2: MOTIVAÇÕES

A motivação destacada pela maior quantidade de estudos primários foi o ganho de estabilidade e tempo no processo de garantia de qualidade do software [PS1, PS2, PS4, PS5, PS10, PS12, PS13, PS14, PS16, PS17], uma vez que os testes smoke e sanity são testes preliminares com o objetivo de verificar as funcionalidades essenciais da aplicação; eles auxiliam a encontrar problemas críticos ou erros em estágios iniciais do processo de desenvolvimento [PS1, PS4] e provêm um rápido feedback para os desenvolvedores [PS17]. Ademais, são testes que não levam em consideração pequenos detalhes do software, poupando tempo de execução [PS10, PS12].

De acordo com o estudo [PS5] os testes smoke exercitam todo o sistema, porém não de forma exaustiva, sendo uma build que passa por essa verificação considerada uma “boa build”. O estudo primário [PS14] enfatiza que os testes sanity devem ser utilizados antes que o sistema seja posto para executar testes sistemáticos que necessitam de mais esforço, pois seriam ineficazes se o sistema não conseguir lidar com tarefas simples.

Os estudos [PS3, PS15] destacou que a falha de qualquer teste smoke causa a interrupção do processo de testagem, além de prevenir a geração de falhas espúrias que são custosas para analisar. Especificamente, as falhas espúrias que são originadas por meio de problemas de ambiente, caso ocorram com frequência, podem reduzir drasticamente a efetividade do processo de qualidade, aumentando o custo e diminuindo o nível de confiança nas falhas reportadas.

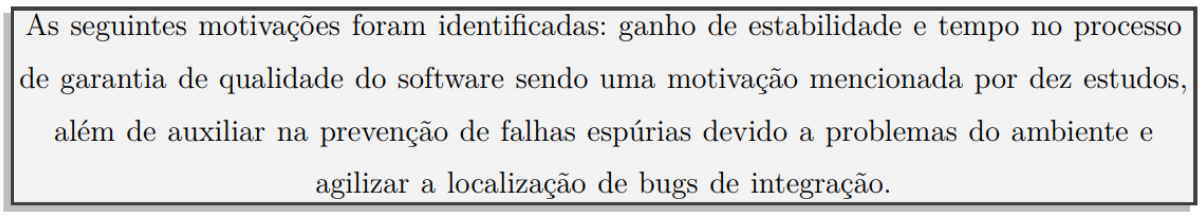
Nesse contexto, os testes smoke são inseridos para performar uma verificação superficial e rápida para garantir que nenhuma falha possa ocorrer devido a

componentes que não estão totalmente operacionais. A título de exemplos, testes smoke podem detectar se um serviço não iniciou corretamente, se o serviço falhou em estabelecer uma conexão com um banco de dados ou volume, e dependências entre serviços que não foram estabelecidas apropriadamente.

Nesse viés de utilizar os testes smoke para verificar falhas de configuração, o estudo [PS7] salientou que, aliado a smoke, os testes sanity também podem ser utilizados com esse propósito, sendo como um teste de performance antes de executar testes de carga no sistema.

Por fim, conjuntos de testes smoke/sanity podem ser utilizados para agilizar o processo de localização de bugs de integração, por meio da implantação dessas verificações no processo de certificação de build, no qual, a cada integração de código, a build seria submetida aos testes em busca de falhas [PS2].

Figura 5 – QP2: motivações



Fonte: A autora (2025)

4.3 QP3: FERRAMENTAS, METODOLOGIAS E FRAMEWORKS

Para responder essa questão de pesquisa, foram coletados as ferramentas reportadas nos estudos primários selecionados, e observou-se que essas ferramentas podem ser designadas para os segmentos de: software automotivo [PS1], aplicações Android [PS2], sistemas cloud [PS3, PS15], Diagramas UML [PS4], GUI/UI [PS5, PS17], Jogos [PS6, PS13], testes de carga em aplicações web [PS7], testes em softwares de aprendizado de máquina [PS10], otimização de casos de teste [PS11], Gerenciamento de redes [PS12], Testes em APIs [PS14] e ferramentas para auxiliar a execução de BVT [PS8, PS9].

Abaixo estão descritas cada uma das ferramentas:

- **Ferramenta Contest [PS1]:** O nicho de aplicação dessa ferramenta são os softwares automotivos, e seu desenvolvimento foi motivado pela necessidade de superar as limitações do Jenkins, que apenas pode executar processos de Integração Contínua/Entrega Contínua (CI/CD) quando o projeto está na ramificação principal (master ou mainline). O intuito principal é automatizar o processo de execução dos testes. O Contest acessa o repositório do projeto e realiza o download dos arquivos que sofreram modificações para o repositório local. Nesse momento, a ferramenta possui a habilidade de comparar a aplicação de origem com os checkpoints do projeto, como forma de indicar o estado esperado do código-fonte no estágio particular de desenvolvimento. Em conjunto com o XCP download, é realizado o flashing dos arquivos binários na Unidade Eletrônica de Controle (ECU). Após isso, o Contest se comunica com a ferramenta CANoe, utilizada na indústria automotiva para simular e testar sistemas de controle eletrônico. Por meio dessa interação, os testes smoke são executados e, a partir dos resultados, são gerados relatórios, possibilitando economia de tempo e recursos necessários para testes manuais em cada release de software. Embora sua criação se originou para o contexto automotivo, há possibilidade de se utilizar em outras áreas desde que sejam realizadas modificações para atender às necessidades do domínio. A ferramenta não está disponível para uso externo, sendo uma solução desenvolvida internamente para uso da empresa.

- **Basic Stand Alone (BSA) [PS2]:** Essa ferramenta realiza testes de sanity automatizados em smartphones Android, executando um conjunto de testes previamente definidos. Ao longo da execução, as evidências são coletadas por meio de capturas de tela para serem analisadas posteriormente pelos testadores. Além disso, a ferramenta suporta execução em múltiplos dispositivos em paralelo. A solução não está disponível para uso externo.

- **Modelo para sistema cloud [PS3]:** Esse estudo define um modelo de testes smoke para sistemas em nuvem e também uma biblioteca de suporte para auxiliar na implementação de casos de teste smoke. Primeiramente, o objetivo do modelo é identificar os principais elementos que podem influenciar a capacidade operacional de um sistema cloud, por exemplo, um dos elementos

é Unidade de Execução (Execution Unit), sendo a parte responsável por hospedar o software em operação (máquinas virtuais, containers, etc.), que, caso não esteja funcionando corretamente, pode prejudicar a integridade do software e causar falhas espúrias. A partir da definição desses componentes, são introduzidos nove critérios de adequação (Cobertura da Unidade de Execução, Cobertura do Serviço, Cobertura do Protocolo do Serviço, Cobertura do Recurso, Tudo Operacional, Cobertura dos Endpoints, Cobertura Total do Serviço, Cobertura da Dependência e Cobertura do Sistema) que exercitam esses elementos e auxiliam na criação de conjuntos de testes smoke. A biblioteca de código aberto em JavaScript fornece uma implementação modificável de funções necessárias para interagir com vários serviços e definir os respectivos oráculos (mecanismos para determinar se um teste passou ou falhou). Uma versão de demonstração da biblioteca está disponível para uso.

- **Framework para Diagramas de Atividade UML [PS4]:** O framework proposto nesse estudo visa mostrar os passos para realizar testes smoke em diagramas de atividade UML como forma de garantir sua corretude, tornando o processo subsequente de implementação mais eficiente e com menos erros. O primeiro passo é a identificação dos caminhos essenciais e de alto risco das funcionalidades do sistema. Em seguida, a criação dos testes deve cobrir os caminhos definidos na etapa anterior. Para que ocorra a execução desses testes manualmente ou por meio de ferramentas de automatização, os resultados da execução devem ser registrados e analisados, para que seja feita uma comparação dos resultados reais com os esperados e, assim, escrita uma documentação dos defeitos. Por último, após a correção dos bugs, os testes smoke devem ser reexecutados para garantir que o sistema está funcionando conforme o esperado.

- **DART [PS5]:** O DART (Daily Automated Regression Tester) é um framework para automatizar o processo de reteste de builds frequentes (noturnos/diários) de aplicações com interfaces gráficas de usuário (GUI). Por meio desse framework é possível automatizar todas as etapas do processo de teste, desde a análise estrutural da GUI até a geração de casos de teste, criação de oráculos, instrumentação de código, execução de testes, avaliação de

cobertura e regeneração de casos de teste. O modelo definido para geração dos testes e oráculos é baseado em objetos, propriedades e ações que alteram o estado da GUI (eventos). A partir dessas informações, um sistema de transição de estados é criado. Esses caminhos variam de tamanho de acordo com a quantidade de ações. Em seguida, o DART identifica as sequências críticas de eventos que são utilizadas para gerar testes smoke. Não há informações no estudo sobre a disponibilidade da ferramenta para uso externo.

- **LAGET [PS6]:** LAGET (Language-Guided Smoke Testing) é um framework que combina planejamento online com um modelo de linguagem natural para encontrar, de forma eficiente, políticas ótimas para cumprir missões (quests) em jogos. O modelo de linguagem natural é introduzido para guiar o agente a explorar o jogo de forma semelhante à humana, recebendo como entrada a descrição da missão e retornando ações recomendadas, evitando significativamente explorações aleatórias e improdutivas. Essas políticas são os caminhos mais curtos para cumprir uma missão, que serão reutilizados em execuções futuras de testes smoke, quando a missão do jogo for alterada.

- **HP LoadRunner [PS7]:** O propósito dessa ferramenta é ser usada para testar o desempenho de carga de uma aplicação. Ela é capaz de simular milhares de usuários utilizando o software da aplicação simultaneamente, armazenando o desempenho da aplicação web para posterior análise de diferentes partes da aplicação e monitoramento dos resultados com a junção da ferramenta CA Wily Introscope, que monitora 24 horas a aplicação. Os testes smoke são realizados com apenas um usuário por script para garantir que a configuração básica da aplicação foi feita corretamente. Não há informações no estudo sobre a disponibilidade da ferramenta para uso externo.

- **ATOML [PS10]:** Sendo uma ferramenta protótipo de geração de testes smoke e testes metamórficos voltados para softwares de aprendizado de máquina. O objetivo do ATOML é gerar dados de teste sintéticos para garantir o funcionamento correto quando os dados forem extremos, por exemplo, verificação de crashs como estouro de memória, divisão por zero, dentre outros. A ferramenta é open-source e está disponível no Github para uso.

- **Algoritmo de Agrupamento Divisivo Hierárquico [PS11]:** A solução proposta tem como objetivo otimizar os casos de teste a serem executados após a disponibilização de uma build, sendo útil especialmente em testes de regressão e de sanidade. Para isso, é calculada a similaridade de Jaccard entre os conjuntos de tokens de todos os casos de teste. Com base nas pontuações de similaridade, os casos de teste são agrupados por meio de um algoritmo de agrupamento divisivo hierárquico, utilizando uma matriz de ligação e um limiar de similaridade. Essa organização permite identificar casos redundantes ou duplicados, que podem ser removidos da suíte de testes. Não há informações no estudo sobre a disponibilidade da implementação para uso externo.

- **Sistemas de orquestração e gerenciamento de NFV (MANO) [PS12]:** A solução proposta nesse estudo viabiliza o uso de um emulador que simula centenas ou milhares de PoPs (Point of Presence) e VIMs (gerenciadores de infraestrutura virtual) em uma máquina física ou virtual. Esse emulador baseia-se em Containernet (extensão do Mininet) e Containers Docker para simular as funções de rede virtualizadas (VNFs). A ideia central é usar um ambiente NFV leve, com uma NFVI (infraestrutura NFV) simplificada e APIs que imitam as do OpenStack, para testar de maneira automatizada as funcionalidades básicas de um sistema que gerencia e orquestra VNFs em múltiplos PoPs (MANO), como o onboarding, a catalogação, a instanciação e escalonamento de serviços, permitindo a detecção de problemas críticos antes da testagem em ambientes NFV completos. Não há informações no estudo sobre a disponibilidade da ferramenta para uso externo.

- **Appium + Open CV [PS13]:** A proposta é utilizar uma combinação de Appium com a biblioteca de visão computacional OpenCV para realizar testes smoke, de estresse e gráficos em aplicativos com interfaces gráficas de usuário (GUI) não padronizadas, como jogos feitos com Unity. O Appium possui a característica de realizar interações com o aplicativo semelhantes às do usuário final, como pressionar botões, selecionar caixas e inserir textos, conseguindo capturar e examinar screenshots dos aplicativos. Já a biblioteca OpenCV é empregada para realizar o reconhecimento de objetos e elementos de GUI

dentro das capturas de tela; a partir de funções da biblioteca, é possível encontrar correspondências aproximadas de padrões de imagem dentro da captura de tela e obter um coeficiente de similaridade. Não há informações no estudo sobre a disponibilidade da implementação para uso externo.

- **Tecnologia Azov [PS14]:** A tecnologia Azov tem seu uso na geração automática de testes de sanidade para sistemas com um grande número de operações de interface, nos quais as técnicas convencionais de desenvolvimento de suítes de teste se tornam muito trabalhosas. A partir de um banco de dados contendo informações estruturadas sobre as operações de interface do sistema sob teste (suas assinaturas) e a documentação das operações, é realizado o refinamento dos tipos de parâmetros e resultados das operações sob teste para possibilitar a geração de valores de parâmetros para cenários simples de uso normal e realizar verificações básicas. Não há informações no estudo sobre a disponibilidade da ferramenta para uso externo.

- **KubeSmokeTester [PS15]:** KubeSmokeTester é uma ferramenta que visa automatizar a geração de testes smoke. Os testes são gerados a partir de critérios [PS3] que exercitam elementos específicos de aplicações Kubernetes. A ferramenta também utiliza mecanismos para definir oráculos de acordo com os critérios escolhidos para a geração dos casos, como: verificar o status da plataforma, busca por erros em arquivos de log, verificação de erros retornados pelos endpoints e erros ao tentar acessar volumes persistentes. Além disso, a execução da ferramenta pode ser realizada em um container independente, sem a necessidade de alterações no sistema sob teste (SUT), possibilitando flexibilidade e fácil integração em pipelines de teste. A ferramenta está disponível para uso via npm.

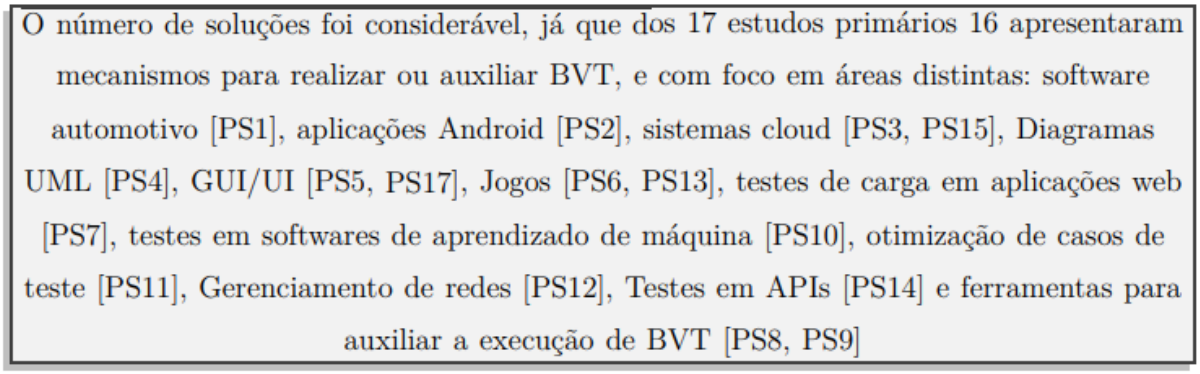
- **AppFlow [PS17]:** AppFlow é um sistema para sintetizar testes smoke para interface de usuário (GUI), utilizando aprendizado de máquina para reconhecer automaticamente telas e widgets comuns. Aplicativos da mesma categoria possuem funcionalidades similares, portanto o AppFlow permite que os desenvolvedores criem uma biblioteca de testes modulares para as funções principais de um aplicativo (por exemplo, um teste de “adicionar ao carrinho”

para aplicativos de compras). Em seguida, é possível testar rapidamente um novo aplicativo da mesma categoria, gerando testes completos a partir dos módulos da biblioteca. A ferramenta e seus componentes estão disponíveis para uso externo.

- **Técnicas de Árvore de Decisão [PS8]:** Essa solução pode ser utilizada como apoio à BVT. Foram utilizadas técnicas de árvore de decisão para extrair um conjunto de regras que podem ser usadas para prever a probabilidade de uma build ser aprovada ou reprovada no processo de certificação, tendo em vista que esse processo consome tempo, consequentemente causando atrasos no desenvolvimento. Para definir essas regras, foram mineradas informações históricas (alterações no código, resultados de certificações de builds) de um grande projeto de software em desenvolvimento nos laboratórios da IBM em Toronto. Essas regras são utilizadas para decidir se deve prosseguir com o teste da build mais recente ou alocar seus recursos em outras atividades até que a build seja certificada. Não há informações no estudo sobre a disponibilidade da implementação para uso externo.

- **VIFER [PS9]:** VIFER (View Identification Failure Repairer) pode ser considerada uma ferramenta auxiliar aos testes de verificação de build, pois o seu intuito é corrigir scripts de testes que possuem falhas de identificação de visão (VIFs). Essas falhas ocorrem quando há modificações na interface do usuário de um aplicativo, como alterações nos identificadores de view (IDs), ocasionando a quebra dos testes automatizados. A ferramenta, por meio de diversas métricas de similaridade (léxica, semântica, layout, etc.), explora as semelhanças entre os elementos de view antigos e novos para correlacionar os identificadores obsoletos com seus equivalentes atualizados, servindo como auxílio na manutenção dos scripts de BVT. Não há informações no estudo sobre a disponibilidade da ferramenta para uso externo.

Figura 6 – QP3: Ferramentas, metodologia e frameworks


O número de soluções foi considerável, já que dos 17 estudos primários 16 apresentaram mecanismos para realizar ou auxiliar BVT, e com foco em áreas distintas: software automotivo [PS1], aplicações Android [PS2], sistemas cloud [PS3, PS15], Diagramas UML [PS4], GUI/UI [PS5, PS17], Jogos [PS6, PS13], testes de carga em aplicações web [PS7], testes em softwares de aprendizado de máquina [PS10], otimização de casos de teste [PS11], Gerenciamento de redes [PS12], Testes em APIs [PS14] e ferramentas para auxiliar a execução de BVT [PS8, PS9]

Fonte: A autora (2025).

5 DESAFIOS E OPORTUNIDADES

Nesta seção serão apresentados os principais desafios e oportunidades identificados na revisão rápida.

- **(CH1) Desafio #1:** Falta de estudos/ferramentas para testes de sanidade. O processo de busca e leitura dos estudos primários mostrou que há pouca discussão acerca de testes de sanidade. Dos 17 estudos primários selecionados, apenas três [PS2, PS11, PS14] destacaram o tema, nos quais os estudos [PS2, PS14] apresentaram ferramentas especializadas para testes de sanidade. Isso mostra que o campo de pesquisa está mais concentrado nos testes smoke para a área de BVT, e há necessidade de mais desenvolvimento de ferramentas e experimentos em testes de sanidade.

- **(CH2) Desafio #2:** Tempo de execução em BVT. McConnell [7] discute que testes smoke podem ser rodados em builds diárias, e que essa prática traz uma série de pontos positivos. Além disso, é explicitado que alguns desenvolvedores defendem que é impraticável lançar builds todos os dias em projetos grandes, porém, em contrapartida, a empresa Microsoft realiza essa prática em seu sistema (Microsoft Windows NT 3.0), que consistia de 5,6 milhões de linhas de código espalhadas em 40.000 arquivos. Levando em consideração que testes de verificação de build devem ser executados diariamente (ou semanalmente em algumas empresas), eles devem possuir um tempo razoável de execução para que os testes subsequentes, que serão mais complexos, tenham tempo suficiente para serem executados [PS3]. Porém, apenas um dos estudos primários [PS11] desenvolveu uma solução na qual a temática fosse a melhora do tempo de execução. Sendo um fator crucial na área de testes automatizados, há escassez de estudos que desenvolvam alternativas para essa questão.

- **(CH3) Desafio #3:** Sistematização e boas práticas em BVT. Como destacado por Cannavacciuolo e Mariani [PS3], a construção dos conjuntos de testes smoke tem sido amplamente confiada à intuição do testador. É notável que há uma lacuna significativa de pesquisas que definam práticas recomendadas

gerais e específicas, dependendo da área de aplicação do BVT, além de critérios sistemáticos que podem ser seguidos para criar conjuntos de testes eficientes e efetivos.

- **(CH4) Desafio #4:** Dificuldade em testar a interface gráfica do usuário (GUI). A verificação da interface do usuário (GUI) de forma adequada foi uma problemática destacada por [PS5, PS17]. Ambos os estudos propuseram ferramentas para agilizar a testagem da GUI, por meio de geração de casos de testes, execução automática e avaliação dos resultados, evidenciando que esse desafio tem um grau de importância no âmbito da interface do usuário, e vem sendo debatido por estudiosos que desenvolvem soluções. Porém, as ferramentas são voltadas para testes smoke, evidenciando o desafio CH1.

- **(OP1) Oportunidade #1:** Desenvolvimento de ferramentas. Relacionando essa oportunidade com o desafio CH1, tendo em vista o leque limitado de ferramentas para testes de sanidade, o desenvolvimento de tecnologias especializadas, também para smoke, é uma oportunidade para praticantes da indústria e acadêmicos de trazerem novas contribuições para a área de BVT.

- **(OP2) Oportunidade #2:** Exploração da área. Por meio de uma rápida pesquisa em mecanismos de busca acadêmicos, é perceptível que, na comparação entre BVT e outros tipos de testes, há minoritariamente estudos focados. Sendo verificações que trazem diversos benefícios para as fases iniciais de validação do sistema, a exploração e experimentação desse campo pode trazer novos avanços em BVT.

- **(OP3) Oportunidade #3:** Exploração de novas tecnologias. Com o avanço cada vez maior em modelos de linguagem natural (LLMs), já utilizados em uma das soluções no estudo primário [PS6], os testes de sanidade e fumaça podem se beneficiar dessa tecnologia para auxiliar em uma maior automatização dos processos de geração de testes, execução e análise dos resultados, trazendo uma nova gama de ferramentas e estudos para o campo de BVT.

6 CONCLUSÃO

Os testes de verificação de build (BVT) possuem um papel importante na esteira de qualidade de um produto. Por serem executados logo nos estágios iniciais de verificação, a eficiência (tempo de execução) e a efetividade (descoberta de falhas) ganham maior relevância, uma vez que falhas detectadas nesse momento são críticas; sua identificação precoce reduz custos e retrabalho.

Diante do exposto, este trabalho apresenta uma revisão rápida sobre BVT. A busca por estudos primários sobre o tópico foi realizada nas bases ACM Digital Library, IEEE Xplore e SpringerLink, resultando em um conjunto de 1312 estudos. Após a aplicação dos critérios de inclusão e exclusão, 17 trabalhos foram selecionados, servindo como ponto de partida para responder às perguntas de pesquisa propostas e identificar os principais desafios e oportunidades nesse campo.

Constataram-se quatro principais desafios enfrentados por BVT: i) falta de estudos/ferramentas para testes de sanidade; ii) tempo de execução em BVT; iii) sistematização e boas práticas em BVT; iv) dificuldade em testar a interface gráfica do usuário (GUI). A partir dos dezessete estudos primários, foram identificadas apenas duas ferramentas especializadas em testes de sanidade, revelando que há uma lacuna nesse quesito.

Além disso, o tema do tempo de execução dessas suítes não foi abordado em muitos trabalhos, apesar de ser um fator de grande importância para BVT. Juntamente a isso, não há boas práticas difundidas, deixando o desenvolvimento a cargo da experiência e intuição do profissional. Por fim, a verificação adequada da GUI é um desafio destacado, que já possui ferramentas sendo desenvolvidas, porém apenas para testes de smoke.

Isso revela uma oportunidade para pesquisadores e praticantes de angariar material e ferramentas especializadas, agregando melhorias de eficácia e eficiência em testes de verificação de build. Outra oportunidade reside na experimentação de tecnologias emergentes, como os modelos de linguagem natural (LLMs), que podem contribuir com a automatização dos processos de geração, execução e análise dos resultados em BVT.

REFERÊNCIAS

[1] B. Cartaxo, G. Pinto, and S. Soares, “The role of rapid reviews in supporting decision-making in software engineering practice,” in Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE ’18). New York, NY, USA: Association for Computing Machinery, 2018, pp. 24–34. [Online]. Available: https://doi.org/10.1145/3210459.3210462

[2] C. Kaner, J. Bach, and B. Pettichord, Lessons Learned in Software Testing: A ContextDriven Approach. New York: Wiley, 2002.

[3] S. Desikan and G. Ramesh, Software Testing: Principles and Practice. Toronto: Pearson Education Canada, 2006.

[4] International Software Testing Qualifications Board. (n.d.) Glossary: Smoke test. Accessed: 2025-06-12. [Online]. Available: https://glossary.istqb.org/en_US/term/smoke-test?term=smoke%20test&exact_matches_first=true

[5] Exforsys Inc. (2012) Sanity testing. Accessed: 2025-06-12. [Online]. Available: https://www.exforsys.com/tutorials/testing-types/sanity-testing.html

[6] T. Hamilton. (2024) Key difference between smoke testing and sanity testing. Guru99. Accessed: 2025-06-13. [Online]. Available: https: //www.guru99.com/smoke-sanity-testing.html

[7] S. McConnell, “Daily build and smoke test,” IEEE Software, 1996.

[8] S. Nithin, H. S J, and P. S, “Enhancing the automotive software test environment using continuous integration and validation pipeline,” in 2023 Innovations in Power and Advanced Computing Technologies (i-PACT), 2023, pp. 1–6.

[9] A. C. Chagas, D. Gonzaga, L. Albuquerque, F. Oliveira, R. Castro, and L. Chaves, “Bsa tool: An experience report of software automation to perform sanity tests in a global software development environment,” in 2023 3rd International Conference on Information Communication and Software Engineering (ICICSE), 2023, pp. 15–20.

[10] C. Cannavacciuolo and L. Mariani, “Smoke testing of cloud systems,” in 2022 IEEE Conference on Software Testing, Verification and Validation (ICST), 2022, pp. 47–57.

[11] P. Jha and M. Sahu, “Smoke testing of uml activity diagrams: An approach for ensuring system reliability,” in 2023 IEEE Silchar Subsection Conference (SILCON), 2023, pp. 1–6.

[12] A. Memon, I. Banerjee, N. Hashmi, and A. Nagarajan, “Dart: a framework for regression testing ”nightly/daily builds” of gui applications,” in International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., 2003, pp. 410–419.

[13] J. Hu, M. Zhang, B. Liu, Y. Wu, and Y. Chen, “A language-guided acceleration method for smoke testing of game quests,” in 2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW), 2024, pp. 7–12.

[14] R. Khan and M. Amjad, “Web application’s performance testing using hp loadrunner and ca wily introscope tools,” in 2016 International Conference on Computing, Communication and Automation (ICCCA), 2016, pp. 802–806.

[15] A. E. Hassan and K. Zhang, “Using decision trees to predict the certification result of a build,” in 21st IEEE/ACM International Conference on Automated Software Engineering (ASE’06), 2006, pp. 189–198.

[16] J. Yoon, S. Chung, K. Shin, J. Kim, S. Hong, and S. Yoo, “Repairing fragile gui test cases using word and layout embedding,” in 2022 IEEE Conference on Software Testing, Verification and Validation (ICST), 2022, pp. 291–301.

[17] S. Herbold and T. Haar, “Smoke testing for machine learning: Simple tests to discover severe bugs,” Empirical Software Engineering, vol. 27, no. 2, p. 45, 2022. [Online]. Available: https://doi.org/10.1007/s10664-021-10073-7

[18] J. Varun and R. A. Karthika, “Achieving agility in projects through hierarchical divisive clustering algorithm,” Journal of Electronic Testing, vol. 38, no. 5, pp. 471–479, 2022. [Online]. Available: https://doi.org/10.1007/s10836-022-06024-9

[19] M. Peuster, M. Marchetti, G. García de Blas, and H. Karl, “Automated testing of nfv orchestrators against carrier-grade multi-pop scenarios using emulation-based smoke testing,” EURASIP Journal on Wireless Communications and Networking, vol. 2019, no. 1, p. 172, 2019. [Online]. Available: https://doi.org/10.1186/s13638-019-1493-2

[20] M. Mozgovoy and E. Pyshkin, “Unity application testing automation with appium and image recognition,” in Tools and Methods of Program Analysis, V. Itsykson, A. Scedrov, and V. Zakharov, Eds. Cham: Springer International Publishing, 2018, pp. 139–150.

[21] R. S. Zybin, V. V. Kuliamin, A. V. Ponomarenko, V. V. Rubanov, and E. S. Chernov, “Automation of broad sanity test generation,” Programming and Computer Software, vol. 34, no. 6, pp. 351–363, 2008. [Online]. Available: https://doi.org/10.1134/S0361768808060066

[22] C. Cannavacciuolo and L. Mariani, “Automatic generation of smoke test suites for kubernetes,” in Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, ser. ISSTA 2022. New York, NY, USA: Association for Computing Machinery, 2022, pp. 769–772. [Online]. Available: https://doi.org/10.1145/3533767.3543298

[23] K. Herzig and N. Nagappan, “The impact of test ownership and team structure on the reliability and effectiveness of quality test runs,” in Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and

Measurement, ser. ESEM ’14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2652524.2652535

[24] G. Hu, L. Zhu, and J. Yang, “Appflow: using machine learning to synthesize robust, reusable ui tests,” in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 269–282. [Online]. Available: https://doi.org/10.1145/3236024.3236055