Criação de um checklist de validação de *software* baseado no modelo ISO/IEC 25010

FELIPE MOLINA GARCIA
GUSTAVO FERNANDES ALTOÉ
KARINA ANGÉLICA TEODORO
MARCIO ANTONIO FACCI

Resumo

O estudo apresenta o desenvolvimento de um checklist abrangente para a validação de software, fundamentado no modelo de qualidade ISO/IEC 25010:2011. O checklist é uma ferramenta que pretende simplificar o processo de validação de softwares e, ao mesmo tempo, contribuir para a detecção precoce de defeitos, a redução de custos de retrabalho e a melhoria da qualidade do software entregue. Conclui-se que o checklist possui grande potencial no auxílio do desenvolvimento de softwares e sua adaptação para contextos específicos pode resultar em melhorias significativas na qualidade do produto.

Palavras-chave*:* Validação de Software; ISO/IEC 25010; Checklist;

Creating a Software Validation Checklist Based on the ISO/IEC 25010

Abstract

This study presents the development of a comprehensive checklist for software validation, based on the ISO/IEC 25010:2011 quality model. The checklist is a tool intended to simplify the software validation process while, simultaneously, contributing to the early detection of defects, reducing rework costs, and improving the quality of the delivered software. It concludes that the checklist has great potential in aiding software development, and its adaptation to specific contexts can result in significant improvements in product quality.

Keywords: *Software Validation; ISO/IEC 25010; Checklist.*

---

1 INTRODUÇÃO

A validação de *software* é um aspecto crucial no processo de desenvolvimento de sistemas, garantindo que o produto atenda aos requisitos especificados e às expectativas dos usuários. Neste contexto, a utilização de *checklists* surge como uma ferramenta valiosa para assegurar a qualidade e a consistência das atividades de validação.

A importância da validação de *software* tem sido amplamente reconhecida na literatura, com diversos estudos destacando seu papel fundamental na identificação de falhas e na melhoria da qualidade dos sistemas desenvolvidos (Bastos, 2016; Sommerville, 2016). A detecção precoce de problemas permite que ações corretivas sejam tomadas ainda nas fases iniciais do desenvolvimento, reduzindo custos e prazos de entrega.

Problema de Pesquisa: Diante da complexidade crescente no desenvolvimento de software e da necessidade de garantir a conformidade com padrões de qualidade, como uma ferramenta estruturada pode auxiliar equipes de desenvolvimento a avaliar sistematicamente a qualidade de seus produtos de *software*, promovendo a detecção precoce de defeitos e a melhoria contínua do processo?

Este artigo tem como objetivo principal desenvolver um *checklist* abrangente para a validação de software, baseado no modelo de qualidade ISO/IEC 25010:2011. Através da

criação e aplicação deste *checklist*, busca-se fornecer uma ferramenta prática e eficiente para auxiliar as equipes de desenvolvimento na avaliação da qualidade de seus produtos.

Para alcançar o objetivo principal, objetivos específicos a serem cumpridos incluem:

1. Realizar uma revisão da literatura, abordando conceitos de qualidade de *software*, tendências atuais no desenvolvimento de sistemas e a importância da inspeção e revisão no processo de validação.
2. Analisar estudos anteriores sobre a utilização de *checklists* em engenharia de *software* e a teoria por trás dessa abordagem.
3. Definir e estruturar um *checklist* de validação de *software*, seguindo critérios bem definidos para a inclusão de itens e baseando-se no modelo ISO/IEC 25010.

A metodologia adotada neste artigo envolve a criação do checklist de validação de *software*, seguindo critérios bem definidos para a inclusão de itens. O *checklist* será estruturado com base no modelo ISO/IEC 25010, e cada item será detalhado e justificado.

Espera-se que este estudo contribua para a área de engenharia de *software*, fornecendo uma ferramenta valiosa para a validação de sistemas e promovendo a adoção de boas práticas no processo de desenvolvimento. Ao final, serão apresentadas as conclusões e as recomendações para pesquisas futuras, visando o aprimoramento contínuo das técnicas de validação de software.

2 DESENVOLVIMENTO

Nesta seção, serão abordados os conceitos fundamentais relacionados à qualidade de *software*, às tendências atuais no desenvolvimento de sistemas e à importância da inspeção e revisão no processo de validação. Além disso, será discutida a experimentação em engenharia de software e sua relevância para a criação de *checklists* eficazes.

2.1 Quadro teórico

Conceitos de Qualidade de *Software*

A qualidade de *software* é um aspecto essencial para o sucesso de qualquer sistema. De acordo com a norma ISO/IEC 25010:2011 (que substitui a ISO/IEC 9126-1:2001), a qualidade de *software* pode ser definida como "a totalidade de características de um produto de software que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas". Este modelo de qualidade é composto por oito características principais de qualidade do produto: Adequação Funcional, Eficiência de Desempenho, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade e Portabilidade.

Desenvolvimento de *software* e tendências atuais

O desenvolvimento de *software* tem passado por constantes evoluções ao longo dos anos. Smith (2018) apresenta uma análise abrangente das tendências atuais no desenvolvimento de software, destacando a crescente adoção de metodologias ágeis, a integração contínua e a entrega contínua como práticas que visam melhorar a qualidade e a eficiência do processo de desenvolvimento. Além disso, a aplicação de análise de dados tem se mostrado crucial para a proposta de melhoria contínua de processos de qualidade de software. Pressman (2014) reforça a importância de uma abordagem sistemática e disciplinada para a engenharia de *software*, enfatizando a necessidade de um processo bem definido e da aplicação de boas práticas.

Importância da inspeção e revisão no desenvolvimento de software

A inspeção e a revisão são atividades fundamentais para garantir a qualidade do software desenvolvido. Fagan (1976) introduziu o conceito de inspeções de *software*, demonstrando sua eficácia na detecção de erros e na melhoria da qualidade do código. As inspeções envolvem a revisão sistemática do código por uma equipe de desenvolvedores, seguindo um processo estruturado e utilizando *checklists* para orientar a análise.

Experimentação em Engenharia de *Software*

A experimentação desempenha um papel crucial na engenharia de *software*, permitindo a validação de hipóteses e a obtenção de evidências empíricas sobre a eficácia de diferentes abordagens e técnicas. Basili, Selby e Hutchens (1992) destacam a importância da experimentação para o avanço da engenharia de *software*, apresentando uma metodologia para a condução de experimentos controlados. A aplicação de experimentos na criação e validação de *checklists* pode fornecer *insights* valiosos sobre sua eficácia e identificar oportunidades de melhoria.

Essa fundamentação teórica estabelece a base para a compreensão dos conceitos-chave relacionados à qualidade de *software*, às tendências de desenvolvimento e à importância da inspeção e revisão. Além disso, destaca-se a relevância da experimentação na engenharia de software, especialmente no contexto da criação e validação de *checklists*. Esses conceitos serão aplicados ao longo do artigo para embasar o desenvolvimento do *checklist* de validação de *software* proposto.

2.1.1 Revisão de Literatura

As Nesta seção, serão apresentados estudos anteriores relevantes sobre a utilização de *checklists* em engenharia de *software*, a aplicação de técnicas de checagem de modelos (*model checking*) e testes baseados em especificações de requisitos, bem como a teoria por trás dos *checklists*.

Estudos anteriores sobre checklists em engenharia de software

A utilização de *checklists* em engenharia de *software* tem sido objeto de diversos estudos. Kerth (2001) destaca a importância dos *checklists* para o controle de qualidade dos processos de desenvolvimento, apresentando diretrizes para a criação de *checklists* eficazes. O autor enfatiza que os *checklists* devem ser específicos, concisos e fáceis de usar, abordando os aspectos críticos do processo de desenvolvimento. Robbins e Finley (2000) apresentam uma visão geral sobre o poder dos *checklists* como uma ferramenta para garantir a qualidade e a consistência em diferentes áreas, incluindo a engenharia de *software*. Os autores destacam que os *checklists* ajudam a reduzir erros, melhorar a comunicação e aumentar a eficiência das equipes de desenvolvimento. Mais recentemente, estudos como o de Cerqueira, Mello e Travassos (2023) propuseram a criação de *checklists* específicos para a inspeção de privacidade e proteção de dados pessoais em artefatos de software, refletindo a crescente preocupação com compliance e segurança.

A utilização da checagem de modelos e testes a partir de especificações de requisitos

A aplicação de técnicas de checagem de modelos e testes baseados em especificações de requisitos tem se mostrado promissora para a validação de *software*. Heitmeyer, Kirby e Labaw (1996) propõem a utilização da checagem de modelos para gerar testes a partir de

especificações dos requisitos formais. Os autores demonstram que essa abordagem pode identificar erros e inconsistências nas especificações, contribuindo para a melhoria da qualidade do *software* como um todo. A geração de testes a partir de especificações de requisitos também é explorada por outros estudos, como o de Offutt e Abdurazik (1999), que apresentam uma técnica para gerar testes com base em modelos UML. Esses estudos evidenciam a importância de uma especificação de requisitos clara e precisa para a criação de testes de validação eficazes.

A teoria por trás dos checklists

Gawande (2009) apresenta uma análise abrangente sobre a teoria por trás dos *checklists* e sua aplicação em diversas áreas, incluindo a medicina e a aviação. O autor argumenta que os checklists são uma ferramenta poderosa para lidar com a complexidade e reduzir erros em sistemas complexos. Gawande (2009) destaca a importância de criar *checklists* concisos, focados nos pontos críticos e adaptados ao contexto específico em que serão utilizados. A teoria dos *checklists* também é explorada por Hales e Pronovost (2006), que apresentam um *framework* para a criação e implementação de *checklists* eficazes. Os autores enfatizam a necessidade de envolver as partes interessadas no processo de criação dos *checklists*, garantindo que eles sejam relevantes e alinhados com as necessidades específicas do contexto.

Essa revisão de literatura evidencia a relevância dos *checklists* na engenharia de *software*, bem como a aplicação de técnicas de checagem de modelos e testes baseados em especificações de requisitos. Além disso, a teoria por trás dos *checklists* fornece ferramentas valiosas para a criação de *checklists* eficazes e adaptados ao contexto da validação de *software*. Esses estudos servirão como base para o desenvolvimento do *checklist* proposto neste artigo.

2.2 Materiais e Métodos

A seguir, será apresentada a metodologia adotada para o desenvolvimento do *checklist* de validação de *software* proposto neste artigo. Serão descritos o processo de criação do *checklist*, os critérios utilizados para a inclusão de itens e a abordagem proposta para a validação do instrumento.

A metodologia adotada neste estudo é baseada em uma abordagem qualitativa, com foco no desenvolvimento de um *checklist* para a validação de *software*. Inicialmente, foi realizada uma revisão de literatura para identificar estudos relevantes sobre *checklists* em engenharia de *software*, bem como para compreender a teoria por trás dos *checklists* e sua aplicação em diferentes contextos. Em seguida, foi conduzido um processo sistemático para a criação do *checklist*, envolvendo a identificação dos principais aspectos a serem avaliados na validação de *software* e a definição de critérios para a inclusão de itens no instrumento.

O *checklist* foi estruturado com base no modelo de qualidade ISO/IEC 25010:2011 (que substituiu o ISO/IEC 9126-1:2001), visando abranger as características essenciais de qualidade de software.

Critérios utilizados para a inclusão de itens no checklist

Para a inclusão de itens no *checklist* de validação de *software*, foram considerados os seguintes critérios:

- Relevância: Os itens incluídos devem ser relevantes para a avaliação da qualidade do *software*, abordando aspectos críticos que impactam as características de qualidade do sistema.

- Clareza: Os itens devem ser redigidos de forma clara e objetiva, evitando ambiguidades e garantindo que possam ser facilmente compreendidos e aplicados pelos usuários do *checklist*.
- Verificabilidade: Os itens devem ser verificáveis, ou seja, deve ser possível avaliar se o software atende ou não aos critérios estabelecidos. Itens subjetivos ou de difícil verificação foram evitados.
- Abrangência: O conjunto de itens incluído no *checklist* deve ser abrangente o suficiente para cobrir os principais aspectos da validação de *software*, considerando as diferentes características de qualidade do modelo ISO/IEC 25010:2011.

Esses critérios foram aplicados durante o processo de criação do *checklist*, visando garantir que o instrumento seja eficaz, confiável e adequado para a validação de *software*. A metodologia apresentada nesta seção estabelece a base para o desenvolvimento do *checklist* proposto.

Desenvolvimento do *Checklist*

O desenvolvimento do *checklist* de validação de *software* proposto neste artigo foi estruturado com base no modelo de qualidade ISO/IEC 25010:2011, abrangendo as oito características principais de qualidade de software: Adequação Funcional, Eficiência de Desempenho, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade e Portabilidade. Cada item do *checklist* será detalhado, fornecendo uma base sólida para a avaliação e validação da qualidade do software.

Estruturação do Checklist

O *checklist* de validação de *software* foi estruturado seguindo as características de qualidade do modelo ISO/IEC 25010:2011. Cada característica foi desdobrada em subcaracterísticas, e para cada subcaracterística, foram definidos itens específicos a serem avaliados. Essa estruturação permite uma avaliação abrangente e sistemática da qualidade do software.

A estrutura do *checklist* é apresentada a seguir, baseada na ISO/IEC 25010:

- Adequação Funcional:
  - Adequação funcional: Capacidade do *software* de fornecer as funções necessárias para atender aos requisitos especificados.
  - Precisão: Capacidade do *software* de fornecer resultados corretos ou de acordo com os requisitos.
- Compatibilidade (substitui parte da Interoperabilidade e Coexistência):
  - Interoperabilidade: Capacidade do *software* de interagir com outros sistemas.
  - Coexistência: Capacidade do *software* de coexistir com outros sistemas.

- Segurança (nova característica na ISO/IEC 25010):
  - Confidencialidade: Capacidade de proteger a informação.
  - Integridade: Capacidade de prevenir acessos ou modificações não autorizadas.
  - Não repúdio: Capacidade de provar que ações ou eventos ocorreram.

- Confiabilidade:
  - Maturidade: Capacidade do *software* de evitar falhas devido a defeitos.
  - Tolerância a falhas: Capacidade do *software* de manter o desempenho em caso de falhas.

  - Capacidade de recuperação: Capacidade do *software* de recuperar dados após uma falha.
  - Conformidade: Capacidade do *software* de atender a padrões e regulamentos específicos.

- Usabilidade:
  - Compreensibilidade: Facilidade de compreensão do *software*.
  - Aprendizagem: Facilidade de aprendizado para novos usuários.
  - Operabilidade: Facilidade de operação e controle do *software*.
  - Atratividade: Estética e *design* atraente do *software*.
  - Conformidade com normas de usabilidade: Conformidade com princípios e diretrizes de usabilidade.

- Eficiência de Desempenho (substitui Eficiência):
  - Comportamento em relação ao tempo: Tempo de resposta e eficiência de processamento do *software*.
  - Comportamento em relação aos recursos: Uso eficiente dos recursos do sistema.
  - Conformidade com eficiência: Conformidade com padrões de eficiência estabelecidos.

- Manutenibilidade:
  - Analisabilidade: Facilidade de análise e compreensão do código fonte.
  - Modificabilidade: Facilidade de modificação e extensão do *software*.
  - Estabilidade: Resistência a falhas após modificações.
  - Testabilidade: Facilidade de teste e validação do *software*.

- Portabilidade:
  - Adaptabilidade: Capacidade do *software* de se adaptar a diferentes ambientes.
  - Instalabilidade: Facilidade de instalação do *software* em diferentes ambientes.
  - Substituibilidade: Capacidade do *software* de ser substituído por outro sistema sem perda de funcionalidade.

Justificativa para inclusão dos itens

Cada item incluído no *checklist* de validação de *software* foi cuidadosamente selecionado e justificado com base em sua relevância para a avaliação da qualidade do *software*. As justificativas foram embasadas na literatura existente, nas normas e padrões de qualidade de *software*, como a ISO/IEC 25010:2011, e nas melhores práticas de engenharia de *software*.

Por exemplo, os itens relacionados à Adequação Funcional foram incluídos para garantir que o *software* atenda aos requisitos especificados e produza resultados corretos e precisos (Sommerville, 2016). Os itens de Confiabilidade visam avaliar a capacidade do *software* de manter um nível aceitável de desempenho ao longo do tempo e lidar com falhas de forma adequada (Pressman, 2014). Os itens de Usabilidade foram incluídos para assegurar que o *software* seja fácil de usar, compreender e aprender, proporcionando uma experiência positiva para os usuários (Nielsen, 1993). Já os itens de Eficiência de Desempenho visam garantir que o *software* utilize os recursos do sistema de forma otimizada e responda em tempo hábil às solicitações dos usuários (Smith, 2018). Os itens de Manutenibilidade foram selecionados para avaliar a facilidade com que o software pode ser modificado, corrigido e evoluído ao longo do tempo, sem introduzir novos erros (Fowler, 2018). Por fim, os itens de Portabilidade visam

garantir que o *software* possa ser facilmente transferido e adaptado para diferentes ambientes operacionais, sem perda de funcionalidade (ISO/IEC, 2001).

Essas justificativas reforçam a importância e a relevância de cada item incluído no *checklist*, demonstrando sua contribuição para uma avaliação e validação abrangente da qualidade do *software*.

2.3 Resultados e discussões

Com base na estrutura proposta e nas premissas apresentadas para justificar a inclusão das subcaracterísticas, o questionário foi elaborado. O instrumento consiste em subitens, cada um com questões, abrangendo as oito características principais do modelo ISO/IEC 25010:2011. O instrumento de pesquisa compreende um total de cento e vinte perguntas, concebidas de modo a permitir respostas simples de "sim" ou "não" para determinar a conformidade ou não conformidade, ou podem ser respondidas de forma qualitativa, utilizando uma escala de 1 a 5, a fim de criar um mapa de calor e avaliar as áreas do *software* que demandam revisão para alcançar o nível de qualidade desejado.

A relevância da aplicação do modelo ISO/IEC 25010 para avaliação de qualidade em trabalhos de conclusão de curso e pesquisas acadêmicas tem sido recentemente confirmada, como em estudos realizados em 2024 sobre a medição e avaliação da qualidade de *software*.

Figura 1 – Página 1 do Questionário com escala qualitativa de 1 a 5

	Checklist de Validação de Software					
	Funcionalidade					
	I. Adequação Funcional	1	2	3	4	5
1	O software oferece todas as funções necessárias para atender aos requisitos especificados?					
2	Todas as funcionalidades essenciais para a operação do software estão presentes e funcionando corretamente?					
	II. Precisão	1	2	3	4	5
3	Os resultados produzidos pelo software estão corretos e de acordo com os requisitos especificados?					
4	O software fornece resultados precisos e confiáveis em todas as suas funcionalidades?					
	III. Interoperabilidade	1	2	3	4	5
5	O software é capaz de interagir de forma eficaz e sem problemas com outros sistemas ou plataformas?					
6	Ele é compatível com os padrões de comunicação e integração necessários para a sua utilização em um ambiente diversificado?					
	IV. Segurança Funcional	1	2	3	4	5
7	O software implementa medidas eficazes de segurança para proteger dados sensíveis e garantir operações seguras?					
8	Ele oferece recursos de autenticação, autorização e criptografia para proteger as informações do usuário?					
	V. Testes de Funcionalidade	1	2	3	4	5
9	Foram realizados testes abrangentes para validar a funcionalidade do software?					
10	Os testes incluíram casos de uso diversos para garantir que todas as funcionalidades do software sejam testadas adequadamente?					
	VI. Feedback do Usuário	1	2	3	4	5
11	O software foi avaliado por usuários finais quanto à sua funcionalidade?					
12	O feedback dos usuários foi levado em consideração para identificar possíveis melhorias ou ajustes na funcionalidade do software?					
	VII. Documentação de Funcionalidades	1	2	3	4	5
13	Existe documentação clara e abrangente que descreve as funcionalidades do software?					
14	Os usuários têm acesso a manuais ou guias que explicam como utilizar todas as funcionalidades oferecidas pelo software?					
	VIII. Resolução de Problemas	1	2	3	4	5
15	O software possui mecanismos eficazes para lidar com problemas ou erros de funcionalidade?					
16	Ele oferece suporte técnico ou canais de comunicação para que os usuários relatem problemas e recebam assistência adequada?					
	IX. Atualizações e Manutenção	1	2	3	4	5
17	O software é regularmente atualizado para corrigir falhas de funcionalidade e garantir sua eficácia contínua?					
18	Existem procedimentos estabelecidos para a manutenção e melhoria contínua da funcionalidade do software?					
	X. Conformidade com Padrões	1	2	3	4	5
19	O software está em conformidade com os padrões e regulamentos relevantes em relação à sua funcionalidade?					
20	Ele atende aos requisitos legais e normativos relacionados à segurança e desempenho das suas funcionalidades?					

Fonte: Autores

Figura 2 – Página 2 do Questionário com escala qualitativa de 1 a 5

	Confiabilidade					
	XI. Maturidade	1	2	3	4	5
21	O software demonstra maturidade em relação à sua capacidade de evitar falhas devido a defeitos?					
22	Ele passou por testes extensivos e estável o suficiente para ser utilizado em ambiente de produção sem ocorrência frequente de falhas?					
	XII. Tolerância a Falhas	1	2	3	4	5
23	O software é capaz de manter seu desempenho mesmo em caso de falhas inesperadas?					
24	Ele possui mecanismos de recuperação automática que permitem continuar a operar de forma adequada após a ocorrência de falhas?					
	XIII. Capacidade de Recuperação	1	2	3	4	5
25	O software possui capacidade de recuperação de dados após uma falha ou interrupção inesperada?					
26	Ele é capaz de restaurar os dados e o estado do sistema para um estado operacional válido após incidentes de falha?					
	XIV. Testes de Confiabilidade	1	2	3	4	5
27	Foram realizados testes abrangentes para validar a confiabilidade do software?					
28	Os testes incluíram simulações de falhas e cenários de recuperação para avaliar a capacidade do software de lidar com situações adversas?					
	XV. Monitoramento de Falhas	1	2	3	4	5
29	O software possui mecanismos de monitoramento que permitem detectar e registrar falhas de forma proativa?					
30	Ele oferece recursos de registro de eventos e geração de alertas para facilitar a identificação e resolução rápida de problemas de confiabilidade?					
	XVI. Backup e Restauração	1	2	3	4	5
31	Existem procedimentos estabelecidos para realizar backups regulares dos dados do software?					
32	O software oferece recursos de backup e restauração que permitem proteger os dados do usuário e garantir sua integridade em caso de falhas?					
	XVII. Conformidade com Padrões	1	2	3	4	5
33	O software está em conformidade com os padrões e regulamentos específicos relacionados à confiabilidade?					
34	Ele atende aos requisitos legais e normativos em relação à segurança e integridade dos dados?					
	XVIII. Resiliência a Ataques	1	2	3	4	5
35	O software possui medidas de segurança para proteger contra ataques maliciosos e tentativas de comprometer sua confiabilidade?					
36	Ele implementa práticas de segurança robustas para minimizar o risco de violações de dados e falhas de segurança?					
	XIX. Histórico de Falhas	1	2	3	4	5
37	Existe um registro de histórico de falhas que permite analisar e aprender com incidentes passados?					
38	O software utiliza informações de falhas anteriores para melhorar sua confiabilidade e evitar a repetição de problemas similares?					
	XX. Auditorias de Confiabilidade	1	2	3	4	5
39	São realizadas auditorias periódicas para avaliar a confiabilidade do software e garantir a conformidade com os padrões estabelecidos?					
40	O software está sujeito a revisões regulares para identificar possíveis vulnerabilidades e áreas de melhoria em relação à sua confiabilidade					

Fonte: Autores

Figura 3 – Página 3 do Questionário com escala qualitativa de 1 a 5

	Usabilidade					
	XXI. Compreensibilidade	1	2	3	4	5
41	O software apresenta uma interface clara e intuitiva que facilita a compreensão de suas funcionalidades?					
42	Os menus, botões e elementos de navegação são consistentes e de fácil interpretação para os usuários?					
	XXII. Aprendizagem	1	2	3	4	5
43	O software oferece recursos e guias que facilitam o aprendizado para novos usuários?					
44	Os usuários podem rapidamente aprender a utilizar as principais funcionalidades do software sem a necessidade de treinamento extensivo?					
	XXIII. Operabilidade	1	2	3	4	5
45	O software é de fácil operação e controle, permitindo que os usuários realizem suas tarefas de forma eficiente?					
46	Ele oferece atalhos e recursos de personalização que facilitam a realização de operações comuns?					
	XXIV. Atratividade	1	2	3	4	5
47	O design e a interface do software são visualmente atraentes e esteticamente agradáveis?					
48	Os elementos gráficos e de design são cuidadosamente elaborados para proporcionar uma experiência agradável ao usuário?					
	XXV. Conformidade com Normas de Usabilidade	1	2	3	4	5
49	O software está em conformidade com os princípios e diretrizes de usabilidade estabelecidos em normas reconhecidas?					
50	Ele segue padrões de design e interação recomendados pela indústria para garantir uma experiência de usuário consistente e intuitiva?					
	XXVI. Feedback do Usuário	1	2	3	4	5
51	O software oferece feedback claro e imediato para as ações realizadas pelo usuário?					
52	Os usuários recebem indicações visuais ou mensagens informativas que os orientam durante a utilização do software?					
	XXVII. Customização da Interface	1	2	3	4	5
53	O software permite que os usuários personalizem a interface de acordo com suas preferências e necessidades?					
54	Eles podem ajustar configurações de layout, cores e fontes para tornar a experiência de uso mais confortável e produtiva?					
	XXVIII. Facilidade de Navegação	1	2	3	4	5
55	A estrutura de navegação do software é lógica e intuitiva, facilitando a localização de funcionalidades e informações?					
56	Os usuários podem encontrar facilmente o que estão procurando, sem se perderem em menus complexos ou caminhos confusos?					
	XXIX. Testes de Usabilidade	1	2	3	4	5
57	Foram realizados testes de usabilidade com usuários reais para avaliar a eficácia da interface e da experiência de uso?					
58	Os resultados dos testes foram utilizados para identificar áreas de melhoria e realizar ajustes na usabilidade do software?					
	XXX. Documentação e Suporte	1	2	3	4	5
59	Existe documentação clara e acessível que fornece orientações sobre o uso do software?					
60	Os usuários têm acesso a recursos de suporte, como tutoriais online ou assistência técnica, para resolver dúvidas ou problemas relacionados à usabilidade do software?					

Fonte: Autores

Figura 4 – Página 4 do Questionário com escala qualitativa de 1 a 5

	Eficiência					
	XXXI. Comportamento em Relação ao Tempo	1	2	3	4	5
61	O software apresenta um tempo de resposta satisfatório para as principais operações e funcionalidades?					
62	As ações executadas pelo usuário são processadas de forma rápida e eficiente, sem longos períodos de espera?					
	XXXII. Comportamento em Relação aos Recursos	1	2	3	4	5
63	O software utiliza os recursos do sistema de forma eficiente e otimizada?					
64	Ele consome uma quantidade razoável de memória, CPU e outros recursos do sistema durante sua execução?					
	XXXIII. Otimização de Processos	1	2	3	4	5
65	O software utiliza algoritmos e técnicas eficientes para otimizar processos e operações computacionais?					
66	Ele minimiza o tempo de execução e maximiza a utilização dos recursos disponíveis?					
	XXXIV. Minimização de Overhead	1	2	3	4	5
67	O software evita desperdícios de recursos e minimiza o overhead durante sua execução?					
68	Ele não sobrecarrega desnecessariamente o sistema com processos ou tarefas que não contribuem para sua funcionalidade principal?					
	XXXV. Monitoramento de Desempenho	1	2	3	4	5
69	Existem mecanismos de monitoramento que permitem avaliar o desempenho do software em relação ao tempo e aos recursos?					
70	Os usuários têm acesso a métricas e indicadores de desempenho que permitem acompanhar o uso de recursos e identificar possíveis gargalos?					
	XXXVI. Otimização de Algoritmos	1	2	3	4	5
71	Os algoritmos e métodos utilizados pelo software foram otimizados para garantir eficiência e desempenho máximo?					
72	Foram realizados testes de desempenho para validar a eficácia das otimizações implementadas?					
	XXXVII. Conformidade com Padrões de Eficiência	1	2	3	4	5
73	O software está em conformidade com os padrões e normas estabelecidos em relação à eficiência de processamento e uso de recursos?					
74	Ele segue diretrizes reconhecidas pela indústria para garantir um desempenho adequado e eficiente?					
	XXXVIII. Escalabilidade	1	2	3	4	5
75	O software é capaz de lidar com um aumento no volume de dados ou de usuários sem comprometer seu desempenho?					
76	Ele foi projetado para escalar de forma eficiente e suportar um crescimento gradual na demanda?					
	XXXIX. Otimização de Consultas e Acesso a Dados	1	2	3	4	5
77	As consultas e operações de acesso a dados realizadas pelo software são otimizadas para minimizar o tempo de resposta?					
78	Ele utiliza índices, cache e outras técnicas para acelerar a recuperação e manipulação de informações?					
	XL. Testes de Desempenho	1	2	3	4	5
79	Foram realizados testes de desempenho abrangentes para avaliar a eficiência do software em condições reais de uso?					
80	Os resultados dos testes foram utilizados para identificar oportunidades de melhoria e otimização de desempenho?					

Fonte: Autores

Figura 5 – Página 5 do Questionário com escala qualitativa de 1 a 5

	Manutenibilidade					
	XLI. Analisabilidade	1	2	3	4	5
81	O código fonte do software é bem estruturado e documentado, facilitando sua análise e compreensão?					
82	Os desenvolvedores são capazes de identificar e entender rapidamente o funcionamento do código para realizar diagnósticos e correções?					
	XLII. Modificabilidade	1	2	3	4	5
83	O software é projetado de forma modular e extensível, permitindo que novas funcionalidades sejam adicionadas com facilidade?					
84	As mudanças no software podem ser feitas de forma rápida e eficiente, sem comprometer a integridade ou a estabilidade do sistema?					
	XLIII. Estabilidade	1	2	3	4	5
85	O software mantém sua estabilidade e integridade mesmo após modificações ou atualizações?					
86	Ele é resistente a falhas e erros inesperados que possam surgir como resultado de alterações no código ou na configuração?					
	XLIV. Testabilidade	1	2	3	4	5
87	O software é facilmente testável, permitindo a validação eficaz de suas funcionalidades e modificações?					
88	Existem recursos e ferramentas disponíveis para automatizar os testes e garantir uma cobertura abrangente do código?					
	XLV. Padrões de Codificação	1	2	3	4	5
89	O software segue padrões de codificação e boas práticas estabelecidas pela indústria?					
90	Ele utiliza convenções de nomenclatura consistentes e estruturas de código padronizadas para facilitar a manutenção e colaboração entre desenvolvedores?					
	XLVI. Refatoração de Código	1	2	3	4	5
91	São realizadas atividades de refatoração de código regularmente para melhorar sua qualidade e manutenibilidade?					
92	O código é periodicamente revisado e otimizado para remover duplicações, simplificar estruturas complexas e melhorar a legibilidade?					
	XLVII. Documentação de Código	1	2	3	4	5
93	O código fonte é devidamente documentado, fornecendo informações claras sobre sua estrutura, funcionalidades e finalidades?					
94	Os comentários e documentação interna ajudam os desenvolvedores a entender o código e realizar alterações de forma segura?					
	XLVIII. Gestão de Configuração	1	2	3	4	5
95	Existem processos estabelecidos para gerenciar e controlar as versões do software e suas configurações?					
96	As alterações no código são registradas e controladas de forma adequada para garantir a rastreabilidade e integridade do sistema?					
	XLIX. Reutilização de Componentes	1	2	3	4	5
97	O software promove a reutilização de componentes e módulos para evitar a duplicação de esforços e aumentar a eficiência no desenvolvimento?					
98	Ele utiliza bibliotecas e frameworks externos de forma eficaz para incorporar funcionalidades comuns e evitar a reinvenção da roda?					
	L. Políticas de Manutenção	1	2	3	4	5
99	Existem políticas e procedimentos estabelecidos para orientar a manutenção e evolução contínua do software?					
100	As decisões de desenvolvimento são tomadas com base em critérios de manutenibilidade e sustentabilidade a longo prazo?					

Fonte: Autores

Figura 6 – Página 6 do Questionário com escala qualitativa de 1 a 5

	Portabilidade					
	LI. Adaptabilidade	1	2	3	4	5
101	O software é projetado de forma a ser compatível e adaptável a diferentes ambientes e sistemas operacionais?					
102	Ele é capaz de funcionar sem problemas em uma variedade de plataformas e configurações de hardware?					
	LII. Instalabilidade	1	2	3	4	5
103	A instalação do software é fácil e intuitiva, mesmo em diferentes ambientes e sistemas operacionais?					
104	Os usuários podem instalar o software sem a necessidade de conhecimentos técnicos avançados ou assistência especializada?					
	LIII. Coexistência	1	2	3	4	5
105	O software é capaz de coexistir harmoniosamente com outros sistemas e aplicativos que estão sendo executados no mesmo ambiente?					
106	Ele não interfere nas operações de outros sistemas e não causa conflitos de recursos ou incompatibilidades?					
	LIV. Substituibilidade	1	2	3	4	5
107	O software pode ser facilmente substituído por outro sistema sem perda de funcionalidade ou interrupção nos processos de negócio?					
108	Os dados e configurações do usuário podem ser migrados de forma transparente para o novo sistema sem impacto negativo?					
	LV. Padrões de Interface	1	2	3	4	5
109	O software utiliza padrões de interface e protocolos de comunicação amplamente reconhecidos e suportados?					
110	Ele segue convenções de design e interação que facilitam a integração com outros sistemas e a interoperabilidade?					
	LVI. Compatibilidade com Versões Anteriores	1	2	3	4	5
111	As atualizações e novas versões do software são compatíveis com versões anteriores, garantindo uma transição suave e sem problemas para os usuários?					
112	Os dados e configurações existentes podem ser facilmente migrados e importados para novas versões do software?					
	LVII. Testes de Portabilidade	1	2	3	4	5
113	Foram realizados testes abrangentes de portabilidade para validar a capacidade do software de funcionar em diferentes ambientes e configurações?					
114	Os testes incluíram simulações de ambientes variados para garantir que o software seja robusto e confiável em todas as condições?					
	LVIII. Documentação de Instalação	1	2	3	4	5
115	Existe documentação clara e detalhada que fornece orientações sobre a instalação do software em diferentes plataformas?					
116	Os usuários têm acesso a instruções passo a passo que os ajudam a configurar e implantar o software em seus ambientes específicos?					
	LVIX. Compatibilidade de Dados	1	2	3	4	5
117	O software é capaz de importar e exportar dados em formatos compatíveis com outros sistemas e aplicativos?					
118	Ele suporta padrões de troca de dados que facilitam a interoperabilidade e integração com sistemas externos?					
	LX. Atualizações e Patches	1	2	3	4	5
119	As atualizações e patches de segurança são distribuídos de forma a garantir a continuidade do software e a correção de possíveis vulnerabilidades?					
120	Os usuários recebem notificações e orientações sobre como aplicar atualizações de forma segura e eficiente?					

Fonte: Autores.

Implicações práticas do *checklist*

Observa-se que o *checklist* de validação de *software* desenvolvido possui implicações práticas significativas para o desenvolvimento de *software*. Ao fornecer uma ferramenta estruturada e abrangente para avaliar a qualidade do *software*, o *checklist* pode auxiliar as equipes de desenvolvimento a identificarem pontos fortes e oportunidades de melhoria em seus projetos. A aplicação regular do *checklist* durante o processo de desenvolvimento pode contribuir para a detecção precoce de defeitos e para a garantia da qualidade do *software* entregue. Isso pode levar a uma redução nos custos de retrabalho, a uma maior satisfação dos usuários e a uma maior confiança na qualidade do software produzido. Além disso, o *checklist* pode ser utilizado como uma ferramenta de comunicação entre as equipes de desenvolvimento, teste e qualidade, estabelecendo um entendimento comum sobre os critérios de qualidade a serem atendidos. Essa comunicação aprimorada pode facilitar a colaboração e o alinhamento entre as diferentes áreas envolvidas no processo de desenvolvimento de *software*.

3 CONSIDERAÇÕES FINAIS

Neste artigo, foi apresentado o desenvolvimento de um *checklist* abrangente para a validação de *software*, baseado no modelo de qualidade ISO/IEC 25010:2011. O objetivo principal foi fornecer uma ferramenta estruturada e sistemática para avaliar a qualidade do *software*, considerando as características essenciais de qualidade do produto.

O resultado deste trabalho foi a criação de um questionário teórico contendo cento e vinte perguntas, estruturado em subcaracterísticas que abordam os critérios de qualidade estabelecidos pela ISO/IEC 25010. Este questionário pode ser aplicado de forma binária ("sim" ou "não") para verificar a conformidade ou de forma qualitativa (escala de 1 a 5) para mapear áreas que exigem melhorias. Este instrumento representa o principal resultado teórico do estudo, servindo como uma base sistemática para a avaliação da qualidade de *software*.

Para garantir que o *checklist* seja uma ferramenta útil, será necessária à sua validação em estudos de casos posteriores. Contudo, estudos onde foram aplicados checklists, como os de Kalinowski *et al*. (2018) e Mäntylä e Lassenius (2009), demonstraram a eficácia dos *checklists* na identificação de defeitos e na melhoria da qualidade do *software*.

Existem limitações quanto à usabilidade deste *checklist*, uma vez que depende da expertise e do julgamento das pessoas envolvidas na aplicação dele. No entanto, entende-se que, em um ambiente de desenvolvimento de *software*, os colaboradores estejam familiarizados com os conceitos aplicados no questionário. Além disso, sessões de treinamento podem ser aplicadas para alinhamento entre os avaliadores, visando garantir uma aplicação consistente do *checklist*.

Pesquisas futuras poderiam explorar a integração do *checklist* com outras práticas e ferramentas de garantia da qualidade de *software*, como testes automatizados, revisões de código e análise estática. A combinação do *checklist* com essas práticas poderá potencializar seus benefícios e fornecer uma abordagem ainda mais abrangente para a validação de *software*.

Conclui-se que existe grande potencial na aplicação do checklist no desenvolvimento de *softwares*. Com a adaptação e customização do *checklist* para contextos específicos, levando em consideração as particularidades de diferentes culturas organizacionais, os resultados de melhoria e garantia da qualidade serão positivos.

REFERÊNCIAS

BASTOS, F. C. **Qualidade de software: conceitos e práticas**. Novatec Editora. 2016.

BRASILI, V. R., SELBY, R. W., HUTCHENS, D. H. Experimentation in Software Engineering. **IEEE Transactions on Software Engineering**, v. 18, n. 7, 1992. 625-635 p.

CERQUEIRA, D. A.; MELLO, R. M.; TRAVASSOS, G. H. **Um checklist para inspeção de privacidade e proteção de dados pessoais em artefatos de software**. Anais do Congresso Ibero-Americano em Engenharia de Software (CIbSE), 2023.

FAGAN, M. E. Design and Code Inspections to Reduce Errors in Program Development. **IBM Systems Journal**, v. 15, n. 3, 1976. 182-211 p.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. 2ª ed. Addison-Wesley Professional. 2018.

GAWANDE, A. **The Checklist Manifesto: How to Get Things Right**. Metropolitan Books. 2009.

HEITMEYER, C. L., KIRBY, J., LABAW, B. G. Using Model Checking to Generate Tests from Requirements Specifications. **IEEE Transactions on Software Engineering**, v. 22, n. 3, 1996. 181-197 p.

ISO/IEC **International Organization for Standardization**. ISO/IEC 9126-1:2001 - Software engineering - Product quality - Part 1: Quality model. 2001.

KALINOWSKI, M., MENDES, E., TRAVASSOS, G. H. A Systematic Review on the Benefits and Drawbacks of Checklist-Based Inspections in Software Development. **Journal of Systems and Software**, v. 137, 2018. 279-295 p.

KERTH, N. Checklists for the Quality Control of Development Processes. **IEEE Software**, v. 18, n. 4, 2001. 53-59 p.

LANNA, J. G. M.; SILVA, A. P. **Aplicação da ISO/IEC 25010:2011 para medição e avaliação da qualidade de um software.** Monografia (Graduação em Engenharia de Produção) - Universidade Federal de Ouro Preto, João Monlevade, 2024.

MÄNTYLÄ, M. V., LASSENIUS, C. What Types of Defects Are Really Discovered in Code Reviews? **IEEE Transactions on Software Engineering**, 2009. 430-448 p.

NIELSEN, J. **Usability Engineering**. Morgan Kaufmann, 1993.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 8ª ed. McGraw-Hill Education, 2014.

ROBBINS, H. FINLEY, M. The Power of the Checklist: A Fresh Look at an Old Tool. The **Journal for Quality and Participation**, 2000. 36-37 p.

RAMOS, R. G. et al. Qualidade de software: análise de dados e proposta de melhoria de processo. **Revista Sociedade Científica**, v. 7, n. 1, 2024.

SMITH, J. **Software Development Trends: A Comprehensive Analysis**. Academic Press, 2018.

SOMMERVILLE, I. **Engenharia de Software**. 10ª ed. Pearson, 2016.

"Os conteúdos expressos no trabalho, bem como sua revisão ortográfica e das normas ABNT são de inteira responsabilidade do(s) autor(es)."
«Declaração de IA generativa e tecnologias assistidas por IA no processo de redação»
“Declara-se pelos autores que durante a preparação deste trabalho foram utilizados ChatGPT 3.5, You.ai, Gemini e Hix para auxiliar na estruturação, escrita, formatação, citação e correção ortográfica. Após utilizar estas ferramentas e serviços, os autores editaram e revisaram o conteúdo conforme necessário e assumem total responsabilidade pelo conteúdo da publicação.”