Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Bacharelado em Engenharia de Software
Trabalho de Conclusão de Curso – Artigo

CONSTRUÇÃO DE UMA FERRAMENTA CASE PARA INSPEÇÃO DE ARTEFATOS DE SOFTWARES UTILIZANDO TÉCNICAS DE INSPEÇÃO BASEADAS EM CHECKLIST

Lucas Melo Godinho[1]
Centro Universitário Academia - UniAcademia, Juiz de Fora, MG
Romualdo Monteiro de Resende Costa[2]
Centro Universitário Academia - UniAcademia, Juiz de Fora, MG
Rafael Maiani de Mello[3]
Universidade Federal do Rio de Janeiro - UFRJ/COPPE, Rio de janeiro, RJ

Linha de Pesquisa: Engenharia de Software

RESUMO

Este artigo apresenta a construção de uma ferramenta CASE desenvolvida para auxiliar na inspeção de artefatos de software, utilizando técnicas baseadas em checklists para identificar defeitos desde a fase de requisitos até o código fonte. A importância da inspeção de software é destacada ao longo do projeto, sendo a escolha da técnica adequada um fator crucial para o planejamento e os resultados esperados. As técnicas baseadas em checklists proporcionam uma análise sistemática, com perguntas "sim" ou "não" respondidas pelos inspetores durante a revisão dos artefatos. A ferramenta tem como objetivo oferecer um suporte eficiente à execução das inspeções, registrando defeitos e auxiliando no acompanhamento das correções. Além disso, oferece a funcionalidade de customizar técnicas de inspeção cadastradas para atender às necessidades específicas de cada projeto, podendo melhorar a qualidade, auxiliar na correção dos defeitos de forma ágil e contribuir para o sucesso do projeto e a satisfação do cliente.

Palavras-chave: Inspeção de Software, Checklist, Ferramenta CASE, Desenvolvimento de Software.

[1] Discente do Curso de Sistemas de Informação do Centro Universitário Academia – UniAcademia.
[2] Docente do Curso de Sistemas de Informação do Centro Universitário Academia - UniAcademia.
[3] Docente do Instituto de Computação da Universidade Federal do Rio de Janeiro - UFRJ

1 INTRODUÇÃO

A inspeção de software é uma prática fundamental para garantir a qualidade dos produtos desenvolvidos, desempenhando um papel crucial em todas as fases do ciclo de vida do desenvolvimento de software, assim como a escolha adequada de técnicas de inspeção é decisiva para o sucesso dessas atividades, tendo influência direta nos resultados obtidos e a eficiência do processo de desenvolvimento.

Com o avanço da inspeção, várias técnicas baseadas em checklists foram desenvolvidas, como ActCheck (Mello, 2011), ArqCheck (Barcelos e Travassos, 2006), FMCheck (Mello, 2012), LGPDCheck(Cerqueira, 2023), entre outras. No entanto, esse tipo de técnica ainda é pouco utilizado na indústria. Dessa forma, a motivação deste trabalho foi contribuir para essas técnicas, levando ao planejamento e construção de uma ferramenta que auxilia na inspeção das mesmas.

Este trabalho foi desenvolvido a partir de uma revisão da literatura relacionada à inspeção de software e suas técnicas, com ênfase nas técnicas baseadas em checklists. A revisão permitiu compreender o processo de inspeção de software, desde o planejamento até a identificação de discrepâncias pelos inspetores.

A ferramenta CASE *(Computer-Aided Software Engineering)*(Britannica, 2023) em questão, é chamada EasyCheck, que tem como objetivo, apoiar a aplicação de técnicas de inspeção de software baseadas em *checklists*. As inspeções baseadas em checklists são sistemáticas e estruturadas, proporcionando aos revisores um conjunto de perguntas diretas que ajudam a identificar defeitos em artefatos de software, desde a fase de requisitos até o código-fonte.

A EasyCheck foi planejada para ser uma aplicação Web, oferecendo serviços para o cadastro de projetos, técnicas de inspeção e aplicação dessas. A ferramenta visa melhorar a eficiência das inspeções, permitindo que os inspetores apliquem checklists personalizados para cada tipo de artefato, registrem defeitos encontrados e acompanhem as correções necessárias. Além disso, a ferramenta permite a geração e exportação de relatórios de discrepâncias, facilitando a análise, o controle e, consequentemente, a comunicação dos resultados.

Este artigo detalha, portanto, na seção 3 o planejamento e na seção 4 construção da ferramenta, apresentando as minúcias de como é feita a inspeção de um artefato de software utilizando a EasyCheck.

2. REFERENCIAL TEÓRICO

Após a realização do estudo prévio, o projeto foi concebido a partir de reuniões com professores do grupo de Engenharia de Software Experimental da Coppe/UFRJ[4], com o objetivo de se apresentar como uma inovação na área.

Apesar da pretensa inovação, o contexto de criação foi baseado em estudos prévios realizados por especialistas, que embasaram o desenvolvimento do projeto e dão respaldo à compreensão de termos essenciais para o entendimento da ferramenta. Nas próximas subseções, esses estudos são apresentados.

2.1. INSPEÇÃO DE SOFTWARE

Introduzida por Michael Fagan (Fagan, 1976), a inspeção de software é um tipo de revisão que pode ser aplicada em todos os artefatos de software, e pode contribuir para reduzir drasticamente o retrabalho e garantir uma melhor qualidade do software.

Trata-se de um método rigoroso e bem definido de revisão, cujo objetivo é a descoberta antecipada de defeitos. Levando em conta que o custo de correção de defeitos aumenta exponencialmente ao longo do ciclo de vida do software, a programação de tarefas que visem a detecção precoce de defeitos é crucial para a redução desse custo e para a melhoria da qualidade do produto. (Kalinowski e Travassos, 2004)

O processo tradicional de inspeção de software desenvolvido por Fagan (Fagan, 1976), consiste em seis atividades principais:

- **Planejamento**: O moderador define o contexto da inspeção, escolhendo a técnica de detecção de defeitos, o documento a ser inspecionado e seu autor. Em seguida, seleciona os inspetores e distribui o material para a inspeção.
- **Apresentação**: Nesta etapa, os autores dos artefatos a serem inspecionados explicam suas características. Caso os inspetores já estejam familiarizados com o projeto e os artefatos, esta fase pode ser dispensada.

[4] Página do grupo de Engenharia de Software Experimental da Coppe/UFRJ http://lens-ese.cos.ufrj.br/ese/

- **Preparação**: Os inspetores analisam os artefatos individualmente, tomando notas e identificando discrepâncias. Técnicas de leitura estruturada podem auxiliar na realização desta tarefa, facilitando a identificação de problemas.
- **Reunião**: Ocorre uma reunião com a participação do moderador, dos inspetores e dos autores dos documentos. Durante esta reunião, as discrepâncias são discutidas e classificadas como defeitos ou falsos positivos. O moderador tem a palavra final sobre a classificação das discrepâncias. A resolução dos defeitos não é discutida nesta fase, e a reunião deve ser limitada a duas horas para manter a eficácia dos participantes. Se necessário, a inspeção pode continuar no dia seguinte.
- **Retrabalho**: Os autores corrigem os defeitos identificados e confirmados durante a reunião.
- **Continuação**: Os artefatos corrigidos são revisados pelo moderador, que avalia a qualidade da inspeção e decide se uma nova inspeção é necessária ou se o processo pode ser concluído.

Como normalmente são realizadas por uma equipe, as inspeções de software permitem a troca de conhecimentos técnicos sobre os aspectos positivos e negativos dos artefatos de software entre os membros do grupo. Além disso, ao habituar os desenvolvedores a revisar os artefatos uns dos outros, essas inspeções incentivam a criação de artefatos mais legíveis com o tempo. (Melo *et al*., 2001)

No entanto, as inspeções são uma prática antiga cujo principal objetivo é a detecção de defeitos. Recentemente, há evidências empíricas de que a detecção de defeitos é mais uma atividade mais individual do que coletiva. Vale ressaltar que os resultados das inspeções são completamente determinados pelos próprios inspetores, suas estratégias para entender os documentos inspecionados e as ferramentas ou suportes disponíveis para eles durante o exercício de inspeção. (Oladele e Adedayo 2014)

2.2. TÉCNICAS DE INSPEÇÃO DE SOFTWARE BASEADAS EM CHECKLIST

De acordo com (Mello, 2011), a escolha da técnica de inspeção a ser utilizada é um dos fatores cruciais no planejamento e nos resultados da inspeção de um projeto de software. Podendo variar no custo e eficácia, dependendo do tipo da técnica escolhida.

As técnicas de leitura ad hoc e Checklist-based Reading (CBR) continuam sendo as mais usadas na indústria. O CBR, de fato, é considerado a técnica padrão nas organizações de software (Laitenberge e DeBaud, 2002). Na leitura ad hoc, geralmente não há suporte técnico disponível. Os revisores confiam em suas próprias habilidades, conhecimentos e experiências para identificar defeitos. Na melhor das hipóteses, os inspetores participam de alguma sessão de treinamento em compreensão de programas antes de iniciar o processo de revisão. (Oladele, 2010)

No CBR, o artefato é examinado com a ajuda de uma lista predefinida de perguntas – um checklist. Esta técnica é mais estruturada e acredita-se que oferece mais suporte ao revisor em comparação com a leitura ad hoc (Oladele, 2010). Muitos estudos empíricos envolvem leitura ad hoc e outra técnica de leitura, CBR e outra técnica de leitura, ou ad hoc, CBR e outra técnica. No entanto, são raros os estudos empíricos que investigam diretamente a eficácia comparativa do CBR e da leitura ad hoc. (Oladele e Adedayo, 2014)

A inspeção baseada em checklist é um tipo de inspeção que utiliza uma estrutura em que os inspetores respondem a perguntas do tipo “sim/não” enquanto revisam um documento de software. (Laitenberger *et al.,* 2001 *apud* Mello, 2011)

A seguir, as subseções apresentam alguns exemplos de técnicas de inspeção de softwares baseadas em checklists:

2.2.1 ArqCheck

O ArqCheck (Barcelos e Travassos, 2006) é uma técnica que visa melhorar a qualidade da arquitetura de software por meio da revisão do documento arquitetural. Esse documento não precisa seguir um padrão específico, mas deve atender a alguns requisitos da norma IEEE 1417, como identificar os elementos da solução, descrever seus papéis, explicar como os requisitos arquiteturais são atendidos e representar o software sob diferentes perspectivas.

Os itens de avaliação de ArqCheck são divididos em três grupos: consistência do documento (avaliando visões modulares, dinâmicas, de alocação e de contexto), atendimento aos requisitos (verificando funcionalidades e elementos arquiteturais) e abordagem de requisitos de qualidade (desempenho, disponibilidade, modificabilidade, segurança, testabilidade e usabilidade). A Tabela 1 mostra um exemplo de itens para verificação da consistência do documento**.**

Tabela 1 - Trecho do checklist de ArqCheck (Barcelos e Travassos, 2006)

Itens de avaliação da consistência de representações entre os diagramas (específicos à abordagem e documentação arquitetural utilizada)				
Nº	**Visão Modular**	**Sim**	**Não**	**NA**
3	Os módulos internos de cada cluster foram descritos em algum diagrama da visão Modular?			
4	Todo relacionamento definido com um cluster foi devidamente mapeado para um de seus módulos internos?			
Nº	**Visão Dinâmica**	**Sim**	**Não**	**NA**
5	Toda porta/interface possui um nome, é utilizada com um único propósito e de forma única?			
6	Os fluxos de execução, descritos na visão dinâmica, alocam todos os módulos definidos na visão Modular?			
7	Todo módulo/cluster representado na visão dinâmica foi descrito na visão Modular?			
8	Todo fluxo entre dois elementos arquiteturais pode ser mapeado para algum relacionamento da visão Modular?			
9	Todo relacionamento descrito na visão modular pode ser mapeado para algum fluxo de comunicação, de dados ou de controle da visão Dinâmica?			
Nº	**Visão de Alocação**	**Sim**	**Não**	**NA**
10	Todo módulo/cluster, representado na visão de alocação, foi descrito na visão Modular?			
11	Toda dependência, representada na visão de Alocação, pode ser mapeada para um ou mais relacionamentos da visão Modular?			
12	Dado os módulos/clusters representados na visão de Alocação, todos os relacionamentos definidos entre eles na visão Modular também foram representados na visão de Alocação			

Fonte: (Barcelos e Travassos, 2006)

2.2.2 ActCheck

ActCheck (Mello, 2011) é uma técnica de inspeção individual para diagramas de atividades, especificamente aqueles que fazem parte da especificação de requisitos de projetos de software e utilizam a sintaxe da UML 2 (OMG, 2009). Baseada em checklist configurável, ActCheck permite a aplicação seletiva de itens de dois checklists (A e B).

A configuração dos checklists ocorre na fase de planejamento da inspeção. Com a equipe de inspetores definida, o desenvolvedor responsável deve responder um questionário para caracterizar a aplicação segundo a técnica de modelagem adotada. Com base nas respostas, o moderador configura os checklists A e B, selecionando os itens relevantes com auxílio de tabelas de rastreabilidade.

Após o planejamento, cada inspetor realiza a inspeção individualmente, preenchendo os checklists e registrando as discrepâncias encontradas em um relatório. A Tabela 2 mostra alguns exemplos de itens de avaliação do checklist B da técnica.

Tabela 2 - Trecho do checklist B de ActCheck (Mello , 2011)

#	Item de Avaliação	Resposta
1	As ações e condições da atividade estão consistentes entre si?	( )Sim ( )Não ( )N.A.
3	Existem ações sendo executadas concorrentemente, mas que são executadas sequencialmente em outras partes do modelo?	( )Sim ( )Não ( )N.A
4	Existem ações semelhantes/ idênticas sendo executadas em raias ou sub-raias distintas?	( )Sim ( )Não ( )N.A
7	As propriedades dos objetos estão sendo respeitadas ao longo da atividade?	( )Sim ( )Não ( )N.A
10	De acordo com o seu conhecimento do domínio, todas as condições da Atividade são viáveis?	( )Sim ( )Não ( )N.A
12	Alguma ação corresponde à outra atividade do projeto, mas esta referência não está explícita no Diagrama de Atividades?	( )Sim ( )Não ( )N.A
14	Todas as outras atividades referenciadas nas ações do Diagrama de Atividades existem no projeto?	( )Sim ( )Não ( )N.A

Fonte: (Mello , 2011)

2.3. CATEGORIAS DE DEFEITO

No contexto das técnicas de inspeção baseadas em checklist apresentadas, ArqCheck (Barcelos e Travassos, 2006) e ActCheck (Mello, 2011), a categorização dos defeitos utilizada segue o modelo sugerido por (Travassos *et al*., 1999) e (Shull *et al*., 1999). A Tabela 3 apresenta cinco categorias, com definição dos conceitos e exemplos. (Mello, 2011)

Tabela 3 - Categoria de defeitos (adaptada de Travassos *et al*., 1999)

Categoria	Definição
Omissão	Informações necessárias não estão apresentadas no artefato.
Fato incorreto	Certas informações no artefato de software divergem das informações descritas na especificação de requisitos ou do conhecimento comum sobre o domínio.
Inconsistência	As informações em uma seção do artefato de software estão em desacordo com aquelas presentes em outras partes do mesmo artefato.
Ambiguidade	As informações no artefato de software são ambíguas, permitindo interpretações variadas pelo desenvolvedor, o que pode resultar em uma implementação incorreta.
Informação Estranha	As informações fornecidas são desnecessárias e não são utilizadas.

Fonte: (adaptada de Travassos *et al*., 1999)

O termo discrepância se refere a um possível defeito encontrado, uma discrepância pode ser considerada um defeito de fato ou um falso positivo. Para classificar os defeitos encontrados nas revisões, parte-se do princípio de que todos os artefatos gerados durante o desenvolvimento de software são baseados no documento de requisitos ou em artefatos derivados deste. Assim, as categorias de defeito correspondem aos tipos de defeitos presentes nos documentos de requisitos, acrescidos dos defeitos introduzidos durante a transformação de artefatos ao longo do desenvolvimento de software. (Kalinowski e Travassos, 2004)

É importante destacar que essas classes genéricas de defeitos podem ser subdivididas em categorias mais específicas conforme necessário. Além disso, essa classificação não é definitiva, e cada organização pode adicionar mais tipos de defeitos conforme suas próprias necessidades. (Shull, 2000)

3. PLANEJAMENTO

Como mencionado na Seção 2, o planejamento da ferramenta foi realizado por meio de reuniões. Durante essas reuniões, foram fornecidos detalhes essenciais para o desenvolvimento da ferramenta, culminando na construção de um documento de especificação de requisitos. Nesse documento, foram discutidas e definidas todas as funcionalidades desejadas para a ferramenta.

O documento criado passou por diversas correções e apontamentos por parte dos pesquisadores integrantes do grupo de pesquisa, até que chegasse em uma versão fechada e acordada por todas as partes. Com o resultado disso, foram definidos os requisitos essenciais para o desenvolvimento da ferramenta. Nas próximas subseções são apresentadas as especificações constantes desse documento, através dos casos de uso e os modelos do projeto.

3.1. ESPECIFICAÇÃO DA FERRAMENTA

3.1.1 CASOS DE USO

A seguir, serão apresentados os casos de uso desenvolvidos para ilustrar a aplicação EasyCheck.

- **Efetuar Login:** A ferramenta deve permitir o login dos usuários e a autenticação deve ser feita por nome de usuário e senha.

O usuário que possuir cadastro como moderador, deverá confirmar o perfil de inspetor ou moderador logo após a autenticação.

- **Cadastrar Projetos:** A ferramenta deve permitir cadastrar, editar e excluir projetos.

O projeto deve ser gerenciado por moderadores e representa o artefato de software ou um conjunto de artefatos a serem inspecionados pelos inspetores, e deve ser gerenciado por moderadores. O projeto deve oferecer a opção de anexar arquivos em PDF para definição dos artefatos.

- **Cadastrar Técnicas de Inspeção:** A ferramenta deve permitir cadastrar, editar e excluir técnicas de inspeção baseadas em checklist.

O gerenciamento deve ser feito por moderadores no módulo “Técnicas de Inspeção”, onde deve haver a possibilidade de incluir Checklists, itens dos Checklists, e as categorias de defeito da técnica.

- **Adicionar Inspeção:** A ferramenta deve permitir que os moderadores adicionem inspeções aos projetos cadastrados, onde poderão definir o inspetor e a técnica de inspeção a ser utilizada. Ao adicionar uma inspeção, a técnica de inspeção escolhida pode ser customizada conforme a necessidade, levando em consideração quais itens do checklist são obrigatórios ou não.

Após a adição da inspeção ao projeto, o inspetor designado terá acesso aos artefatos do projeto e poderá aplicar a técnica de inspeção.

- **Aplicar Técnica de Inspeção:** A ferramenta deve permitir aplicar as técnicas de inspeção que foram previamente cadastradas.

A aplicação da técnica deve ser realizada pelo inspetor e consistirá em responder aos itens do checklist e, quando necessário, preencher discrepâncias de acordo com os artefatos definidos no projeto. O inspetor poderá aplicar uma técnica de inspeção após o moderador ter adicionado uma inspeção para ele.

- **Exportar Relatório de Discrepâncias:** A ferramenta deve permitir a exportação em formato CSV do relatório de discrepâncias.

O relatório de discrepâncias terá um modelo padrão para todas as técnicas de inspeção. A Tabela 4, mostra, como exemplo, o template do arquivo csv do relatório de discrepâncias.

Tabela 4: Template csv do relatório de discrepâncias.

	A	B	C	D	E	F
1	nome_inspetor	data_inspecoa	tempo_inspecao	projeto	atividade	tecnica_inspecao
2	Lucas Melo	08/03/2024	100 minutos	EasyCheck		ActCheck
3	#	item_relacionado	tipo_defeito	descricao	localizacao	onde_repete
4	1	5	Inconsistência	Algum estereótipo foi utilizado incorretamente em alguma ação.	Seção 1	

5	2	22	Informação estranha	Uma ação da atividade descreve a implementação do sistema, o que é inadequado para a descrição de um caso de uso	Seção 1	Seção 2
6	3	2	Ambiguidade	O rótulo de uma raia ou sub-raia não está claro, não identificando as ações relacionadas.	Seção 2	
7	4	8	Inconsistência	Algum estereótipo foi utilizado incorretamente em alguma ação.	Seção 1	
8	5	29	Omissão	Alguma ação referente a comportamento interno do sistema não foi referenciada	Seção 4	Seção 5
9	6	1	Informação estranha	Certa informação não é necessária	Seção 3	
10	7	13	Fato incorreto	Uma ação x da atividade descreve a implementação do sistema	Seção 3	Seção 5
11	8	6	Fato Incorreto	Uma ação y da atividade descreve a implementação do sistema	Seção 5	

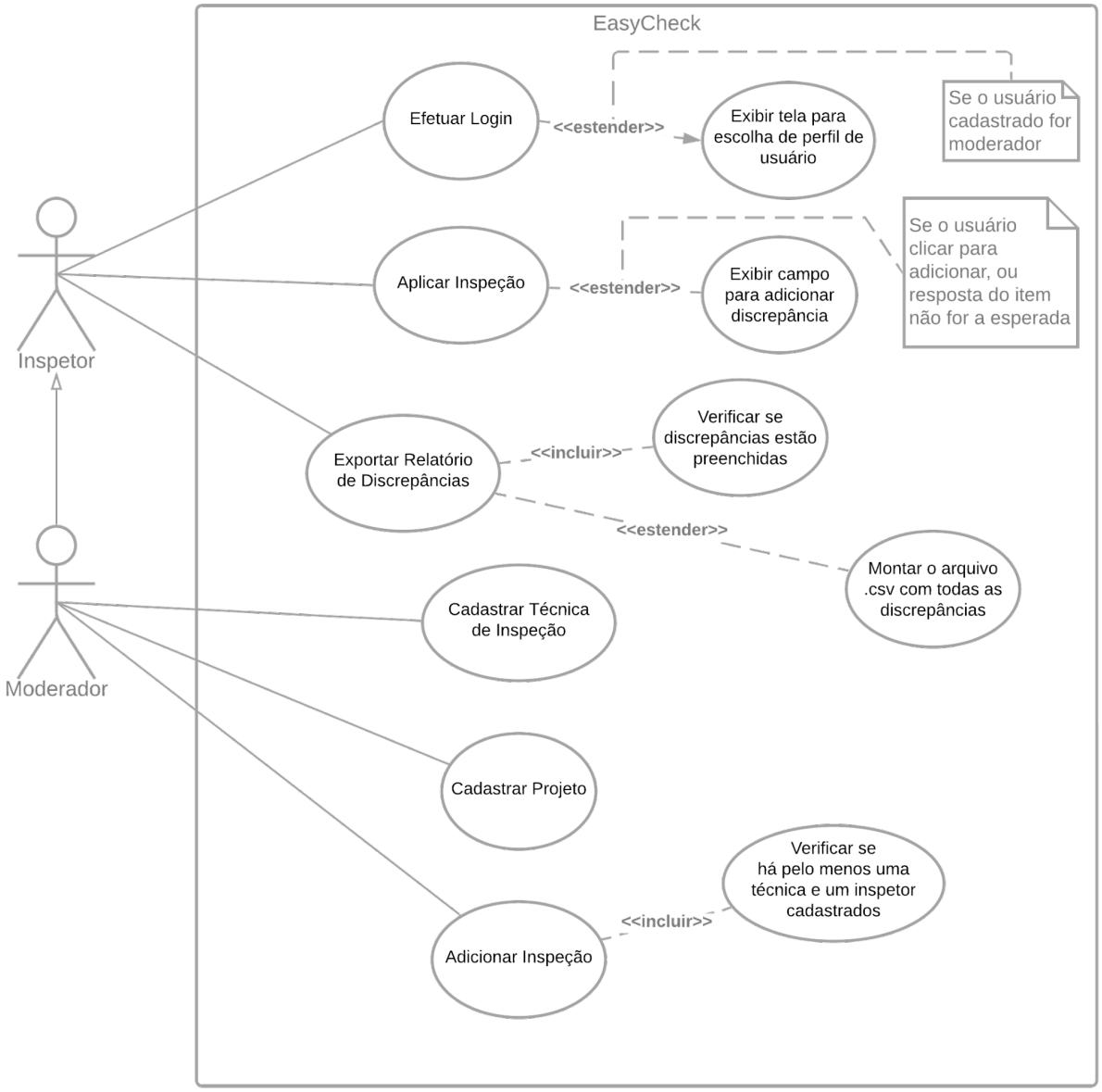
Fonte: do autor

Com base nestes casos de uso, foi realizada uma descrição detalhada de cada um, contendo os atores, as pré-condições, os fluxos principais e alternativos e, caso necessário, as regras de negócios, discutidas durante o processo de levantamento dos requisitos.

Para complementar a análise e facilitar a compreensão das interações entre os diferentes componentes da ferramenta, apresenta-se, a seguir, o diagrama de casos de uso. Este diagrama visa permitir a visualização, de maneira mais clara, das principais funcionalidades da ferramenta e a relação entre os atores envolvidos.

Na figura 2 é apresentado o modelo do diagrama. Essa representação gráfica serve como uma ferramenta importante para identificar e comunicar os requisitos do sistema, bem como para garantir que todas as partes interessadas tenham uma compreensão comum das funcionalidades propostas.

Figura 1: Modelo do diagrama de casos de uso.

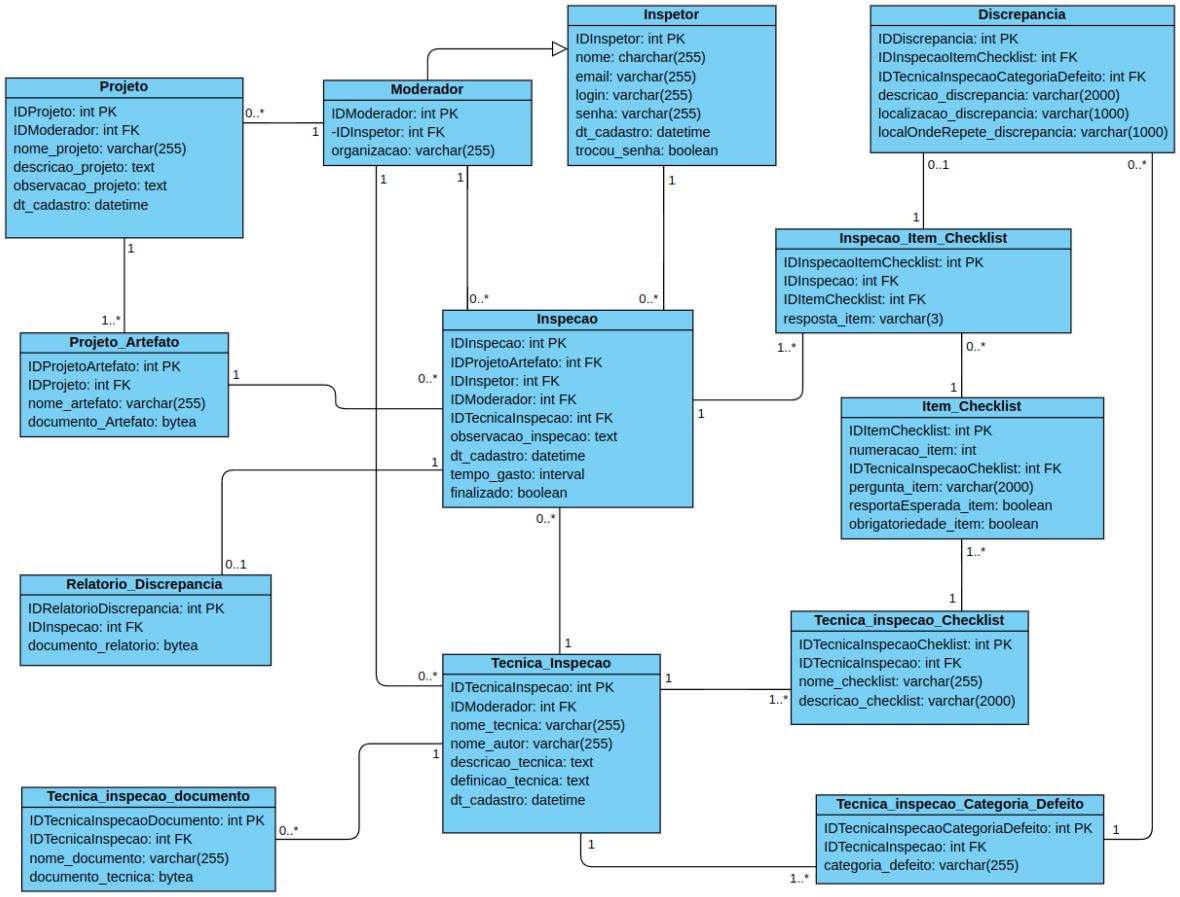


Fonte: do autor

3.1.2. DIAGRAMA DE CLASSES

A Figura 3, apresenta o diagrama de classes, que descreve a estrutura do sistema, incluindo as classes, seus atributos e as relações entre elas. Este diagrama fornece uma visão da organização interna do sistema e das interações entre seus componentes, facilitando a compreensão da arquitetura do software. Além disso, serviu de base para a construção do banco de dados da ferramenta, que será descrito na seção 5.1.

Figura 3: Modelo do diagrama de classes.



Fonte: do autor

3.2. CONSIDERAÇÕES FINAIS DO PLANEJAMENTO

Durante o período de planejamento, foram desenvolvidos diversos materiais adicionais, incluindo a definição dos termos utilizados, a arquitetura da ferramenta, a escolha das tecnologias e o modelo de interface. Esses elementos foram fundamentais para o amadurecimento da ideia da ferramenta, possibilitando o início do processo de desenvolvimento do código.

O desenvolvimento teve início somente após a conclusão de todos os conceitos no documento de especificação dos requisitos. No entanto, o processo de planejamento se estendeu por aproximadamente dois meses após o início da criação do código, devido às correções das marcações indicadas pelos professores.

4. DESENVOLVIMENTO

Pode-se observar que as técnicas de inspeção baseadas em checklist apresentadas no referencial teórico, ArqCheck (Barcelos e Travassos, 2006) e ActCheck (Mello, 2011), seguem um padrão semelhante ao serem aplicadas. Ao revisar um artefato de software, o inspetor basicamente responde aos itens de avaliação e, eventualmente, preenche as discrepâncias, que são numeradas em um relatório.

Com base nesse padrão, a EasyCheck foi desenvolvida trazendo funcionalidades que tornam possível a aplicação dessas e de outras técnicas de inspeção baseadas em checklist. O público-alvo são os profissionais da área de tecnologia da informação, e o objetivo é apoiar a aplicação de técnicas de inspeção de software baseadas em checklist.

Todas as funcionalidades estão disponíveis para dois perfis de usuário: moderador e inspetor. O moderador é responsável por cadastrar componentes que possibilitam a aplicação de inspeções pelo inspetor. O inspetor, por sua vez, aplica as inspeções e, em seguida, gera o relatório de discrepâncias conforme o planejamento do moderador.

O planejamento do moderador envolve o cadastro de componentes essenciais para a aplicação de uma inspeção, como técnicas de inspeção, projetos e a adição de inspeções a esses projetos. Durante o processo de adição de uma inspeção, o moderador pode customizar a técnica de inspeção, removendo itens não obrigatórios.

A cada nova inspeção adicionada pelo moderador, o inspetor recebe uma notificação na tela inicial da ferramenta. Todas as informações relevantes para a inspeção, cadastradas pelo moderador, estão disponíveis para o inspetor, incluindo detalhes do projeto, os artefatos a serem inspecionados, informações da técnica de inspeção e da própria inspeção. Isso facilita o acompanhamento do andamento das técnicas de inspeção, e todas as etapas são registradas para consultas futuras.

Durante a aplicação das técnicas, o relatório de discrepâncias segue um formato padrão para todas elas, e o inspetor pode exportá-lo em formato CSV quando finalizado. Cada item do checklist tem uma "resposta esperada", orientando o usuário no preenchimento do relatório de discrepâncias e indicando possíveis defeitos.

Em resumo, a EasyCheck permite que profissionais criem e gerenciem projetos, inspeções e técnicas de inspeção, gerando relatórios de discrepâncias de maneira eficiente e organizada após a devida aplicação. A EasyCheck destaca-se

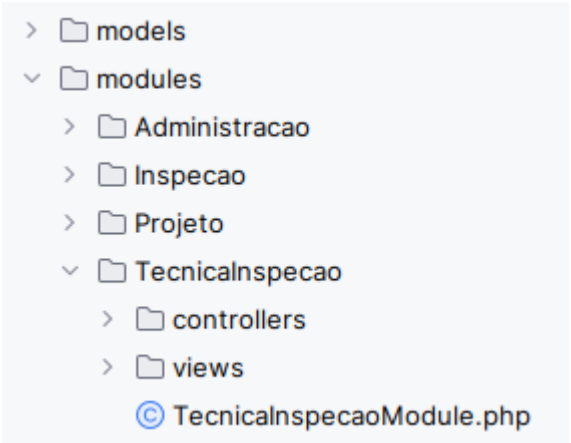
como uma inovação na área por ser a primeira ferramenta a oferecer essas funcionalidades específicas para técnicas de inspeção baseadas em checklists.

4.1. CONSTRUÇÃO

A partir da concepção da ideia de uma aplicação Web, as tecnologias escolhidas para o desenvolvimento foram definidas durante o período de planejamento. O critério de seleção baseou-se na experiência do desenvolvedor com as tecnologias de desenvolvimento web, resultando na escolha da linguagem PHP juntamente com o framework Yii na versão 1.1, que adota o padrão de arquitetura MVC (Model-View-Controller) e javascript juntamente com Ajax e jQuery para a visualização.

É importante destacar que o padrão arquitetural MVC geralmente segue os princípios da orientação a objetos. Portanto, a modelagem das classes foi crucial para a construção do banco de dados e para a organização das models, controllers e views. A Figura 4 ilustra como a divisão dos arquivos da ferramenta é organizada conforme este padrão arquitetural.

Figura 4: Padrão MVC



Fonte: do autor

Grande parte do código, principalmente a relacionada às operações básicas do banco de dados, como, criar, ler, atualizar e excluir, foi gerada por uma ferramenta do próprio framework, chamada gii, onde é possível gerar, a partir de um template configurável e a tabela do banco de dados, o código e a estrutura de todos os componentes do MVC.

Pode-se dizer que foi reduzido consideravelmente o tempo de desenvolvimento com a utilização do gii, porém, por gerar o código referente a uma tabela de cada vez, foi necessário fazer todas as adaptações que o gii não consegue cobrir, como a persistência correta dos dados ao criar e atualizar os registros no banco de dados, regras de negócio e a parte de visualização.

4.1.1. BANCO DE DADOS

Como mencionado anteriormente, o banco de dados foi desenvolvido com base na lógica delineada no diagrama de classes e implementado utilizando o PostgreSQL. De acordo com o site oficial do PostgreSQL, este é um sistema de banco de dados objeto-relacional robusto e de código aberto que expande a linguagem SQL, incorporando diversos recursos para armazenar e escalar com segurança as cargas de dados mais complexas. Suas origens remontam a 1986, como parte do projeto POSTGRES na Universidade da Califórnia em Berkeley, e ele possui mais de 35 anos de desenvolvimento ativo.

Para a configuração inicial, utilizou-se o recurso de Migration do framework yii. Neste contexto, isso significa que foram empregados métodos específicos do framework para a manipulação do esquema e do banco de dados. Em cada migração, o método up() deve ser sobrescrito para conter a lógica de atualização do banco de dados, enquanto o método down() deve incluir a lógica de reversão. O comando yii migrate gerencia todas as migrações disponíveis em uma aplicação. Como o PostgreSQL suporta transações, é possível sobrescrever safeUp() e safeDown() para garantir que, em caso de erro durante a atualização ou reversão, toda a migração seja revertida.

Inicialmente, foram criados dois esquemas: o “public” e o “desenvolvimento”. Cada um desses esquemas possui suas próprias tabelas e funções. O esquema “desenvolvimento” foi criado especificamente para o desenvolvimento da ferramenta, mas também tem a função de representar qualquer outro esquema relacionado a uma organização que possa ser criado no futuro, como “UniAcademia” e “Coppe”. Esses esquemas que representam organizações funcionam como bancos de dados separados, com seus próprios inspetores, moderadores, projetos, inspeções, etc.

O esquema “public”, como o nome sugere, serve de utilidade para a aplicação em geral e possui tabelas que armazenam dados utilizados por todos os outros

esquemas que representam organizações. Dessa maneira, o banco de dados da ferramenta foi construído de forma escalável, permitindo que mais de uma organização utilize a ferramenta sem interferências entre si. As Figuras 7 e 8 mostram as tabelas de cada schema mencionado.

Figura 7: Schema public

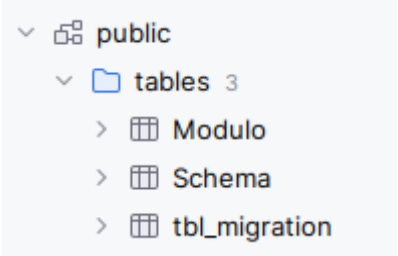
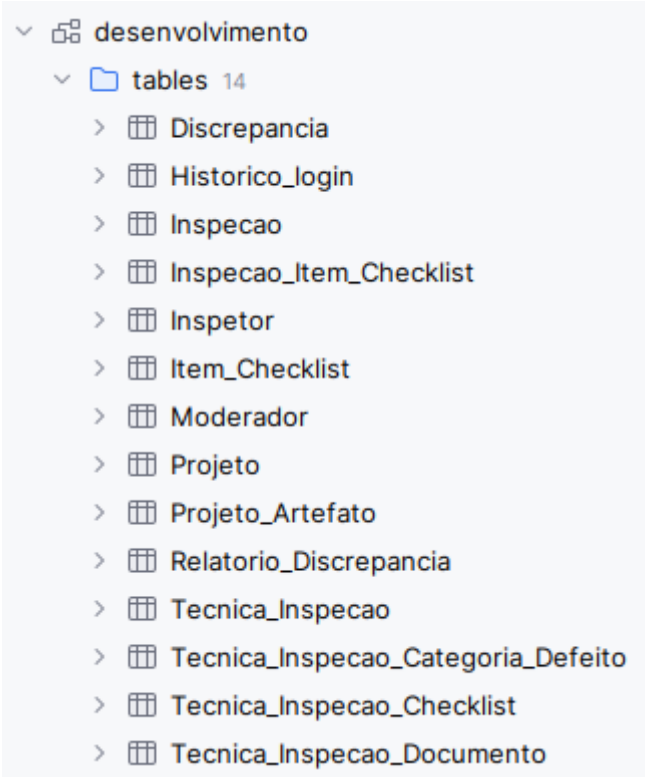


Figura 8: Schema desenvolvimento



O código da ferramenta está disponível no repositório do GitLab no seguinte endereço: https://gitlab.com/lucasmgodinho/easycheck

5. UTILIZAÇÃO DA FERRAMENTA

Nesta seção é mostrado exemplos de utilização da ferramenta. Após o inspetor efetuar o login na ferramenta, caso seja cadastrado como moderador, ele poderá escolher o papel que irá exercer, entre inspetor e moderador, caso contrário, ele sempre efetuará o login como inspetor. O moderador vai se deparar com a tela inicial da ferramenta e três módulos disponíveis: projetos, inspeções e técnicas de inspeções. A Figura 9 mostra parte da tela principal da ferramenta após o login como moderador.

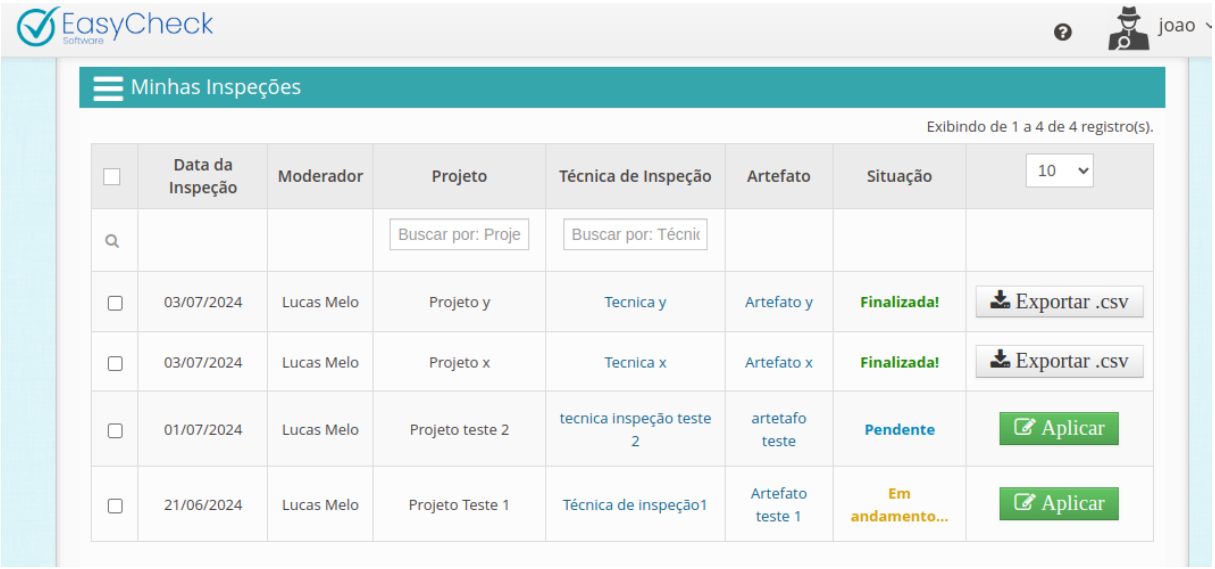
Figura 9: Tela inicial acessada a partir do papel de moderador.

A seguir são apresentadas as funcionalidades de cada módulo apresentado:

- **Projeto:** Possibilita o cadastro de projetos com suas devidas informações, como o nome do projeto, moderador, descrição, observação e os artefatos de software dos projetos.
- **Técnicas de inspeção:** Possibilita o cadastro de técnicas de inspeção baseadas em checklist com suas devidas informações, como o nome da técnica, autor, descrição, definição, documentos referentes a técnica, categorias de defeitos, os checklists e seus respectivos itens de avaliação. Cada item de avaliação terá o campo da pergunta, a resposta esperada e se o item é obrigatório para a inspeção.
- **Inspeções:** Módulo onde é possível cadastrar inspeções, possibilita o moderador escolher o projeto, o artefato deste projeto, a técnica de inspeção a ser utilizada, e os inspetores que farão a aplicação. Neste módulo também é possível customizar a técnica removendo os itens de avaliação que não foram definidos como obrigatórios no cadastro da técnica de inspeção.

As inspeções cadastradas por moderadores serão disponibilizadas para os inspetores fazerem a aplicação. O inspetor pode acessar todas as suas inspeções pelo módulo de inspeções a partir da tela inicial após efetuar o login. A Figura 10 mostra parte da tela principal da ferramenta após o login como inspetor.

Figura 10: Tela do módulo Inspeção acessada a partir do papel de inspetor



A seguir são apresentadas as funcionalidades do módulo de inspeção, acessado a partir de um inspetor:

- **Aplicar inspeção:** Possibilita ao inspetor aplicar a inspeção que foi designada para ele, terá acesso a todas as informações cadastradas pelo moderador referentes a esta inspeção. A aplicação se resume em revisar o artefato e responder os itens de avaliação com respostas Sim, Não ou Não se aplica.
- **Adicionar discrepâncias:** Na aplicação da inspeção, caso a resposta escolhida for diferente da esperada, aparecerá um campo para o preenchimento da discrepância, onde o inspetor escolhe uma categoria de defeitos, descreve a discrepância, o local da discrepância no artefato, e caso ela se repita, o local onde isso ocorre.
- **Exportar relatório de discrepância:** Possibilita o inspetor, caso todos os itens de avaliação de uma inspeção sejam respondidos, exportar o relatório de discrepâncias em um arquivo csv.

A imagem 11 apresenta alguns itens de avaliação sendo respondidos no momento da aplicação de uma inspeção.

Figura 11: Tela no momento da aplicação de uma inspeção.

É importante ressaltar que todos os valores contidos nas figuras são meramente ilustrativos e não correspondem a dados reais. Eles foram inseridos com o propósito de exemplificar os conceitos discutidos no artigo e facilitar a compreensão dos leitores. Dessa forma, os números apresentados nas figuras devem ser interpretados apenas como uma representação hipotética e não como informações precisas ou verificadas.

6. CONSIDERAÇÕES FINAIS

A ferramenta CASE EasyCheck, apresentada neste trabalho, visa destacar-se como uma contribuição importante no auxílio das inspeções de software, abordando uma necessidade crítica na engenharia de software: a identificação e correção precoce de defeitos. O objetivo principal da EasyCheck é proporcionar um suporte eficaz e sistemático durante a inspeção de artefatos de software, utilizando técnicas baseadas em checklists.

A utilização dessa ferramenta pode levar a uma melhora significativa na qualidade dos projetos de software. A ferramenta possui o potencial de facilitar a detecção precoce de defeitos, reduzindo os custos associados a correções tardias, podendo impactar na confiabilidade dos produtos entregues. Além disso, ao

sistematizar o processo de inspeção, a EasyCheck pode promover uma cultura de qualidade e de revisão contínua dentro das equipes de desenvolvimento.

Dessa forma, a EasyCheck apresenta-se podendo ser uma ferramenta útil e eficiente para a inspeção de software, sendo capaz de se adaptar às diversas necessidades de desenvolvedores e gestores de projetos. Sua implementação pode trazer benefícios consideráveis em termos de qualidade, eficiência e controle de processos, tornando-se um recurso valioso para organizações que buscam melhorar seus produtos de software. A ferramenta terá um registro INPI e, por enquanto, não possui um domínio próprio, mas será hospedada no Lens da UFRJ.

Uma breve apresentação da ferramenta foi gravada e está disponibilizada no endereço: https://youtu.be/0eSMieqqmqU

REFERÊNCIAS

(Britannica, 2023) Britannica, The Editors of Encyclopaedia. "computer-aided software engineering". Encyclopedia Britannica, 20 Mar. 2023, https://www.britannica.com/technology/computer-aided-software-engineering.

(Cerqueira, 2023) CERQUEIRA, Diego André; DE MELLO, Rafael Maiani; TRAVASSOS, Guilherme Horta. Um checklist para inspeção de privacidade e proteção de dados pessoais em artefatos de software. In: CONGRESSO IBERO-AMERICANO EM ENGENHARIA DE SOFTWARE (CIBSE), 26. , 2023, Montevideo, Uruguai. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2023 . p. 206-213. DOI: https://doi.org/10.5753/cibse.2023.24704.

(Fagan, 1976) Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. IBM Systems Journal, 15(3), 182–211. doi:10.1147/sj.153.0182.

(Kalinowski e Travassos, 2004) Kalinowski, M., & Travassos, G. H. (n.d.). ISPIS: a framework supporting software inspection processes. Proceedings. 19th International Conference on Automated Software Engineering, 2004. doi:10.1109/ase.2004.1342772

(Laitenberge e DeBaud, 2002) Laitenberge, O., and DeBaud, J. M. 2002. An Encompassing life cycle centric survey of Software Inspection. Journal of systems and software, 50, 5-31.

(Laitenberger *et al*., 2001) Laitenberger, O., El Anam, K., Harbich, T. G. An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective- Based Reading of Code Documents. IEEE Transactions on Software Engineering, vol. 27, No. 5, 2001.

(Melo *et al*., 2001) MELO, W., TRAVASSOS, G.H., SHULL, F., 2001, "Software Review Guidelines", Technical Report ES - 556101 - COPPEIUFRJ, August 2001.

(Mello, 2012) Mello, R. M. de, Teixeira, E. N., Schots, M., Werner, C. M. L., & Travassos, G. H. (2012). Checklist-Based Inspection Technique for Feature Models Review. 2012 Sixth Brazilian Symposium on Software Components, Architectures and Reuse. doi:10.1109/sbcars.2012.25

(Mello, 2011) Mello, R. M., Massollar, J. L., Travassos. G. H., "Técnica de Inspeção Baseada em Checklist para Identificação de Defeitos em Diagramas de Atividades". XX SBQS, 2011.

(Oladele, 2010) Oladele, R. O. 2010. Reading Techniques for Software Inspection: Review and Analysis. Journal of Institute of Mathematics and Computer Sciences (Computer Science Series), India, 21(2): 199 – 209

(Oladele e Adedayo 2014) OLADELE, R. O.; ADEDAYO, H. O. On empirical comparison of checklist-based reading and adhoc reading for code inspection. **International Journal of Computer Applications**, v. 87, n. 1, 2014.

(OMG, 2009) OMG (2009) "OMG Unified Modeling Language (OMG UML), Superstructure -Version 2.2", https://www.omg.org/spec/UML/2.2/Superstructure/PDF

(Shull, 2000) Shull, F., Rus, I., & Basili, V. (2000). How perspective-based reading can improve requirements inspections. Computer, 33(7), 73–79. doi:10.1109/2.869376

(Shull *et al*., 1999) Shull, F., Travassos, G. H., Carver, J., Basili, V. R. Evolving a Set of Techniques for OO Inspections. University of Maryland Technical Report CS-TR4070, Outubro de 1999.

(Travassos *et al*., 1999) Travassos, G. H., Shull, F., Carver, J. Evolving a Process for Inspecting OO Designs. XIII Simpósio Brasileiro de Engenharia de Software: Workshop de Qualidade de Software. Outubro, 1999.