Arthur Rocha Amaral e Guilherme Oliveira Antônio
{ arthur.amaral.1100245, gantonio } @sga.pucminas.br

Documento de Visão para o Sistema Covering

20 de fevereiro de 2022

Proposta dos alunos Arthur Rocha Amaral e Guilherme Oliveira Antônio ao curso de Engenharia de Software como projeto de Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo do professor José Laerte Xavier e orientação acadêmica do professor Marco Rodrigo Costa.

OBJETIVOS

O seguinte projeto visa auxiliar os desenvolvedores a produzirem códigos com padrões de qualidade, definidos por repositórios, de forma mais rápida. Assim, garante segurança e padronização das entregas e melhora a qualidade dos testes automatizados, afetando consecutivamente a manutenibilidade do software. Para isso, a solução tem como foco a facilitação do processo de verificação de métricas de testes, ainda no ambiente de desenvolvimento. Tais métricas são adotadas pela maioria dos *frameworks* de testes automatizados, para garantir uma melhor qualidade de código a partir do *feedback* gerado pelos testes. Assim, a ferramenta encurta o tempo entre análise de cobertura de testes e a manutenção dos trechos sem cobertura, agilizando o processo de desenvolvimento de novas *features* e de correções. Tendo por fim, como público alvo, desenvolvedores de diversos níveis da indústria de software.

ESCOPO

O sistema proposto é uma extensão para o Visual Studio Code (VSCode), o ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) mais utilizado no mundo. Visa facilitar o uso das métricas de coberturas de testes automatizados, trazendo esses dados para dentro do IDE e utilizando-os como parâmetros para configurar algumas ações da rotina de desenvolvimento. Além disso, critérios de bloqueio podem ser definidos para essas ações, com a possibilidade de serem configurados por repositório, definindo níveis de aceitação

de qualidade dos testes automatizados. Esses bloqueios também podem ser definidos para os processos que envolvem versionamento de código, garantindo a qualidade dos entregáveis, com tais critérios sendo observados ainda durante o desenvolvimento. Para isso o programa conta com:

- ler um arquivo de relatório de cobertura de linhas (LCOV[1], do inglês *line coverage*) gerado por *frameworks* de testes que será coletado em tempo real para exibição da situação atual do desenvolvimento;
- ler um arquivo de configuração na raiz do projeto para definir os critérios de aceitação do projeto e as variáveis necessárias para a configuração da extensão;
- exibir para os arquivos abertos nos editores de textos do VSCode, o estado de cobertura de cada linha;
- exibir o resumo de dados de cobertura na barra lateral do IDE;
- redirecionar automaticamente para uma linha de código sem cobertura dentre as linhas indicadas na barra lateral da IDE;
- gerar dados com base na definição de um *branch* de referência para análise do código;
- integrar com a Interface de Programação de Aplicação (API, do inglês *Application Programming Interface*) de controle de versionamento do VSCode chamado Source Control API, a qual executará o bloqueio ou aceitação do conteúdo desenvolvido, seguindo os parâmetros de qualidade definidos anteriormente.

O sistema proposto se assemelha com outras três aplicações, duas delas são extensões para o VSCode. Porém, oferecem apenas a visualização dos dados de cobertura. A primeira se chama Code Coverage[2] e aplicação possui somente duas das funcionalidades, as quais se assemelham com as propostas por essa aplicação, sendo elas:

- destaque das linhas cobertas no arquivo no editor;
- exibição da porcentagem de cobertura por arquivo.

Já a segunda extensão chamada Coverage Gutters[3] possui o objetivo único de destacar as linhas cobertas por testes nos arquivos abertos nos editores de textos do VSCode, diferindo do objetivo principal deste trabalho. As duas extensões citadas utilizam arquivos de LCOV para exibir os dados de cobertura. Assim, conseguem atender a qualquer linguagem que utiliza este tipo de arquivo para gerar os dados de cobertura. Já a aplicação proposta, apesar de também utilizar tal padrão de relatório de cobertura, tem somente como linguagens suportadas oficialmente o JavaScript[4] (JS) e o TypeScript[5] (TS).

[1] https://ltp.sourceforge.net/coverage/lcov/geninfo.1.php
[2] https://marketplace.visualstudio.com/items?itemName=markis.code-coverage
[3] https://marketplace.visualstudio.com/items?itemName=ryanluker.vscode-coverage-gutters
[4] https://www.javascript.com
[5] https://www.typescriptlang.org

A terceira aplicação que se aproxima da solução proposta é o SonarQube. Essa tem o foco na execução da análise estática de código, visando apontar as métricas de *bugs*, *code smells* e vulnerabilidades, exibindo também os dados de cobertura de código na aplicação. Porém, os escopos se afastam pelos seguintes motivos:

- o SonarQube é uma aplicação que precisa ser executada em um servidor, trazendo um gargalo no tempo de visualização dessas métricas;
- para acompanhamento das métricas analisadas pelo SonarQube ainda na IDE, existe uma extensão para o VSCode, o SonarQube Project Status. Contudo, essa extensão somente exibe as métricas na parte de segurança de código;
- a aplicação proposta neste trabalho, visa apontar as métricas de cobertura através de uma análise ainda no *client side*. Exibe os dados de resultado dos testes sem a necessidade de execução de uma instância dedicada da aplicação para análise do código e visualização dos dados. Trazendo principalmente um ganho de tempo para os desenvolvedores.

FORA DO ESCOPO

Algumas funções que poderiam ser atribuídas ao sistema, porém fogem do problema a ser resolvido, envolvem a geração do arquivo LCOV, o qual fica sobre a responsabilidade do *framework* de teste utilizado em cada projeto. Também foge do escopo propor ações específicas para melhorar a qualidade dos testes. Além disso, o sistema não interpretará dados de LCOV para projetos em outras linguagens diferentes de JavaScript e TypeScript. Por fim, fica definido que a aplicação é limitada a contexto de uma extensão do VSCode.

GESTORES, USUÁRIOS E OUTROS INTERESSADOS

Nome	José Laerte Xavier
Qualificação	Gestor/Orientador
Responsabilidades	Auxiliar no acompanhamento do desenvolvimento da plataforma e fornecer orientações acadêmicas.

Nome	Marco Rodrigo Costa
Qualificação	Gestor/Orientador
Responsabilidades	Auxiliar no acompanhamento do desenvolvimento da plataforma e

	fornecer orientações acadêmicas.

Nome	Arthur Rocha Amaral
Qualificação	Estudante/Desenvolvedor
Responsabilidades	Responsável pelo desenvolvimento da plataforma ao longo da disciplina de TCC I.

Nome	Guilherme Oliveira Antônio
Qualificação	Estudante/Desenvolvedor
Responsabilidades	Responsável pelo desenvolvimento da plataforma ao longo da disciplina de TCC I.

Qualificação	Desenvolvedor Líder
Responsabilidades	Atua como desenvolvedor líder de um time de 8 pessoas. Esse usuário define padrões de projeto com objetivo de tornar a manutenção e criação de *features* mais rápidas e mais seguras para o restante do time.
Como poderá usar o sistema proposto	Definindo variáveis de configuração para adicionar restrições ao processo de desenvolvimento, garantindo padrão e qualidade do código-fonte.

Qualificação	Desenvolvedor de Software Júnior
Responsabilidades	Como desenvolvedor em um time de 8 pessoas, o usuário entrega quase que diariamente *features* e correções do projeto em que trabalha. Ele segue alguns padrões propostos pelas lideranças técnicas da equipe como formatação do código e rotinas de execução de trabalho (por exemplo com testes de bancadas e validação de código em dupla).
Como poderá usar o sistema proposto	Como no projeto em que trabalha há restrições para garantir qualidade de código com base em cobertura de testes, o usuário precisará saber qual é o resultado atual da cobertura do código em que ele está trabalhando, com acesso facilitado aos trechos descobertos ou cobertos parcialmente.

LEVANTAMENTO DE NECESSIDADES

1. Visualização dos dados de cobertura de teste atual. Apesar de existir alguns outros métodos de visualizar a cobertura de testes, integrá-la ao IDE acelera drasticamente o processo de desenvolvimento dos testes.

2. Visualização dos dados de cobertura com base na diferença entre versões de código. Para os projetos que utilizam um versionamento de código Git[6] (sistema de controle de versões distribuído), as alterações do código-fonte são executadas em ramificações do código principal. Com isso, o processo distribui a responsabilidade do desenvolvimento entre os integrantes, os quais têm como foco entregarem um novo trecho de código, o qual deve ser uma extensão do código existente. Seguindo esse conceito é necessário analisar a diferença entre a ramificação e a versão principal, gerando métricas de cobertura somente do que foi desenvolvido na ramificação.

3. Definição de padrões de cobertura de teste para o projeto. Cada projeto deve ter seus critérios de avaliação, sendo necessário inserir alguns parâmetros para configuração da extensão. Dessa maneira, para que se adéque a cada contexto e possa executar bloqueios durante a ação de *push* em repositório Git.

4. Exibição de linhas de código sem cobertura. Ao exibir as linhas sem cobertura de código, facilita a localização de onde o desenvolvedor deve se empenhar para criar mais casos de testes. Além disso, garantir uma cobertura mais ampla por meio da navegação até os arquivos exibidos direcionando-os aos pontos exatos sem cobertura.

5. Controle da execução dos testes por meio de interface gráfica que independa da biblioteca utilizada. Para garantir que os dados sejam gerados para a análise da cobertura, é interessante que o controle da execução dos testes estejam disponíveis de forma prática, e configuradas por projeto apenas uma vez. Sendo um estímulo positivo para que o desenvolvedor execute mais vezes os casos de teste.

6. Disponibilização das necessidades acima no ambiente de desenvolvimento. O principal motivo desta aplicação ser uma extensão do VSCode é devido à disponibilidade de criação de funções que controlam a IDE e o diretório em que está aberto, entregando um ganho de produtividade aos desenvolvedores.

[6] https://git-scm.com

FUNCIONALIDADES DO PRODUTO

Necessidade: Visualização dos dados de cobertura de testes atual		
ID	**Funcionalidade**	**Categoria**
1	Localizar o arquivo de relatório de cobertura para interpretação de forma automática sem necessidade de receber o caminho para esse arquivo	Crítico
2	Exibir no editor o *status* de cobertura de cada linha	Crítico
3	Exibir a porcentagem de cobertura de código total na aba lateral	Crítico
4	Permitir a ativação/desativação da visualização do *status* de cada linha	Importante
5	Permitir a troca da visualização da porcentagem de cobertura total para a porcentagem de cobertura do arquivo aberto	Útil

Necessidade: Visualização dos dados de cobertura com base na diferença entre versões de código		
ID	**Funcionalidade**	**Categoria**
1	Permitir a troca da visualização da porcentagem de cobertura total para a porcentagem de cobertura referente às linhas criadas e alteradas na ramificação atual	Crítico
2	Exibir no editor apenas o *status* de cobertura das linhas criadas ou alteradas na ramificação atual	Crítico

Necessidade: Definição de padrões de cobertura de teste para o projeto		
ID	**Funcionalidade**	**Categoria**
1	Poder adicionar um arquivo de configuração na raiz do projeto	Importante
2	Permitir a definição de um valor mínimo de cobertura	Importante
3	Permitir a fixação da ramificação de referência para o projeto	Importante
4	Integração com aplicação de controle de versionamento (Git) para bloqueio da ação de *push* com base nos padrões definidos	Importante
5	Permitir a definição do caminho para o arquivo de relatório de cobertura	Útil

Necessidade: Exibição de linhas de código sem cobertura		
ID	**Funcionalidade**	**Categoria**
1	Na aba lateral exibir todas as linhas da aplicação que não foram cobertas	Importante
2	Quando o modo de visualização do status de cobertura estiver ativo com base na diferença das ramificações do versionamento de código, deve ser exibido apenas as linhas sem coberturas de testes que foram criadas ou alteradas na ramificação atual	Importante
3	Ao clicar em uma nas linhas listadas como descobertas, redireciona a janela do editor para o arquivo na exata linha selecionada	Útil

Necessidade: Controle da execução dos testes por meio de interface gráfica que independe da biblioteca utilizada		
ID	**Funcionalidade**	**Categoria**
1	Permitir a execução do comando definido no arquivo de configuração para execução dos testes pela interface da extensão	Útil
2	Permitir a definição de dois scripts no arquivo de configurações para executar os testes em modo de recarregamento automático após edição de algum arquivo.	Útil

Necessidade: Disponibilização das necessidades acima no ambiente de desenvolvimento		
ID	**Funcionalidade**	**Categoria**
1	Exibir na barra lateral do VSCode um item referente à extensão Covering	Crítico
2	Visualizar os dados dentro da aba da extensão no VSCode	Crítico
3	Configuração das opções do modo de visualização dentro da aba da extensão VSCode	Importante

INTERLIGAÇÃO COM OUTROS SISTEMAS

O sistema será interligado com o *Visual Studio Code* por meio de seu contexto de extensão. Também será necessária a integração com a API do *vscode.git* que fornece acesso ao repositório Git aberto no Visual Studio Code.

RESTRIÇÕES

Restrições de produto:

- Requisitos de eficiência:
    - O sistema deverá processar as alterações do arquivo de relatório de cobertura com um tempo limite de 2 segundos após a alteração do arquivo.
- Requisitos de confiabilidade:
    - O sistema deverá executar os cálculos com uma precisão de duas casas decimais em qualquer cálculo no sistema ou nas saídas dele.

Restrições organizacionais:

- Requisitos de implementação:
    - O sistema deverá ser desenvolvido na linguagem TypeScript.

Restrições externas:

- Requisitos de portabilidade
    - O sistema deverá ser executado em qualquer sistema operacional utilizando o VSCode
    - O sistema deverá ser executado na versão do VSCode 1.65 ou superior.
- Requisitos de interoperabilidade:
    - O sistema deverá se comunicar com a API Source Control do VSCode.

DOCUMENTAÇÃO

Como principal documentação do projeto será escrito um arquivo *readme* no repositório do projeto, o qual especificará como se dá a utilização da extensão. Com isso, é detalhado os requisitos mínimos a serem instalados e todos os parâmetros obrigatórios e opcionais para o funcionamento da extensão. O arquivo *readme* será apresentado dentro da página da extensão no *marketplace* do *Visual Studio Code*.

Como documentação arquitetural do projeto, será criado diagramas UML(*Unified Modeling Language*) como diagrama de componentes, diagrama de caso de uso e diagramas de sequência. Todos esses diagramas serão disponibilizados no repositório da aplicação e serão apresentados por meio de arquivos *markdown*.