**MÉTRICAS DE TESTE DE SOFTWARE**

1. **INTRODUÇÃO**

Para garantir de modo preciso que o produto tenha uma confiabilidade alta e seja estável, é necessário que se obtenha dados sobre como o produto está se comportando, a quantidade de falhas conhecidas e até mesmo se os testes do produto estão funcionando perfeitamente para que evite comportamentos inesperados e fora das regras de negócio do produto. Para isso, foram desenvolvidos meios que permitam obter métricas do software que são úteis para, de acordo com Trodo (2009), obter relatórios sobre a confiabilidade, estabilidade e desempenho do software.

Além disso, as métricas de software podem alterar completamente o planejamento e execução das atividades (Bourque; Fairley, 2014), tanto atrasando-as como diminuindo a quantidade de atividades. Supondo que haja uma aplicação já em produção e que obteve métricas que indicam uma possível falha gravíssima de segurança do sistema, é compreensível que todas as outras funcionalidades em desenvolvimento sejam postergadas e o foco seja compreender e resolver este erro de segurança. Pensando no mesmo cenário, se fossem obtidas novas métricas sobre a situação do produto com a indicação de que o problema de segurança já foi completamente resolvido, o time pode então retornar as atividades que estavam focados anteriormente.

Bourque e Fairley (2014) divide as métricas de teste em dois escopos: as métricas relacionadas ao programa que está sendo testado e as métricas que estão sendo aplicadas ao grupo de testes.

1. **MÉTRICAS RELACIONADAS AO PROGRAMA**

Estas avaliações do programa levam em conta as características do produto e as suas devidas ocorrências para que se elabore as métricas, podendo ser desde a quantidade de linhas de um código até a históricos de ocorrências do programa.

* 1. **Características gerais do código:**

De acordo com Bourque e Fairley (2014), as características básicas do software podem ser usadas para métricas de teste, desde o tamanho do código-fonte, a quantidade de chamadas de módulos, até como o projeto é estruturado. Essas características são importantes para compreender a complexidade do sistema e para definir novas métricas a partir dessas.

* 1. **Número de ocorrência:**

Essa métrica está relacionada a quantidade total de ocorrências que o produto já teve. Ocorrências são consideradas qualquer erro, inconsistências e futuros pontos de melhoria (Trodo, 2009).

A partir deste número, é possível compreender a confiabilidade do produto e entender a frequência de possíveis ocorrências no futuro.

* 1. **Status das ocorrências:**

Além da quantidade, também é importante compreender como essas ocorrências estão sendo gerenciadas. As ocorrências podem estar em três status:

* Pendentes de Solução: ainda não solucionadas;
* Para ser retestado: aguardando um novo teste;
* Encerrado: Soluções finalizadas e retestadas.

Compreender o número de ocorrências e a quantidade de cada uma em cada status pode ser útil para entender a situação atual do projeto e como o time lida com as ocorrências do projeto (Trodo, 2009).

* 1. **Classificação de possíveis falhas:**

A classificação de possíveis falhas envolve a criação de uma listagem de todas as falhas que o sistema pode apresentar entendendo as características dele. Além disso, é importante catalogar também as falhas que o sistema já apresentou e a frequência desses erros, pois isso indica que o sistema pode ter uma tendência a cometer as mesmas falhas. Essa métrica pode ser essencial para previsão de erros que podem surgir ou ressurgir em futuras *features* (Bourque; Fairley, 2014; Trodo, 2009).

* 1. **Índice de densidade de defeitos:**

O índice de densidade de defeitos é obtido através da quantidade de defeitos em razão do tamanho do programa (Bourque; Fairley, 2014). De acordo com Trodo (2009), os defeitos são considerados ocorrências que em algum momento foram o motivo de alterações tanto do *software* como da documentação, tanto por questão de falhas como para possíveis *melhorias*.

* 1. **Índice de severidade de defeitos:**

A severidade dos defeitos indica a prioridade que cada defeito reconhecido tem no sistema. Essas prioridades são definidas em níveis, sendo geralmente 4: Crítico (Nível 4), Sério (Nível 3), Médio (Nível 2) e Baixo (Nível 1); assim, para obter o índice deve multiplicar cada ocorrência pelo seu nível e dividir pela quantidade de defeitos (Trodo, 2009).

* 1. **Tempo médio para encontrar defeito:**

Essa métrica é utilizada para compreender a duração média que se gasta para identificar um defeito num sistema. É calculado a partir das horas gastas de execução de testes em razão dos defeitos detectados (Trodo, 2009).

* 1. **Tempo para resolver defeito:**

Semelhante a métrica anterior, esta calcula a duração de resolver um problema identificado, sendo útil para definir o custo de manutenção do sistema (Trodo, 2009).

* 1. **Avaliação de confiabilidade**

A avaliação de confiabilidade ocorre por meio de uma estimativa estatística que mede a probabilidade de o sistema funcionar sem a apresentação de falhas em condições específicas. Essa métrica é de suma importância pois pode alterar o planejamento e o ciclo de vida do software; uma avaliação negativa pode fazer com que o software retroceda a etapas anteriores e uma avaliação positiva pode encerrar o período de teste e passar para a próxima etapa do ciclo de vida (Bourque; Fairley, 2014).

* 1. **Modelos de crescimento da confiabilidade:**

A partir do momento que se estabelece a avaliação da confiabilidade, é possível determinar os modelos de crescimento de confiabilidade, que são predições sobre a confiabilidade do *software.* Estas predições tendem a aumentar a sua confiabilidade conforme as falhas são identificadas e corrigidas.

1. **MÉTRICAS RELACIONADAS AO GRUPO DE TESTES**
   1. **Cobertura de testes:**

Essa métrica se refere a abrangência da cobertura de testes em relaçãoaos elementos do software. Definir uma medida de cobertura de testes pode ser complexo já que não existe uma quantidade exata de testes que o sistema precisa ter para ser considerado completamente coberto e então, para que se obtenha essa medida, antes é necessário montar uma coleção de testes necessários que o sistema precisaria ter, e, a partir disso, verificar a aplicação dos testes que o produto tem em relação aos testes considerados necessários. (Trodo, 2009; Bourque; Fairley, 2014).

Além disso, Bourque e Fairley (2014) afirma que a cobertura de testes não deve somente abranger todos os elementos do *software,* mas também cobrir minuciosamente cada um desses elementos.

* 1. **Efetividade de caso de teste:**

A efetividade de caso de teste pode ser obtida através de uma verificação para perceber se os casos de testes realmente encontram defeitos. O cálculo é feito através da relação de número de casos de teste que se tornaram ocorrências e o número total de testes (Trodo, 2009).

* 1. ***Fault seeding*:**

Esta métrica é obtida através de um processo conhecido como *Fault Seeding* (Injeção de falhas), onde propositalmente são injetadas falhas no sistema a fim de analisar se os testes conseguem identificar os erros injetados. Nesse processo, também pode ser possível que verdadeiras falhas do projeto sejam encontradas.

Por mais que seja uma técnica útil, deve ser usada com cuidado para que os erros propositais não acabem sendo deixados no código, o que pode ser resolvido com versionamento do código. (Bourque; Fairley, 2014).

* 1. **Pontuação de mutação:**

A pontuação de mutação se assemelha um pouco com o *Fault Seeding*, a diferença é que nela são criadas diversas cópias com mutações do código e o objetivo é que o teste consiga identificar e eliminar estas mutações criadas (conhecidos como mutações mortas). A pontuação então se dá pela quantidade de mutações mortas em razão a quantidade total de mutações geradas (Bourque; Fairley, 2014).

**FERRAMENTAS DE TESTES DE SOFTWARE**

O processo de testar programas envolve muitas tarefas, execução de programas e o processamento de diversas informações. Criar testes manualmente sem nenhum auxílio de ferramentas pode ser uma prática trabalhosa e que pode abrir brechas para possíveis erros graças ao aumento de linhas de código. A fim de simplificar este processo, aumentar a produtividade e garantir testes mais sólidos, muitas ferramentas foram criadas para automatizar e auxiliar a criação de testes (Bourque; Fairley, 2014; Vicente, 2010).

A primeira etapa e considerada muito crucial é a seleção das ferramentas que serão utilizadas no processo de testes; para isso, é essencial compreender os tipos de ferramentas de teste, como elas agem e participam do teste.

1. **CATEGORIAS DE FERRAMENTAS**
2. **Ferramentas de *Test Harness*:**

As ferramentas dessa categoria criam um ambiente controlado de teste que simulam uma entrada, a comunicação com outros módulos e registram os resultados obtidos para análise (Bourque; Fairley, 2014).

O Test Harness consiste em dois elementos: *drivers* e *stubs,* que interagem com o componente sob teste (*Component Under Test* – CUT) a fim de simular comportamentos do usuário e do servidor e garantir que um componente seja executado isoladamente. (Bourque; Fairley, 2014; Rocha; Martins, 2008).

Os drivers simulam o comportamento do usuário, inicializando a execução de testes com entradas específicas, interagindo diretamente com o CUT e registrando os resultados. Além disso, os drivers verificam se os objetivos de teste foram realmente alcançados e reportam caso haja alguma falha. (Rocha; Martins, 2008).

Como muitas vezes um CUT depende de outros componentes, acaba se tornando essencial ter também substitutos para esses componentes a fim de se realizar os testes com maior agilidade (já que em um substituto, não é realizado nenhuma operação de fato, mas sim uma interface que simula os componentes) e de maneira mais isolada. Esses substitutos são os stubs, que fornecem entradas indiretas que simulariam o comportamento dos outros componentes do servidor. (Rocha; Martins, 2008).

1. **Geradores de teste:**

Essas ferramentas são capazes de criar os casos de teste seguindo diversas estratégias: de maneira aleatória (que pode ser interessante por analisar tópicos que não foram planejados), baseados nos caminhos que o *software* pode seguir a partir de diversas formas de entrada, com base no modelo do sistema ou na junção de todos esses (Bourque; Fairley, 2014).

1. **Ferramentas de captura / replay:**

As ferramentas de captura e replay reproduzem automaticamente testes ou interações reais de usuários que já foram executados com entradas e saída definidas. Assim, é possível verificar se as saídas do programa ainda permanecem as mesmas (Bourque; Fairley, 2014).

1. **Analisadores de cobertura e instrumentadores:**

Os analisadores e instrumentadores são ferramentas diferentes, mas que trabalham em conjunto. Os analisadores de cobertura verificam quais partes do código (funções, linhas, etc) foram realmente cobertas pelo teste. Este processo somente é permitido a partir do uso dos instrumentadores, que modificam o código a fim de permitir que seja registrado quais partes do código foram cobertos (Bourque; Fairley, 2014).

Vicente (2010) indica a alta frequência em que ferramentas de cobertura são utilizadas em metodologias ágeis e que muitas delas podem ser integradas com outras ferramentas de desenvolvedor, como o Eclipse e outras IDEs. Estas ferramentas também podem ser usada para definir a métrica de cobertura de testes.

1. **Rastreadores (*tracers*):**

As ferramentas do tipo “rastreadores” são capazes de analisar e manter um histórico de quais caminhos do programa foram executados, podendo ser úteis em cenários de depuração e monitoramento de desempenho (Bourque; Fairley, 2014).

1. **Ferramentas de teste de regressão**

As ferramentas de teste de regressão são usadas para garantir que o programa não tenha novos problemas após uma alteração no código. Para isso, ela permite que o conjunto de testes (ou parte dele) já criados possam ser reexecutados quando ocorre uma nova alteração (Bourque; Fairley, 2014).

1. Ferramentas de avaliação de confiabilidade

As ferramentas de avaliação de confiabilidade analisam e exibem os resultados dos testes de maneira visual a fim de facilitar a avaliação das medidas relacionadas à confiabilidade.

1. GRUPO DE FERRAMENTA DE TESTES

De acordo com o Bourque e Fairley (2014), o recomendável não é escolher somente uma, mas sim um conjunto de ferramentas que participarão de formas diferentes do processo de teste, pois cada ferramenta engloba um período, parte ou característica diferente a ser testada. Podemos usar num mesmo produto por exemplo, ferramentas de *test harness,* geração de testes e ferramentas de captura e replay para descoberta de falhas, as ferramentas de testes de regressão para reutilização de testes conforme surgem novas melhorias e as ferramentas de avaliação de confiabilidade para a exibição dessas falhas e da confiabilidade do produto.

**REFERÊNCIAS**

BOURQUE Pierre, FAIRLEY Richard. **Software Engineering Body of Knowledge (SWEBOK):** Software engineering body of knowledge. 3. ed. IEEE Computer Society (2014). Acesso em out. de 2024.

ROCHA, C.R., MARTINS, E. **A method for model based test harness generation for component testing.** J Braz Comp Soc 14, 7–23 (2008). <https://doi.org/10.1007/BF03192549>

TRODO, Lia Degrazia. Uso de métricas nos testes de software. 2009. Acesso em nov. de 2024.

VICENTE, André Abe. **Definição e gerenciamento de métricas de teste no contexto de métodos ágeis.** 2010. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, University of São Paulo, São Carlos, 2010. doi:10.11606/D.55.2010.tde-23062010-083439. Acesso em: nov de 2024.