Introdução a Teste de Software

Herysson R. Figueiredo
herysson.figueiredo@ufn.edu.br

Introdução a teste de *software*

Nesta aula teremos a introdução a “Testes de Software” e sendo uma aula introdutório trataremos da base teórica, assim como os termos relacionados ao assunto. Queremos que você entenda a importância de se testar o software durante todo desenvolvimento e saiba o que é um teste.

Objetivos

- Estudar sobre qualidade de software;
- Entender garantia de qualidade;
  - Processo,
  - Elementos.
- Conhecer as atividades de verificação e validação (V&V);
- Definir teste de software
  - Caso de teste,
  - Cenário de teste,
  - Defeito, Erro, Falha e Engano;

Qualidade de *software*

“Você conhece um software de qualidade?

“No que você se baseou para julgar a qualidade do software? O que é qualidade de software? Como será que se faz para atingir a qualidade de um software?”.

Qualidade de *software*

"Qualidade de software é a conformidade com requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo software desenvolvido profissionalmente"

Pressman (PRESSMAN, 2006)

Qualidade de *software*

Sempre que falamos de Qualidade de *Software* devemos lembrar nas da **garantia de qualidade de software – SQA (*Software Quality Assurance*)**. Que pode ser definida como um padrão de ações planejado e sistematizado necessário para garantir alta qualidade no software

Pressman (PRESSMAN, 2021)

SQA (*Software Quality Assurance*)

Sempre que falamos de Qualidade de *Software* devemos lembrar nas da **garantia de qualidade de software – SQA (*Software Quality Assurance*)**. Que pode ser definida como um padrão de ações planejado e sistematizado necessário para garantir alta qualidade no software

Pressman (PRESSMAN, 2021)

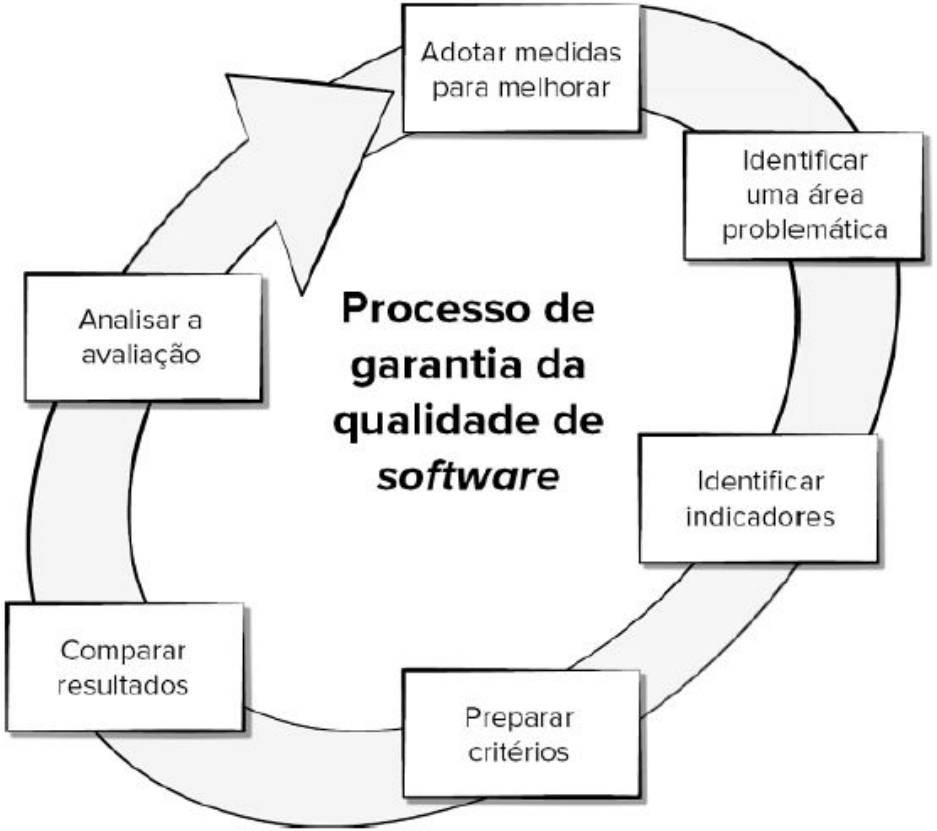
SQA (*Software Quality Assurance*)

SQA (*Software Quality Assurance*)

A SQA engloba um amplo espectro de atividades e preocupações, incluindo:

- Um processo de SQA
- Tarefas específicas de garantia e controle de qualidade, como revisões técnicas e uma estratégia de testes em múltiplas camadas.
- A prática efetiva de engenharia de software (métodos e ferramentas).
- Um procedimento para garantir a conformidade com os padrões de desenvolvimento de software.
- Mecanismos de medição e de geração de relatórios.

Processo da SQA:



Elementos da SQA (*Software Quality Assurance*)

Padrões: A SQA garante que os padrões adotados sejam seguidos e que os produtos resultantes estejam em conformidade com eles.

Revisões e auditorias: As revisões técnicas são realizadas para revelar erros, enquanto as auditorias são conduzidas pelo pessoal de SQA para assegurar que as diretrizes de qualidade estão sendo seguidas.

Teste: A SQA usa o teste como uma das atividades mais importantes para avaliar a qualidade.

Elementos da SQA (*Software Quality Assurance*)

Coleta e análise de erros/defeitos: A SQA coleta e analisa dados sobre erros e defeitos para entender melhor como eles são introduzidos e como podem ser eliminados.

Gerenciamento de mudanças: A SQA garante que práticas adequadas de gerenciamento de mudanças sejam instituídas para evitar a confusão que leva a uma qualidade inadequada.

Educação: A SQA assume a liderança na melhoria do processo de software e patrocina programas educacionais para engenheiros e gerentes.

Elementos da SQA (*Software Quality Assurance*)

Gerência dos fornecedores: A SQA é responsável por garantir que o software adquirido de fornecedores externos atenda aos padrões de qualidade.

Gestão de segurança: A SQA garante o emprego de processos e tecnologias apropriados para a segurança do software.

Proteção: A SQA pode ser responsável por avaliar o impacto de falhas de software e iniciar etapas para a redução de riscos.

Gestão de riscos: O grupo de SQA garante que as atividades de gestão de riscos sejam conduzidas apropriadamente e que planos de contingência tenham sido estabelecidos.

Execução da SQA

A SQA é composta por tarefas associadas a dois grupos distintos:

Engenheiros de software: Realizam o trabalho técnico.

Grupo de SQA: Um grupo que é responsável pelo planejamento, supervisão, manutenção de registros, análise e relatórios referentes à garantia da qualidade. A principal função do grupo de SQA é ajudar a equipe de software a obter um produto final de alta qualidade.

Metas, atributos e métricas

As atividades de Garantia da Qualidade de Software (SQA) são realizadas para atingir um conjunto de **metas**. Para cada meta, são identificados **atributos** que indicam a existência de qualidade, e são utilizadas **métricas** para medir cada um desses atributos.

Metas, atributos e métricas

As metas pragmáticas da SQA são:

Qualidade dos requisitos: O objetivo é garantir a correção, a completude e a consistência do modelo de requisitos, pois isso terá um forte impacto na qualidade do produto final.

Qualidade do projeto: Todo elemento do modelo de projeto deve ser avaliado para garantir que apresente alta qualidade e que o projeto esteja de acordo com os requisitos. A SQA busca por atributos de projeto que sejam indicadores de qualidade.

Metas, atributos e métricas

As metas pragmáticas da SQA são:

Qualidade do código: O código-fonte e outros artefatos relacionados devem estar em conformidade com os padrões de codificação locais e apresentar características que facilitem a manutenção.

Eficácia do controle de qualidade: A equipe de software deve usar seus recursos limitados da forma mais eficaz possível para atingir um resultado de alta qualidade.

Metas, atributos e métricas para qualidade de software
Fonte: Adaptado de (Hya96)

Meta	Atributo	Métrica
Qualidade das necessidades	Ambiguidade	Número de modificadores ambíguos (por exemplo, muitos, grande, amigável)
	Completude	Número de TBA, TBD
	Compreensibilidade	Número de seções/subseções
	Volatilidade	Número de mudanças por requisito
		Tempo (por atividade) quando é solicitada a mudança
	Facilidade de atribuição	Número de requisitos não atribuíveis ao projeto/código
	Clareza do modelo	Número de modelos UML
		Número de páginas descritivas por modelo
		Número de erros UML

Meta	Atributo	Métrica
Qualidade do projeto	Integridade da arquitetura	Existência do modelo da arquitetura
	Completude dos componentes	Número de componentes que se atribui ao modelo da arquitetura
		Complexidade do projeto procedural
	Complexidade da interface	Número médio de cliques para chegar a uma função ou conteúdo típico
		Apropriabilidade do layout
	Padrões	Número de padrões usados
Qualidade do código	Complexidade	Complexidade ciclométrica
	Facilidade de manutenção	Fatores de projeto
	Compreensibilidade	Porcentagem de comentários internos
		Convenções de atribuição de variáveis
	Reusabilidade	Porcentagem de componentes reutilizados
	Documentação	Índice de legibilidade

Metas, atributos e métricas

Meta	Atributo	Métrica
Eficiência do controle de qualidade	Alocação de recursos	Porcentagem de horas de pessoal por atividade
	Taxa de completude	Tempo de finalização real *versus* previsto
	Eficácia da revisão	Ver métricas de revisão
	Eficácia dos testes	Número de erros encontrados e criticalidade
		Esforço exigido para corrigir um erro
		Origem do erro

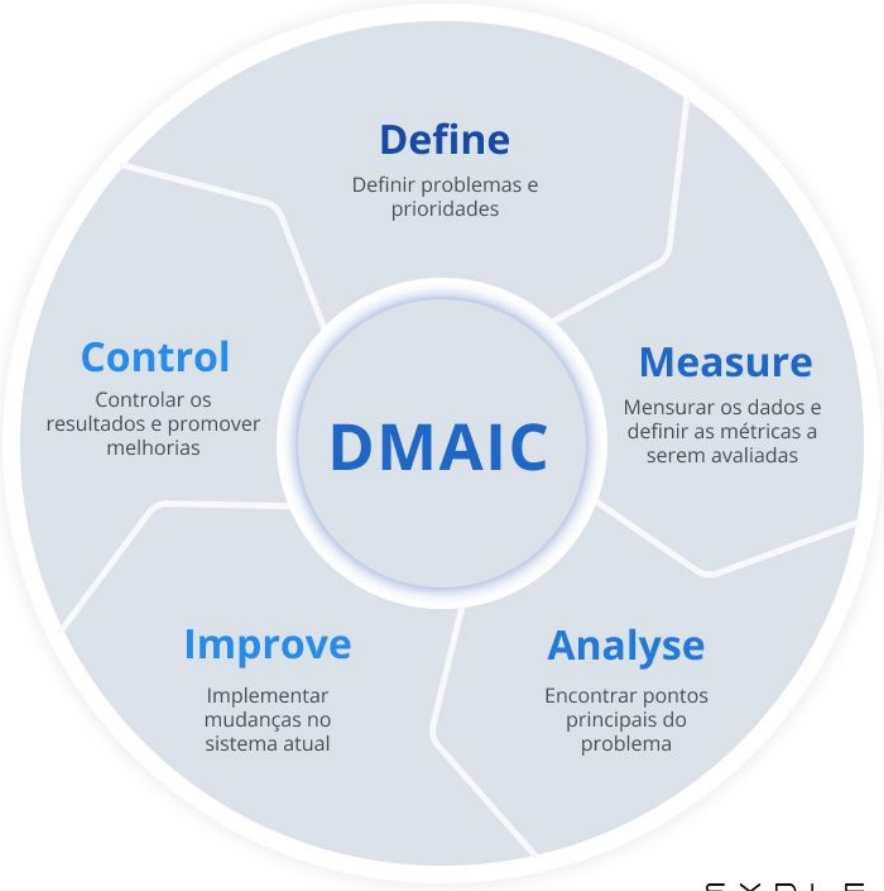
Metas, atributos e métricas

Meta	Atributo	Métrica
Eficiência do controle de qualidade	Alocação de recursos	Porcentagem de horas de pessoal por atividade
	Taxa de completude	Tempo de finalização real *versus* previsto
	Eficácia da revisão	Ver métricas de revisão
	Eficácia dos testes	Número de erros encontrados e criticalidade
		Esforço exigido para corrigir um erro
		Origem do erro

Seis sigma para engenharia de software

Seis Sigma é a estratégia para a estatística da garantia da qualidade mais utilizada na indústria atual. Originalmente popularizada pela Motorola na década de 1980, a estratégia Seis Sigma

“é uma metodologia rigorosa e disciplinada que usa análise estatística e de dados para medir e melhorar o desempenho operacional de uma empresa através da identificação e da eliminação de defeitos em processos de fabricação e relacionados a serviços”



Fonte: Sydle (2022)

Metodologia Seis Sigma

Definir: Definir os requisitos do cliente, os artefatos a serem entregues e as metas do projeto por meio de métodos de comunicação bem definidos com o cliente.

Medir: Medir o processo existente e seu resultado para determinar o desempenho atual da qualidade, coletando métricas de defeitos.

Metodologia Seis Sigma

Analisar: Analisar as métricas de defeitos para determinar as "poucas causas vitais" (as principais causas dos problemas).

Melhorar (Improve): Melhorar o processo eliminando as causas básicas dos defeitos.

Controlar: Controlar o processo para garantir que trabalhos futuros não reintroduzam as causas dos defeitos.

O Papel da Verificação e Validação (V&V) na SQA

VERIFICAÇÃO	Atividades para responder à pergunta: "*Estamos construindo o produto corretamente?*";
VALIDAÇÃO	Atividades para responder à pergunta: "*Estamos construindo o produto certo?*".

O Papel da Verificação e Validação (V&V) na SQA

Verificação (**Estou construindo o produto da forma correta?**): A verificação foca em garantir que os produtos de uma determinada fase do desenvolvimento estejam em conformidade com as condições e padrões estabelecidos no início daquela fase.

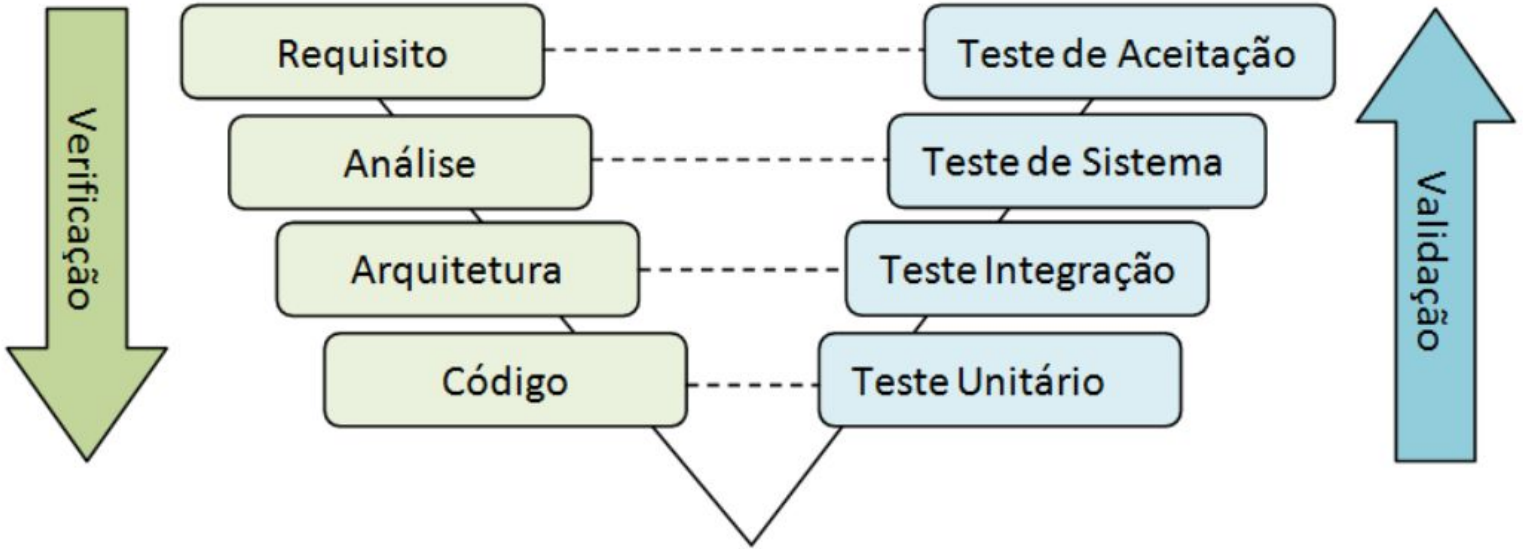
- Revisões
- Padrões
- Auditorias
- Qualidade do Projeto e do Código

O Papel da Verificação e Validação (V&V) na SQA

Validação (Estou construindo o produto certo?): A validação é o processo de avaliar o software ao final do desenvolvimento para garantir que ele atende às necessidades e requisitos do cliente.

- Testes de Software

Qualidade de *software*



Qualidade de *software*

Além de classificadas como verificação e/ou validação, as atividades V&V são classificadas como estáticas e dinâmicas:

- Atividades estáticas são aquelas que **não dependem** de um artefato executável para serem realizadas. Em outras palavras, não é necessário ter um código fonte para realizar atividades de V&V estáticas.
- Atividades dinâmicas são aquelas que **dependem** de um artefato executável para serem realizadas. Onde é necessário ter um código fonte executável para realizar atividades de V&V dinâmicas.

Artefato de Teste: é a especificação de um conjunto de entradas de **teste**, condições de execução e resultados esperados, identificados com a finalidade de fazer a avaliação de algum aspecto particular de um cenário.

Qualidade de *software*

As atividades de V&V são tipicamente executadas usando uma ou mais das quatro técnicas listadas a seguir:

ANÁLISE	Que é o uso de técnicas ou modelos matemáticos, simulações algoritmos ou procedimentos científicos para determinar se o produto ou artefato está em conformidade com seus requisitos.

Qualidade de *software*

As atividades de V&V são tipicamente executadas usando uma ou mais das quatro técnicas listadas a seguir:

DEMONSTRAÇÃO	É um exame visual do produto de software sendo executado em um cenário específico a fim de identificar se ele está em conformidade com os requisitos. As famosas "demos" são um exemplo dessas demonstrações.

Qualidade de *software*

As atividades de V&V são tipicamente executadas usando uma ou mais das quatro técnicas listadas a seguir:

INSPEÇÃO	É uma examinação visual (comumente envolvendo a manipulação manual dos artefatos) de um artefato não executável, novamente para identificar a conformidade do artefato com seus requisitos. Por sua característica comumente estática, Sommerville (SOMMERVILLE, 2011) salienta que técnicas de análise, como as revisões formais e inspeção, oferece garantia somente entre a correspondência de um artefato com a sua especificação, não sendo adequada para identificar a utilidade do software para o cliente.

Qualidade de *software*

As atividades de V&V são tipicamente executadas usando uma ou mais das quatro técnicas listadas a seguir:

TESTE	É a simulação de um artefato executável com entradas e pré-condições conhecidas e a comparação entre a saída atual e a saída esperada para determinar se o artefato está em conformidade com os requisitos.

Teste de *software*

Teste de Software é a técnica mais fácil de se executar, mesmo em ambientes com processo de desenvolvimento inadequado, no qual não há documentação alguma, há um executável e, mesmo que de forma não adequada, a atividade de teste pode ser aplicada.

“Teste de *software* é o último reduto no qual a qualidade do software pode ser avaliada e defeitos podem ser identificados antes de serem entregues ao usuário final.” (PRESSMAN, 2006)

Teste de *software*

"*Teste é a simulação de um artefato executável com entradas e pré-condições conhecidas e a comparação entre a saída atual e a saída esperada para determinar se o artefato está em conformidade com os requisitos*" (FIRESMITH, 2013).

"Teste é o processo de executar um sistema ou componente sob condições específicas, observando e registrando os resultados, avaliando alguns aspectos do sistema ou componente" (ISO/IEC/IEEE 24765, 2010).

"Teste é o processo de executar um programa ou sistema com a intenção de encontrar erros" (MYERS, 2004).

Teste de *software*

Teste de software – a mentalidade é: se existia algum defeito nas funcionalidades que eu executei durante os testes (ou no código-fonte executado), defeitos esses alcançados pela forma como eu usei o sistema (dados, sequência de execução etc.), esse defeito foi encontrado, reportado e corrigido.

“Uma solução para não ter problema no software é testar todas as possibilidades.”

Dessa forma, nós podemos garantir que todas as funcionalidades foram executadas de todas as formas possíveis e como consequência, todos os defeitos foram encontrados. Isso é o que chamamos de teste exaustivo.

Quanto tempo levaria para ser executado um teste exaustivo?

Teste de *software*

```
1 int exemplo_simples(int j) {
2    j = j - 1;
3    j = j / 30000;
4    return j;
5 }
```

Teste de *software*

No exemplo anterior seria impossível realizar testes por exaustão em um tempo em um tempo plausível. Além da impossibilidade de testar por exaustão, ainda temos o impacto da escolha dos dados corretos, pois um conjunto de dados pode fazer com que o defeito seja identificado, e outro conjunto de dados pode fazer com que o defeito nunca seja identificado.

Teste de *software*

```
1 int exemplo_simples(int j) {
2    j = j - 1;
3    j = j / 30000;
4    return j;
5 }
```

Teste de *software*

Considerando a impossibilidade de testar por exaustão, nós devemos aplicar os critérios de teste, que nos ajudam, entre outras coisas, a definir melhor nossos **casos de testes** e os dados de entrada a fim de identificar defeito.

Casos de teste

De acordo com a norma IEEE 829-2008 (IEEE 829-2008, 2008), **caso de teste** é um conjunto de dados de entradas, condições de execução do sistema e o resultado desenvolvido/projetado para atingir um objetivo específico, como exercitar um caminho específico em um programa ou verificar a concordância do programa com o requisito.

Casos de teste

Casos de teste é uma sequência de passos que devem ser executados no sistema, sendo que os dados de entrada e saída esperada para cada passo são especificados. Os casos de teste devem “direcionar” as ações do testador em uma determinada funcionalidade e fazer com que ele observe se o resultado que ele obtém em cada passo é o resultado esperado, de acordo com os requisitos

Casos de teste

Segundo norma (IEEE 829-2008, 2008) menciona que um **caso de teste** deve conter:

1. IDENTIFICADOR DO CASO DE TESTE	Número único de identificação do caso de teste em questão.
2. OBJETIVO DO CASO DE TESTE	Mostra de forma breve qual é a intenção de teste do caso de teste.
3. ENTRADAS	Todas as entradas que serão consideradas no caso de teste.

Casos de teste

Segundo norma (IEEE 829-2008, 2008) menciona que um **caso de teste** deve conter:

4. SAÍDAS	As saídas esperadas de acordo com as entradas indicadas na execução dos testes.
5. AMBIENTES NECESSÁRIOS OU CONDIÇÕES DE AMBIENTE	Quais ambientes de *hardware e software*, inclusive com versionamentos, que devem ser montados para o caso de teste em questão.

Casos de teste

Segundo norma (IEEE 829-2008, 2008) menciona que um **caso de teste** deve conter:

6. PROCEDIMENTOS ESPECIAIS REQUISITADOS PARA EXECUÇÃO DO CASO DE TESTE	Como o analista de teste deve proceder para a execução do caso de teste em questão. Deve especificar com detalhes o que deve ser testado e a forma que o teste deve ser executado.
7 DEPENDÊNCIA COM OUTROS CASOS DE TESTE	No caso deste caso de teste estar relacionado com outro caso de teste, então, especificar este relacionamento aqui.

Casos de teste

Pensando no exemplo, um caso de teste possível seria:

1. Caso de teste 1: entrada negativa
2. Verificar se entradas negativas são possíveis
3. (E) Digite “- 30000”
4. (S) 0,99
5. Nenhum
6. Nenhum
7. Não

Com esse caso de teste, o testador iria verificar que há um defeito, pois certamente o programa iria falhar em não retornar o resultado esperado.

Teste de *software*

```
1 int exemplo_simples(int j) {
2    j = j - 1;
3    j = j / 30000;
4    return j;
5 }
```

Casos de teste

O que fazer quando o programa falha e não retorna o resultado esperado?

Ao identificar um defeito, o testador deve reportar o defeito de forma a permitir que o time de desenvolvimento identifique facilmente como eliminar o defeito do sistema.

A forma de reportar defeitos depende muito de padrões definidos em cada empresa, ou time de desenvolvimento, e das ferramentas utilizadas para gerenciar o projeto e a atividade de teste, como por exemplo, Jira e Zephyr da Atlassian , Bugzilla ou testLink .

Reportar defeitos

Uma forma de como reportar um defeito é (ISTQB, 2016)

- **Identificador do defeito:** todo defeito deve ter um identificador único para facilitar a comunicação e localização;
- **Descrição do defeito:** uma descrição sucinta do defeito encontrado;
- **Versão do produto:** para saber em qual versão do sistema o defeito foi encontrado;
- **Passos detalhados:** descrição de passos realizados no sistema, incluindo os dados utilizados para encontrar o defeito. *Screenshots* e vídeos podem ser muito úteis também. A ideia é permitir que os desenvolvedores consigam reproduzir o defeito em ambiente de depuração;

International Software Testing Qualifications Board (ISTQB)

Reportar defeitos

Uma forma de como reportar um defeito é (ISTQB, 2016)

- **Data de reporte do defeito:** para facilitar o gerenciamento;
- **Reportado por:** para saber qual o testador identificou o defeito;
- ***Status:*** aqui, diferentes nomes podem ser aplicados, mas o objetivo é permitir que todos os envolvidos saibam se o defeito já foi endereçado, analisado, corrigido, se tem algum desenvolvedor encarregado de arrumar o programa, se o testador já viu a resolução e assim por diante. Um exemplo de lista de possíveis status é: novo/aberto, designado, em verificação, resolvido, fechado, falhado, não será corrigido etc

Reportar defeitos

Uma forma de como reportar um defeito é (ISTQB, 2016)

- **Corrigido por:** Nome do desenvolvedor que corrigiu o defeito;
- **Data de encerramento:** a data que o defeito foi dado como inexistente;
- **Severidade:** para informar quão grave é o defeito no sistema, como por exemplo, bloqueia a versão, crítico, pequeno;
- **Prioridade:** para definir a prioridade em corrigir o defeito: alta, média ou baixa

Cenário de Testes

De acordo com a norma IEEE 829-2008 (IEEE 829-2008, 2008), cenário de teste é um conjunto de casos de teste relacionados, que comumente testam o mesmo componente ou a mesma funcionalidade do sistema.

Cenários de caso de teste também chamados de suítes ou ciclos de teste.

Cenário de Testes

Ao definir esse agrupamento de casos de teste, temos que pensar que eles serão extremamente úteis para testes de regressão (que são testes feitos na fase de manutenção do sistema para garantir que as novas funcionalidades não inseriram defeitos nas antigas), para revalidar mudanças nos casos de teste e também para treinar novos testadores e pessoas que darão suporte ao sistema.

Cenários de caso de teste também chamados de suítes ou ciclos de teste.

Exemplo de caso de teste - Autenticação (*login)*

1. Caso de teste 1 - *Login*
2. Verificar *login* inválido
3. Entradas (E)
4. Saídas (S)
    a. (E) No celular, clique no ícone do aplicativo.
    b. (S) O aplicativo deve abrir e exibir a tela de *login*.
    c. (E) Insira um e-mail inválido, sem o sinal de @ (professor.gmail. com) e clique em "*next*".
    d. (S) O aplicativo deve deixar a borda do campo *login* vermelho e a mensagem "Por favor, digite um *login* válido" deve ser exibida.
    e. (E) Clicar em OK na mensagem.
    f. (S) O cursor de digitação deve estar no campo de *login* e o usuário deve poder corrigir o login anterior.
5. Smartphone com o aplicativo instalado
6. Nenhum
7. Não (afinal, não preciso ter feito nenhuma ação antes para testar o login)

Engano, defeito, erro e falha

Segundo a definição da norma ISO/IEC/IEEE 24765:2010 :

ENGANO (*MISTAKE*)	Uma ação humana que produz um resultado incorreto, como uma codificação ou modelagem errada;
DEFEITO (*FAULT*)	Uma imperfeição ou deficiência em um artefato que faz com que este não esteja em conformidade com os requisitos ou especificações, sendo necessária sua correção ou substituição. Termos como "erro" e "*bug*" comumente são usados para expressar defeitos;

Defeito, erro e falha

Segundo a definição da norma ISO/IEC/IEEE 24765:2010 :

ERRO (*ERROR*)	Resultado incoerente produzido por meio de uma ação no sistema;
FALHA (*FAILURE*)	Incapacidade do *software* exercer a função para a qual foi desenvolvido.

Defeito, erro e falha

Sendo assim, engano introduz um defeito que quando é exercitado, idealmente por meio da execução de um caso de teste, produz um erro no sistema e esse erro, consequentemente, faz com que o sistema falhe.

Atividades

Desenvolver dinâmica proposta.

Referências

PRESSMAN, Roger S. Engenharia de Software. Mc Graw Hill, 6 ed, Porto Alegre, 2006.

PRESSMAN, Roger S. Engenharia de Software. Mc Graw Hill, 6 ed, Porto Alegre, 2021.

BRAGA, Pedro Henrique Cacique. Teste de Software. Pearson Education do Brasil. São Paulo. 2016. Disponível na Biblioteca Virtual.

DELAMARO, Márcio; MALDONADO, José Carlos, JINO, Mario. Introdução ao teste de software. Elsevier. Rio de Janeiro. 2007.

PEZZÉ, Mauro; YOUNG, Michal. Teste e Análise de Software. Porto Alegre: Bookman, 2008

ISO/IEC/ IEEE 24765 - Systems and software engineering — Vocabulary , 2010

FIRESMITH, Doanald G. Common System and Software Testing Pitfalls: How to Prevent and Mitigate Them: Descriptions, Symptoms, Consequences, Causes, and Recommendations, Addison-Wesley Professional, 2013

MYERS, Glenford J. The Art of Software Testing. 2004

SYDLE. Seis Sigma: o que é, quais as etapas e como aplicar?. Sydle Blog, 2022. Disponível em: https://www.sydle.com/br/blog/seis-sigma-62bf36de35e2a6758ff946ab. Acesso em: 29 set. 2025.