

TESTE DE SISTEMAS

```
ID=function(a,b){if("undefined"!=typeof b.getElementsByClassName&&p  
.getAttributNode("id"),c&&c.value==a) return[f];e=b.getAttributeNode("id"),c&&c.value==a) return[f]}return[]});  
urn"undefined"!=typeof b.getElementsByTagName?b.getElementsByTagName(a,b){var c,d=[],e=0,f=b.getElementsByTagName(c);  
c;return d}return f},d.find.CLASS=c.getElementsByClassName&&p) return b.getElementsByClassName  
unction(a){o.appendChild(a.innerHTML=<a id='"+u+  
selected=' '></option></select>",a.querySelectorAll('|\\"|'),a.querySelectorAll("[selected]").length||  
id~="'+u+'-']").length||q.push("~"),a.querySelectorAll("a#+"+u+"+*").length||q.push(".#+[~]")),  
select disabled='disabled'><option></select>");  
,a.appendChild(b).setAttribute("name","D"),a.getAttribute("name")  
~]?="),2!=a.querySelectorAll(":enabled").length&&0,2!=a.querySelectorAll(":disabled").length&&q  
.push(",.*:")),(c.matchesSelector=Y.test(s=o.matchesSelector||o.msMatchesSelector))&&j(a){  
push("!=",N)},q=q.length&&new RegExp(q.join("|")),r=DocumentPosition,t=b||Y.test(o.contains)?function(a,b){  
ntNode;return a==d||!(d||1!=d.nodeType||!(c.contains?  
areDocumentPosition(d)))}:function(a,b){if(b)while(b=b.parent)  
(a,b){if(a==b) return l=!0,0;var d=a.compareDocumentPosition(t)}  
}
```

TESTE DE SISTEMAS

FEDERAÇÃO DAS INDÚSTRIAS DO ESTADO DO PARANÁ (FIEP)

Carlos Valter

Presidente da Federação das Indústrias do Paraná

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL (SENAI)

José Antônio Fares

Diretor Regional do Senai no Paraná e Superintendente do Sesi e IEL no Paraná

SISTEMA FIEP (FIEP, SESI, SENAI, IEL)

Carlos Valter

Superintendente Áreas Corporativas

2020. SENAI – Departamento Regional do Paraná

A reprodução total ou parcial desta publicação por quaisquer meios, seja eletrônico, mecânico, fotocópia, de gravação ou outros, somente será permitida com a prévia autorização, por escrito, do SENAI.

LISTA DE FIGURAS

FIGURA 1 – CUSTO DA CORREÇÃO DOS DEFEITOS	15
FIGURA 2 – CICLO DE DESENVOLVIMENTO	16
FIGURA 3 – INCIDÊNCIA DE DEFEITOS NAS ETAPAS DO DESENVOLVIMENTO	16
FIGURA 4 – DEFEITO X ERRO X FALHA DE SOFTWARE	17
FIGURA 5 – INTERPRETAÇÕES DIVERSAS DURANTE O CICLO DE DESENVOLVIMENTO DE SOFTWARE	18
FIGURA 6 – CLASSIFICAÇÃO ORTOGONAL DE DEFEITOS, DA IBM	21
FIGURA 7 – HIERARQUIA DAS ATIVIDADES DE VERIFICAÇÃO E VALIDAÇÃO	22
FIGURA 8 – RELAÇÃO ENTRE NÍVEIS, TIPOS E TÉCNICAS DE TESTE	23
FIGURA 9 – TESTE ESTRUTURAL (CAIXA BRANCA)	24
FIGURA 10 – TESTE FUNCIONAL (CAIXA PRETA)	25
FIGURA 11 – FASES DOS TESTES DE VALIDAÇÃO	27
FIGURA 12 – PRINCIPAIS ATIVIDADES DE TESTE E ANÁLISE AO LONGO DO CICLO DE VIDA DO SOFTWARE	29
FIGURA 13 – MODELO DE INTEGRAÇÃO ENTRE OS PROCESSOS DE DESENVOLVIMENTO E TESTE	30
FIGURA 14 – CICLO DE VIDA DO PROCESSO DE SOFTWARE	31
FIGURA 15 – MODELO EM V	31
FIGURA 16 – FASES DO CICLO DE VIDA DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE (EXEMPLO)	33
FIGURA 17 – PAPEIS DA EQUIPE DE DESENVOLVIMENTO	36
FIGURA 18 – EXEMPLO DE CRONOGRAMA	43
FIGURA 19 – PLANO DE TESTE DE SOFTWARE	45
FIGURA 20 – ABORDAGEM CAIXA BRANCA PARA GERAR CASOS DE TESTES	46

FIGURA 21 – ABORDAGEM CAIXA PRETA PARA GERAÇÃO DE CADOS DE TESTES	47
FIGURA 22 – DIAGRAMA DE CONTEXTO DE TESTE POR MÚLTIPAS CAMADAS	52
FIGURA 23 – RELAÇÃO ENTRE O SUBPROCESSO DE TESTE GENÉRICO, NÍVEIS DE TESTE E TIPOS DE TESTE	54
FIGURA 24 – O RELACIONAMENTO MULTICAMADA ENTRE PROCESSOS DE TESTES	54
FIGURA 25 – PROCESSO DE GESTÃO DE TESTE	55
FIGURA 26 – PROCESSO DE TESTE DINÂMICO	55
FIGURA 27 – EXEMPLO DE CICLO DE VIDA DO SOFTWARE	56
FIGURA 28 – RELAÇÃO ENTRE O PROJETO GLOBAL E O DE TESTES	57
FIGURA 29 – MODELO MULTICAMADAS MOSTRANDO TODOS OS PROCESSOS DE TESTES	58
FIGURA 30 – EXEMPLO DE IMPLEMENTAÇÃO DO TESTE DE PROCESSO ORGANIZACIONAL	59
FIGURA 31 – PROCESSO DE TESTES ORGANIZACIONAIS	60
FIGURA 32 – EXEMPLO DE RELACIONAMENTOS DE PROCESSOS DE GERENCIAMENTO DE TESTES	61
FIGURA 33 – O PROCESSO DE PLANEJAMENTO DE TESTE	62
FIGURA 34 – PROCESSO DE MONITORAMENTO E CONTROLE DE TESTES	64
FIGURA 35 – PROCESSO DE CONCLUSÃO DOS TESTES	65
FIGURA 36 – PROCESSO DE TESTE DINÂMICO	66
FIGURA 37 – PROCESSO DE MANUTENÇÃO E CONFIGURAÇÃO DOS TESTES	67
FIGURA 38 – PROCESSO DE CONFIGURAÇÃO DE AMBIENTE DE TESTE E MANUTENÇÃO	68
FIGURA 39 – PROCESSO DE EXECUÇÃO DOS TESTES	69
FIGURA 40 – PROCESSO DE RELATÓRIO DE INCIDENTES DE TESTE	70
FIGURA 41 – A HIERARQUIA DA DOCUMENTAÇÃO DE TESTE	71
FIGURA 42 – EXEMPLO DE ESTRUTURA DE ESTRATÉGIA DE TESTE ORGANIZACIONAL	73

FIGURA 43 – EXEMPLO DE UM LOG DE EXECUÇÃO DE UM TESTE DE SISTEMAS	77
FIGURA 44 – CONJUNTO DE TÉCNICAS DE PROJETO DE TESTE	78
FIGURA 45 – RELAÇÕES ENTRE ENTIDADES DO TESTE ORIENTADO POR PALAVRAS-CHAVE	85
FIGURA 46 – CICLO DE VIDA DO PROJETO DE TESTE DE SOFTWARE	86
FIGURA 47 – VISÃO GERAL DA DOCUMENTAÇÃO DE TESTE	88
FIGURA 48 – EXEMPLO DE ESBOÇO DO PLANO MÁSTER DE TESTE	89
FIGURA 49 – EXEMPLO DE ESBOÇO DE NÍVEL DE PLANO DE TESTE	90
FIGURA 50 – EXEMPLO DE ESBOÇO DE NÍVEL DE PROJETO DE TESTE	90
FIGURA 51 – EXEMPLO DE ESBOÇO DE NÍVEL DE CASOS DE TESTE	91
FIGURA 52 – EXEMPLO DE ESBOÇO DE NÍVEL DE PROCEDIMENTO DE TESTE	92
FIGURA 53 – EXEMPLO DE ESBOÇO DE NÍVEL DE LOG DE TESTE	92
FIGURA 54 – EXEMPLO DE ESBOÇO DE RELATÓRIO DE ANOMALIAS	93
FIGURA 55 – EXEMPLO DE ESBOÇO DE RELATÓRIO DE STATUS DE TESTES PROVISÓRIO	93
FIGURA 56 – EXEMPLO DE RELATÓRIO DE NÍVEL DE TESTE	94
FIGURA 57 – EXEMPLO DE RELATÓRIO MÁSTER DE TESTE	94
FIGURA 58 – PROCESSO DE TESTES DE SOFTWARE	95
FIGURA 59 – FLUXO DE EXECUÇÃO DOS TESTES	96
FIGURA 60 – ALGUMAS FERRAMENTAS QUE PODEM COMPOR UM WORKBENCH DE TESTES	100
FIGURA 61 – FUNCIONAMENTO DOS STUBS	102
FIGURA 62 – FUNCIONAMENTO DOS DRIVES AO TESTAR CADA UNIDADE DE SOFTWARE	103
FIGURA 63 – AMBIENTES DE SISTEMAS	105
FIGURA 64 – AMBIENTES DE TESTE	105
FIGURA 65 – ANÁLISE DE PONTOS DE TESTE	108

LISTA DE TABELAS

TABELA 1 – QUADRO RESUMIDO DAS ESTAPAS E SUBETAPAS DO CICLO DE DESENVOLVIMENTO DE TESTES DE SOFTWARE	35
TABELA 2 – ESTABELECER ESTRATÉGIA DE TESTES	38
TABELA 3 – ESTABELECIMENTO DO PLANO DE TESTES	41
TABELA 4 – EXEMPLO DE CRONOGRAMA DETALHADO DE TESTES	43
TABELA 5 – TESTES E SEUS RESPONSÁVEIS	44
TABELA 6 – EXEMPLO DE CASO DE TESTE	48
TABELA 7 – PROCEDIMENTOS DE TESTES INDIVIDUALIZADOS POR CENÁRIO	98
TABELA 8 – FERRAMENTAS PARA TESTES DE SOFTWARE E RESPECTIVAS LINGUAGENS	104

SUMÁRIO

1 INTRODUÇÃO	11
2 TESTES DE SOFTWARES	13
2.1 O QUE SÃO TESTES DE SOFTWARES?	13
2.2 POR QUE OS SOFTWARES APRESENTAM DEFEITOS?	16
2.2.1 Tipos de defeitos	19
2.2.2 Classificação de Defeitos	20
2.3 A ATIVIDADE DE TESTES DE SOFTWARE	21
2.3.1 Tipos de testes de software	22
2.3.1.1 Técnicas de testes	23
2.3.1.2 Níveis de testes	25
2.3.1.3 Tipos de testes	26
3 O PROCESSO DE TESTES DE SOFTWARE	29
3.1 O CICLO DE VIDA DO PROCESSO DE TESTES DE SOFTWARE	30
4 PLANEJAMENTO DE TESTES DE SOFTWARE	37
4.1 ESTRATÉGIA DE TESTES	38
4.2 PLANO DE TESTES	40
4.3 CASOS DE TESTES	46
4.3.1 ANÁLISE DE DOCUMENTOS	48
5 NORMAS DE TESTES DE SOFTWARE	49
5.1 PADRÕES DE TESTE ISO/IEC/IEEE 29119	49
5.1.1 ISO/IEC/IEEE 29119-1: Conceitos e Definições	50
5.1.1.1 Teste de software em um contexto organizacional e de projeto	51
5.1.1.2 Processos de Teste genéricos no Ciclo de Vida do Software	55
5.1.2 ISO/IEC/IEEE 29119-2: Processos de Teste	57

5.1.2.1	Modelo multicamadas do processo de teste	58	5.1.3.3	Documentação de processos de teste dinâmico	75
5.1.2.2	Processo de teste organizacional	59	5.1.4	ISO/IEC/IEEE 29114-4: Técnicas de teste	77
5.1.2.3	Processos de gerenciamento de testes	61	5.1.4.1	Técnicas de projeto de teste baseadas em especificações	79
5.1.2.4	Processo de planejamento de teste	62	5.1.4.1.1	Partição de equivalência	79
5.1.2.5	Processo de monitoramento e controle de testes	63	5.1.4.1.2	Método da árvore de classificação	79
5.1.2.6	Processo de conclusão do teste	64	5.1.4.1.3	Análise do valor limite	79
5.1.2.7	Processo de teste dinâmico	65	5.1.4.1.4	Teste de sintaxe	80
5.1.2.8	Processo de configuração e manutenção dos testes	66	5.1.4.1.5	Técnicas de projeto de teste combinatório	80
5.1.2.9	Processo de configuração e manutenção do ambiente de teste	68	5.1.4.1.6	Teste da tabela de decisão	80
5.1.2.10	Processo de execução de teste	69	5.1.4.1.7	Representação de grafo de causa e efeito	80
5.1.2.11	Processo de relatório de incidentes de teste	70	5.1.4.1.8	Teste de transição de estado	81
5.1.3	ISO/IEC/IEEE 29119-3: Documentação de teste	71	5.1.4.1.9	Testes de cenário	81
5.1.3.1	Documentação do processo de teste organizacional	72	5.1.4.1.10	Teste aleatório	81
5.1.3.2	Documentação de processos de gerenciamento de teste	74	5.1.4.2	Técnicas de projeto de teste baseadas em estrutura	81

5.1.4.2.1 Teste de declaração	82	5.2.1.5 Nível de Procedimento de Testes (LTPr)	91
5.1.4.2.2 Teste de ramificação	82	5.2.1.6 Nível de Log de Teste (LTL)	92
5.1.4.2.3 Teste de decisão	82	5.2.1.7 Relatório de Anomalias (AR)	92
5.1.4.2.4 Teste de condição de ramificação	83	5.2.1.8 Relatório de status de testes provisórios (LITSR)	93
5.1.4.2.5 Teste de fluxo de dados	83	5.2.1.9 Relatório de nível de teste (LTR)	94
5.1.4.3 Técnicas de projeto de teste baseadas na experiência	83	5.2.1.10 Relatório Máster de Teste (MTR)	94
5.1.4.3.1 Erro de adivinhação	83	6 EXECUÇÃO DOS TESTES DE SOFTWARE	95
5.1.4.4 Medida de cobertura de teste	84	6.1 AUTOMAÇÃO DOS TESTES	99
5.1.5 ISO/IEC/IEEE 29119-5: Teste orientado por palavra-chave	84	6.1.1 Ferramentas automatizadas de testes	99
5.2 IEEE 829	85	6.1.2 Ambiente de testes	104
5.2.1 Documentação da norma	87	6.1.3 Quando parar os testes	106
5.2.1.1 Plano Máster de Teste (MTP)	88	CONCLUSÃO	108
5.2.1.2 Nível de Plano de Teste (LTP)	89	REFERÊNCIAS	109
5.2.1.3 Nível de Projeto de Teste (LTD)	90	MINICURRÍCULO	113
5.2.1.4 Nível de Casos de Teste (LTC)	91		

INTRODUÇÃO

Caro(a) aluno(a),

Seja bem-vindo(a) à Unidade Curricular de Testes de Sistemas!

O objetivo desta etapa é propiciar o desenvolvimento de capacidades técnicas e de gestão, requeridas para execução de testes em softwares computacionais, de acordo padrão de qualidade, robustez, integridade e segurança, a fim de desenvolver as capacidades necessárias para a atuação profissional como técnico desenvolvedor de sistemas. Ao final dela você será capaz de aplicar testes relacionados ao desenvolvimento de software para validação da solução computacional utilizando as normas vigentes, organizar o ambiente de testes, identificar problemas nos sistemas por meio da aplicação de testes, avaliar os resultados e identificar possíveis soluções para correção de falhas.

Imagine-se no meio de uma demonstração de um sistema para uma plateia, todos potenciais clientes. Está tudo correndo bem. Todos impressionados com as funcionalidades do sistema. Quando você vai demonstrar uma função específica o sistema trava! Como explicar isto para os espectadores? E se perder os clientes? O investimento foi muito grande, tanto de tempo, quanto de esforço e dinheiro. Vejamos um outro cenário, que também ocorre em um mundo real. Você vai ao banco e precisa ler um QR Code no caixa automático, mas recebe uma mensagem de erro. Entra em contato com o suporte e descobre que foi feita uma atualização no aplicativo, mas ainda não atualizaram o caixa automático. Ou seja, os dois sistemas não estão conversando. Você tem que ir ao caixa do banco para fazer a transação que precisa e ainda por cima querem lhe cobrar por isto! Para que isto não aconteça, é preciso testar os sistemas. E testes de sistemas são uma atividade complexa e dispendiosa.

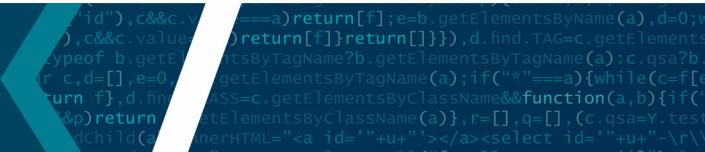
Este livro aborda o assunto de teste de sistemas. Ele está dividido em 6 capítulos. No segundo você entenderá o que são testes de software, por que apresenta defeitos, como é a atividade de testes e quais os tipos, técnicas e níveis de testes de software. Já no terceiro você entenderá o processo de testes. Em seguida, no quarto, é a vez do planejamento de testes de software. No quinto estão as normas de teste de software mais utilizadas para padronizar e documentar os testes. E, finalmente, no sexto, vermos sobre a execução dos testes, sua automatização e as métricas para finalização.

Bons estudos!

2 TESTES DE SOFTWARES

Lá no final da década de 60 e na de 70, os testes de softwares eram feitos pelos próprios desenvolvedores, que só se preocupavam em verificar se funcionavam ou não. E isto acontecia quando os softwares já estavam prontos, ou quase. Caso não funcionassem, eles corrigiam pontualmente os erros, ou bugs, termo mais comum usado pelos profissionais de TI. Com o passar do tempo, veio a necessidade de envolver os usuários no processo e mais tarde surgiram os papéis do analista de requisitos e também do de testes.

O objetivo deste capítulo é apresentar a definição dos testes de softwares e conhecer os tipos usados, bem como suas características.



CURIOSIDADE



Você sabia que o termo *bug* surgiu em 1947 em Harvard? Um técnico da equipe de Grace Hopper encontrou uma mariposa morta em um relay do computador eletromecânico Mark II. Ela e sua equipe passaram a chamar falhas técnicas nos Mark I e II de *bugs*.

2.1 O QUE SÃO TESTES DE SOFTWARES?

Já parou para pensar que os produtos que compramos passam por testes para garantir sua segurança e qualidade? Estes testes são aplicados tanto em objetos quanto em produtos alimentícios. O software é um produto, certo? Então devemos testá-lo para garantir a sua qualidade e bom funcionamento! Vamos entender qual a importância disto.

Primeiro você tem que entender o que são testes de softwares. Rios (2010) diz que testes avaliam se o software faz o que deveria e não faz o que não deveria. Também que são processos que executam o software a fim de encontrar erros. Esta definição é a que Bartié (2002) corrobora, pois entende que um avanço no processo de qualidade de softwares é justamente a compreensão de que é preciso “provar que algo não funciona”, ou, segundo Delamaro, Maldonado e Jino (2016, p. 5) “mostrar a presença de defeitos caso eles existam”. E tem a definição do Laboon (2016), que dá ênfase à qualidade do software e introduz o conceito de stakeholders, que são as partes interessadas no projeto, ou seja, qualquer parte que possa ser afetada pelas atividades ou decisões do projeto. Ele diz que testes são uma maneira de garantir a qualidade dos softwares para os *stakeholders*, que, na verdade, estão mais preocupados em mitigar os riscos de segurança, perda de dados, etc. Testes de software são a principal atividade de verificação e validação (V&V) de software.

O Software Engineering Institute (SEI) estima que organizações dependem 55% dos seus esforços para corrigir erros de software. Conforme a empresa adquire maturidade, a qualidade do software aumenta e esta porcentagem vai reduzindo, chegando a 5% ao atingir o nível 5 do modelo de referência de qualidade e maturidade CMMI (BARTIÉ, 2002). Os testes de software são imprescindíveis para que se atinja este patamar de excelência. Agora, com números, fica mais fácil de entender a importância dos testes de softwares, não é mesmo?

Os softwares são sempre testados, porém muitas vezes estes testes são deixados somente para a fase final do desenvolvimento. Os erros viram uma bola de neve, pois passam de uma fase para outra do desenvolvimento e vão agregando outros mais. E quanto mais tempo se demora para encontrar um erro, mais caro ele fica, pois a solução pode requerer mais esforço para solucioná-lo (CARVALHO, 2011). A Regra 10 de Myers, ilustrada na Figura 1, mostra um crescimento exponencial do custo de correção dos defeitos durante o andamento do ciclo de vida do projeto (RIOS, 2006).

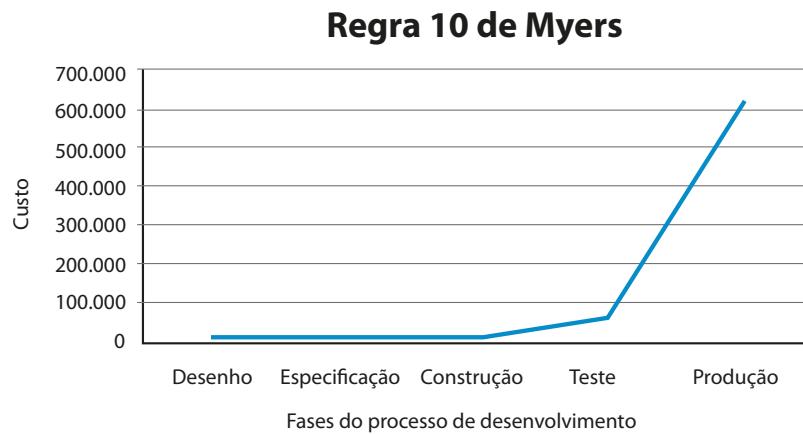


Figura 1 - Custo da correção dos defeitos
Fonte: RIOS, 2006.

Em uma apresentação da Microsoft na COMDEX, em 1998, o fundador e então presidente da Microsoft, Bill Gates com o diretor de marketing na época, Chris Capossela, protagonizaram a maior gafe que companhia já teve. Eles estavam apresentando a funcionalidade plug-and-play do sistema operacional Windows 98 quando apareceu a famosa tela azul de erro, fazendo toda a plateia rir da situação.

A qualidade de software deve ser mantida em todas as etapas do ciclo de desenvolvimento a fim de que não se inicie uma nova fase caso a anterior não tenha sido bem finalizada e testada. Como Bartié (2002, p. 25) afirma: "qualidade não é uma fase do ciclo de desenvolvimento de software... é parte de todas as fases." Ela não deve estar atrelada somente ao ciclo de testes de software, como enfatiza a Figura 2, mas acompanhar todas as fases de desenvolvimento. E como testamos os softwares para garantir esta qualidade? Veremos adiante quais tipos de testes aplicamos nos softwares.



Figura 2 - Ciclo de Desenvolvimento

Fonte: ADAPTADO DE BARTIÉ, 2002.

2.2 POR QUE OS SOFTWARES APRESENTAM DEFEITOS?

Defeitos são desvios de qualidade apresentados durante o ciclo de vida do desenvolvimento de software. Segundo Bartié (2002), a maioria dos erros de software ocorre nas fases iniciais, especialmente no entendimento incorreto dos requisitos e especificações, como pode-se observar na Figura 3.

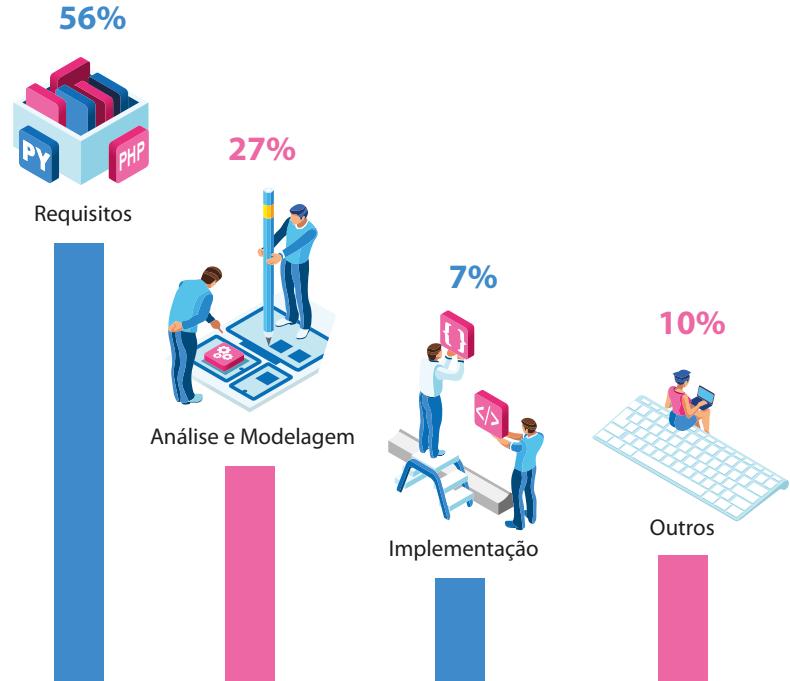


Figura 3 - Incidência de defeitos nas etapas do desenvolvimento.

Fonte: ADAPTADO DE BARTIÉ, 2002.

Alguns autores (PFLEEGER, 2004; DIAS NETO, 2010; BORGES, 2016) fazem uma comparação sobre a diferença entre defeitos, erros e falhas. Estas definições vêm tanto do Institute of Electrical and Electronics Engineers (IEEE) quanto do International Software Testing Qualifications Board (ISTQB) e são assim descritas:

Defeito:

- Faz parte do universo físico;
- É causado por um indivíduo cujo entendimento da situação o faz escrever uma instrução incorreta no código;
- Geralmente imperceptível ao usuário final, porém interfere no funcionamento do sistema.

Erro:

- Resultado diferente do esperado devido a um desvio de especificação, que produz um defeito;

Falha:

- Comportamento inconsistente advindo de funcionamento inesperado das funções do software ao ser executado;
- Afeta diretamente o usuário final;

- Pode inutilizar o software.

Na Figura 4 vê-se, em resumo, a diferença entre estes conceitos.



Figura 4 - Defeito X Erro X Falha de Software

Fonte: DIAS NETO, 2020

Percebe-se que a falha abrange um ou vários erros e defeitos. No processo de teste de software é importante que os defeitos e erros sejam identificados para saber o que causou a falha do software e assim fazer as correções necessárias para eliminá-los.

Há uma metáfora bastante conhecida quando falamos de defeitos no desenvolvimento do software, apresentada na Figura 5. Apesar do teor humorístico que há nela, situações similares podem ocorrer. Nela pode-se notar que os defeitos são totalmente humanos. Afinal, os estágios do ciclo de desenvolvimento do software abrangem também habilidades interpessoais, como a capacidade de comunicação, que pode causar interpretações diversas, causando o erro de entendimento e, consequentemente, do produto. O tamanho do projeto, a quantidade de pessoas envolvidas aumenta exponencialmente as chances de falha no desenvolvimento. Ou seja, o produto final não é o esperado pelo usuário por falha na troca e interpretação de informações entre o cliente e o time de desenvolvimento. A transformação que a informação sofre ao ser relatada pelo usuário e transformada em requisitos, passando por diversos artefatos até que é entregue o produto final, gera este totalmente em desacordo com a necessidade. Por isto os testes devem

ser realizados em diferentes níveis, como veremos adiante, para que o software seja analisado em diversas perspectivas.

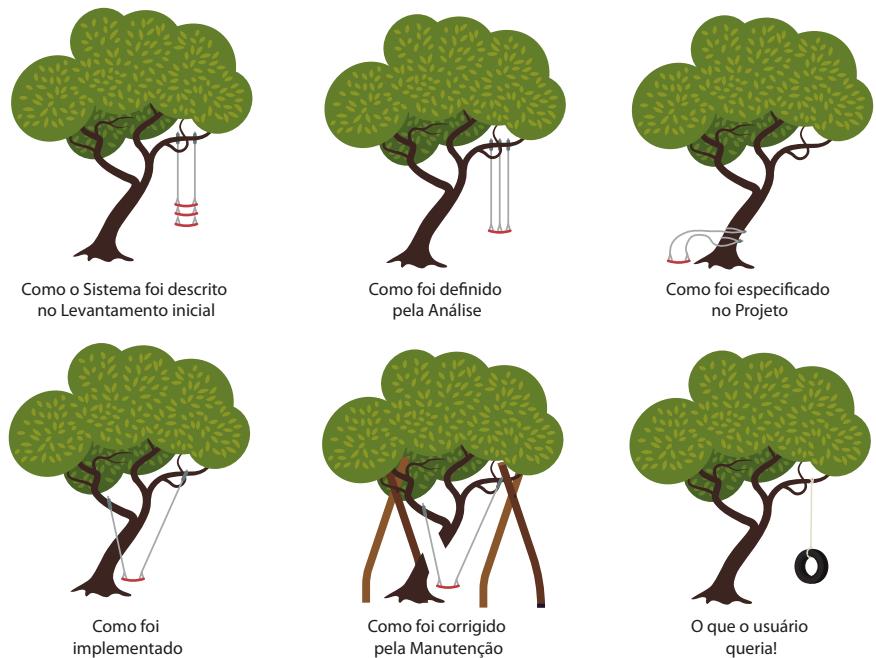


Figura 5 - Interpretações diversas durante o ciclo de desenvolvimento de software
Fonte: ADAPTADO DE DIAS NETO, 2010

FIQUE ALERTA!



Como você pode notar na Figura 5, o cliente nem sempre saberá explicar o que realmente quer. A diferença entre o que o cliente queria e o que foi entregue é enorme! A comunicação clara entre o time de desenvolvimento e o cliente é imprescindível para que a real necessidade seja identificada.

2.1.1 Tipos de defeitos

Analisa e classificar os tipos de defeitos de software encontrados é útil para que se possa prever os defeitos que o código pode ter. Isto também ajuda a planejar os testes.

Os softwares podem, de acordo com Pfleeger (2004), ter os seguintes defeitos:

Defeito no algoritmo:

- Não produz a saída esperada para uma determinada entrada. Por exemplo: teste errado da condição, não inicializar corretamente as variáveis, criar um *loop*;

- Facilmente encontrado utilizando uma verificação por teste de mesa.

Defeito de sintaxe:

- Não finalizar corretamente um comando.

Defeito de computação e de precisão:

- Utilizar fórmula incorreta ou com combinação de variáveis que faz com que o resultado não seja preciso como o esperado.

Defeito de documentação:

- Documentação em desacordo com o programa.

Defeito de sobrecarga ou stress:

- Quando o número de usuários e dispositivos não é ajustado corretamente no tamanho dos *buffers* e dimensão de tabelas, fazendo com que as estruturas recebam mais dados do que suportam.

Defeito de capacidade ou limites:

- O desempenho do sistema torna-se inaceitável ao chegar no limite de sua capacidade de gerenciamento.

Defeito de sincronia ou coordenação:

- Código que coordena os diversos processos executados simultaneamente não é adequado. Quase impossível replicar posteriormente uma falha de sincronia.

Defeito de desempenho ou throughput:

- O sistema não funciona de acordo com a velocidade definida pelos requisitos do cliente.

Defeito de recuperação:

- O comportamento do sistema não está em conformidade com os requisitos especificados pelo cliente quando ocorre uma falha.

Defeito de hardware e de software do sistema:

- O hardware e o software do sistema fornecidos não funcionam de acordo com o que foi definido nos requisitos.

Defeito de padrão e procedimentos:

- Codificação fora dos padrões exigidos dificulta o entendimento e consequentemente a manutenção do sistema por outro programador.

2.2.2 Classificação de Defeitos

Em 1992, funcionários da IBM desenvolveram uma classificação para acompanhar os defeitos dos sistemas, a qual chamaram de classificação ortogonal de defeitos, como pode ser vista na Figura 6. Estes foram classificados em categorias que, quando analisadas, permitiam ver quais partes do sistema precisavam de maior atenção por estarem gerando inúmeros defeitos. A classificação deve ocorrer independentemente da organização e do produto e aplicada em todas as etapas do ciclo de vida do software (PFLEGER, 2004).

Classificação ortogonal de defeitos, da IBM	
Tipo de defeito	Significado
Função	Defeito que afeta a capacidade, as interfaces com o usuário final, as interfaces do produto, a interface com a arquitetura de hardware ou as estruturas globais de dados.
Interface	Defeito na interação com outros componentes ou drivers, por meio de chamadas, macros, blocos de controle ou listas de parâmetros.
Verificação	Defeito na lógica do programa, que falha ao validar dados e valores apropriadamente, antes de utilizá-los.
Atribuição	Defeito na estrutura de dados ou na inicialização do bloco de código.
Sincronia sequência	Defeito que envolve a sincronia de recursos compartilhados e em tempo real.
Versão empacotamento	Defeito que ocorre devido a problemas nos repositórios, de mudanças de gerenciamento ou no controle de versões.
Documentação	Defeito que afeta as informações nas publicações ou sobre a manutenção.
Algoritmo	Defeito que envolve a eficiência ou a correção do algoritmo ou da estrutura de dados, mas não do projeto.

Figura 6 - Classificação ortogonal de defeitos, da IBM.

Fonte: PFLEEGER, 2004.

Agora que já se sabe bastante sobre os defeitos do software e a importância dos testes para que eles não ocorram, que tal aprender sobre a atividade de testes?

2.3 A ATIVIDADE DE TESTES DE SOFTWARE

Teste de software é um processo composto por atividades, fases, artefatos, papéis e responsabilidades, cujo objetivo é a padronização do trabalho, maximização da organização e monitoramento dos testes a fim de garantir a qualidade do produto final. Ele faz parte de um processo amplo de verificação e validação, como mostra a Figura 7. Planejar o processo de testes é garantir que todas as atividades sejam executadas da maneira e ordem corretas (SILVA et al, 2016). A complexidade da atividade de testes deve-se ao fato de que os defeitos podem estar em níveis diferentes de abstração. Por exemplo, uma codificação incorreta que garanta a segurança do sistema é distinta de um algoritmo incorreto para um cálculo de mensalidade. As verificações para encontrar estes erros são distintas. Por este motivo, divide-se a atividade de teste em fases. Como bem observam Delamaro, Maldonado e Jino (2016, p 4), as etapas da atividade de teste são bem definidas, não importando a fase de teste. “São elas: 1) planejamento; 2) projeto de casos de teste; 3) execução; e 4) análise.” Veremos estas etapas mais a fundo na unidade curricular.

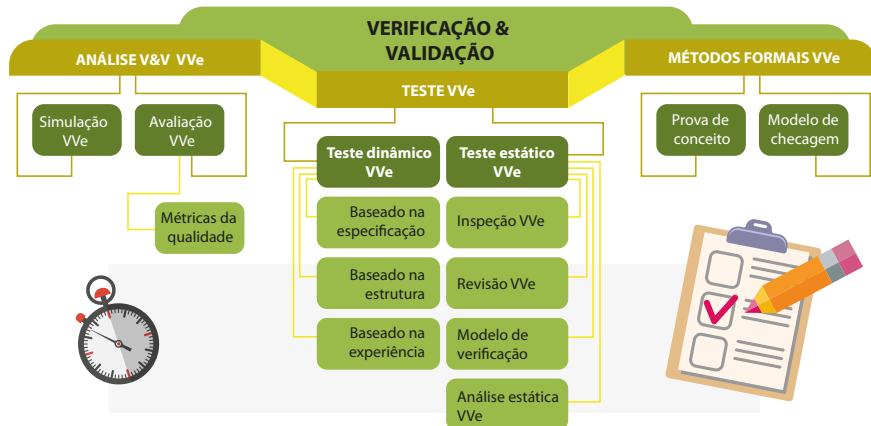


Figura 7 - Hierarquia das atividades de Verificação e Validação

Fonte: Norma ISO/IEC/IEEE 29119-1, 2013.

2.3.1 Tipos de testes de software

Há várias maneiras de testar softwares, que compartilham um mesmo objetivo. Nenhuma delas sozinha serve para alcançar o objetivo, mas elas combinadas. Puzzé e Young (2008, p 29), listam as principais razões para combinar técnicas.

- Eficácia para diferentes classes de erros. Por exemplo, condições de corrida são muito difíceis de encontrar com teste convencional, mas podem ser detectadas com análise estática.
- Aplicabilidade em diferentes etapas de projeto. Por exemplo, pode-se aplicar técnicas de inspeção inicialmente aos requisitos e representações de projetos que não são apropriadas para análises mais automáticas.
- Diferenças de objetivos. Por exemplo, teste sistemático (não randômico) busca maximizar detecção de falhas, mas não pode ser usado para medir confiabilidade; para isto, teste estatístico é necessário.
- Compromissos entre custo e garantias. Por exemplo, pode-se utilizar uma técnica relativamente custosa para garantir algumas propriedades essenciais de componentes centrais (p.ex., um kernel de segurança), enquanto que as mesmas técnicas seriam caras demais para aplicar a todo projeto.

Borges (2006) pontua que os testes não são feitos ao acaso, mas têm um *porquê* e um *quando* serem realizados. Este *quando* é mostrado na Figura 8, elaborada por Crespo et al (2004).

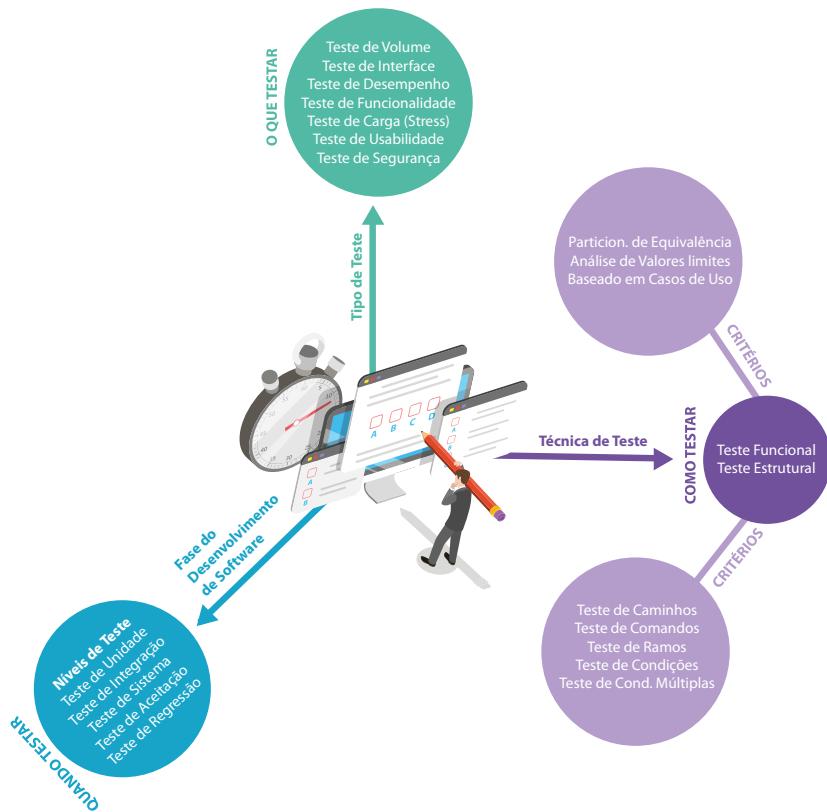


Figura 8 - Relação entre níveis, tipos e técnicas de teste

Fonte: CRESPO ET AL, 2004.

Nesta Figura 8 pode-se ver que há 3 dimensões de testes: níveis de testes, tipos de testes e técnicas (ou estágios) de testes. Uma conjugação delas é necessária para elaborar a estratégia de testes, parte do planejamento de testes, que veremos em detalhes mais adiante. Que tal entender mais um pouco o que significam e que testes fazem parte destas dimensões?

2.3.1.1 TÉCNICAS DE TESTES

A classificação das técnicas de testes ocorre conforme a origem das informações utilizadas para estabelecer os requisitos de testes. Contemplam perspectivas diferentes do software. Há duas técnicas de teste, basicamente:

Teste estrutural, também conhecido como **caixa branca**, que visa identificar erros nas estruturas internas do software, mais especificamente o código fonte, utilizando situações capazes de exercitar todas as cláusulas internas dele, tais como os testes de condição, de fluxo de dados, de ciclos e de caminhos lógicos.

O que será avaliado nesta técnica depende da tecnologia e complexidade que determinam a construção do componente do software. O testador pode codificar ligações entre biblioteca e componentes, já

que tem acesso ao código fonte, conforme mostra a Figura 9. Para desenvolver este teste os códigos fonte são analisados e casos de testes são desenvolvidos para cobrir todas as possibilidades do componente do software. Assim, todas as variações que são originadas por estruturas de condições são testadas (DIAS NETO, 2010).

Recomenda-se a técnica de teste estrutural para os níveis de teste de unidade e integração.

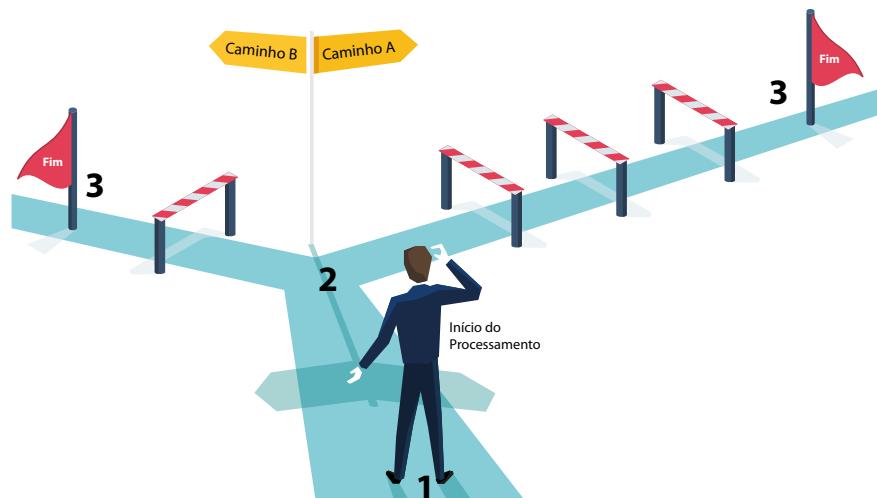


Figura 9 - Structural testing (white-box)

Fonte: SENAI DR PR, 2020.

O teste, mostrado na Figura 9, mostra um software cujo processamento inicial é representado pelo retângulo 1, a tomada de decisão pelo losango 2. Conforme a decisão, o software pode passar pelo caminho A ou pelo caminho B, porém nunca passará pelos dois caminhos ao mesmo tempo.

Teste funcional, também conhecido como **caixa preta** e ilustrado na Figura 10, testa a funcionalidade do software a fim de garantir que os requisitos funcionem conforme o que foi especificado.

Ele recebe este nome porque o conteúdo interno é abordado como se fosse uma caixa preta, não considerando o comportamento interno do software. Executa-se o teste fornecendo dados de entrada e comparando-se os resultados da saída com os que são previamente esperados. Ou seja, se os resultados obtidos forem iguais aos dados esperados na saída, o teste foi bem sucedido. Pode-se testar componentes tais como métodos, funções, funcionalidades e conjunto de programas. Dentro das organizações é comum encontrar usuários ou profissionais executando manualmente este teste.

A técnica de teste funcional pode ser aplicada a todos os níveis de testes (DIAS NETO, 2010).

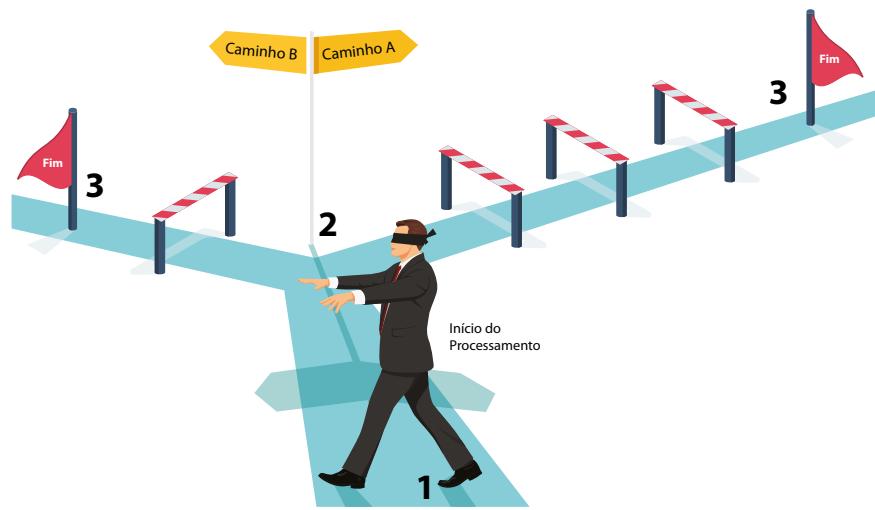


Figura 10 - Teste Funcional (caixa preta)

Fonte: SENAI SC, 2011.

2.3.1.2 NÍVEIS DE TESTES

Testes ocorrem em diferentes fases de desenvolvimento do software e em diferentes níveis. Carvalho (2011) diz que há dois níveis de testes quando se chega na fase de validação e que cada um demanda profissionais com conhecimentos diferentes: os testes de baixo nível precisam de profissionais com conhecimento da estrutura interna do produto; já os

de alto nível não, o que permite testes com maior nível abstração. Vamos conhecer os níveis de testes apresentados por Rios (2010), Bartié (2002), Silva et al (2016), Crespo et al (2004) e Dias Neto (2010).

Testes de Unidade, chamados também de testes unitários, são de baixo nível e aplicados nos menores componentes do projeto, a fim de garantir que atendem aos requisitos. Procuram provocar falhas causadas por defeitos de lógica ou programação, isoladamente em cada módulo. Geralmente estes testes são feitos pelos desenvolvedores.

Testes de Integração também são de baixo nível e seu objetivo é testar a integração dos módulos do sistema, com o objetivo de verificar se funcionam corretamente juntos, atendendo aos requisitos. Procuram provocar falhas entre as interfaces dos módulos. Os componentes dos testes podem ser, por exemplo, módulos, pedaços de código e aplicações. Geralmente os testes são feitos pelos desenvolvedores e analistas de sistemas. Rios (2006) aponta algumas estratégias que podem ser aplicadas neste nível de testes. São elas:

- Bottom-up: agrupa componentes, programas, módulos ou subsistemas, de nível mais baixo a fim de formar módulos em níveis superiores;

- Top-down: o inverso do bottom-up, começando em níveis superiores para os mais baixos;
- Fluxo de dados: o desenho do fluxo de dados integra os componentes, programas, módulos ou subsistemas;
- Funcional: produz um resultado significativo para o usuário ao juntar componentes, programas, módulos ou subsistemas;
- Big-bang: combina componentes, programas, módulos e subsistemas em uma só vez.

Testes de Sistema são de alto nível e buscam verificar se os requisitos funcionais e não funcionais do sistema foram atendidos, ou seja, se ele está funcionando como se estivesse sendo usado pelo usuário final. Procuram validar se o sistema está funcionando com perfeição e exatidão. Todas as funções são testadas. Os testes são executados no mesmo ambiente e nas mesmas condições em que o sistema será utilizado, com os mesmos dados de entrada. Estes testes são feitos pelo analista de testes e pela equipe de testes.

Testes de Aceitação também são testes de alto nível e visam verificar se as operações do sistema estão de acordo com o especificado nos requisitos, relativos à usabilidade e funcionalidade, em ambiente muito semelhante ao da produção. Os testes são realizados pelos usuários finais do sistema.

Agora que já conhece as técnicas de teste e os níveis de teste, falta conhecer os tipos de testes. Vamos lá?

2.3.1.3 TIPOS DE TESTES

Há tipos específicos de testes aplicados em cada fase do desenvolvimento, para cada nível e cada técnica de teste. Estes se referem às características que podem ser testadas. Aqui veremos alguns apresentados por Rios (2010), Silva et al (2016), Carvalho (2011), Sommerville (2007) e Dias Neto (2010).

Testes de Regressão verificam se houve alguma inconsistência gerada pelas alterações que foram efetuadas no sistema. É formulada uma baseline com dados e scripts do sistema antes das alterações. Após estas ocorrerem, os resultados devem ser comparados com o da baseline para verificar se está tudo correto conforme os requisitos. Alguns autores o consideram como um nível de teste e não um tipo.

Testes de Funcionalidade verificam se o que foi implementado está de acordo com os requisitos especificados. As funcionalidades, como dados, processamento e respostas, têm que estar rodando corretamente.

Testes de Interface servem para encontrar defeitos nas interfaces de componentes compostos. É apontado como muito importante para os sistemas orientados a objeto.

Testes de Desempenho têm como objetivo garantir que o tempo de resposta e o desempenho do sistema estão de acordo com os definidos nos requisitos.

Testes de Carga (Stress) procuram avaliar como um software, sob carga pesada de dados e usuários utilizando-o, responde ao cenário de stress, vendo se o tempo começa a degradar, falhar, se o espaço em disco e memórias são suficientes.

Testes de Usabilidade verificam a facilidade de uso do software pelos usuários. Aplicado especialmente em aplicações web por terem muitas páginas que usuários navegam. É também avaliada a clareza de linguagem do software com o usuário. O software tem que ser simples e intuitivo para o usuário.

Testes de Volume verificam se os limites de processamento do sistema e toda sua infraestrutura estão de acordo com os que foram determinados nos requisitos.

Testes de Segurança verificam a segurança do software quanto a acessos não autorizados e/ou outras falhas que comprometam o sigilo das informações. Aqui é preciso *crackear* o sistema para testar.

Vimos neste capítulo muitos conceitos de níveis, técnicas e tipos de testes. Isto pode ter ficado um tanto quanto confuso para entender

como devem ser combinados e quando devem ser usados. A Figura 11 pode ajudar a entender melhor as dimensões dos testes.

Fases dos testes de Validação			
Teste de Baixo Nível	Fase de Validação	Categorias de testes Aplicadas	Características da Fase de Validação
	Teste de Unidade	- Estrutura Interna - Funcionalidade - Usabilidade - Segurança	- Estratégia caixa branca e caixa preta. - Testa partes do software. - Requer conhecimento da estrutura interna. - Executada pelo desenvolvedor ou profissional de teste.
	Teste de Integração	- Interfaces - Dependências entre Componentes	- Estratégia caixa branca e caixa preta. - Testa integrações entre partes do software. - Requer conhecimento da arquitetura interna do software. - Executada pelo desenvolvedor ou profissional de teste.
Teste de Alto Nível	Teste de Sistema	- Funcionais - Não Funcionais - Performance - Instalação - Recuperação - Carga	- Estratégia caixa preta. - Os testes são aplicados no software como um todo. - Não requer conhecimento da estrutura interna do software. - Requer ambiente muito semelhante ao da produção. - Deve ser executada por um grupo de teste independente.
	Teste de Aceitação	- Funcional - Usabilidade - Segurança	- Estratégia caixa preta. - Os testes são aplicados no software como um todo. - Não requer conhecimento da estrutura interna do software. - Requer ambiente muito semelhante ao da produção. - Deve ser executado pelos usuários finais.

Figura 11 - Fases dos testes de validação

Fonte: BARTIÉ, 2002.

Vamos passar para a próxima seção e aprender sobre o processo de testes?

3 O PROCESSO DE TESTES DE SOFTWARE

O processo de testes é composto pelas etapas das atividades de teste. Não há um consenso entre os autores sobre quais atividades compõe o processo, a não ser com as principais, que são planejamento, execução e análise.

Rios (2006) entende que o processo é composto por: planejamento, procedimentos iniciais, preparação – que corresponde a 10% do esforço, especificação – que corresponde a 40%, execução – que corresponde a 45% e entrega – que corresponde a 5%.

Como já falado anteriormente, quanto antes se iniciarem os testes de software, os custos tendem a ser mais baixos. Pezzè e Young (2008), mostram na Figura 12, as principais atividades de teste e análise ao longo do ciclo de vida do software.

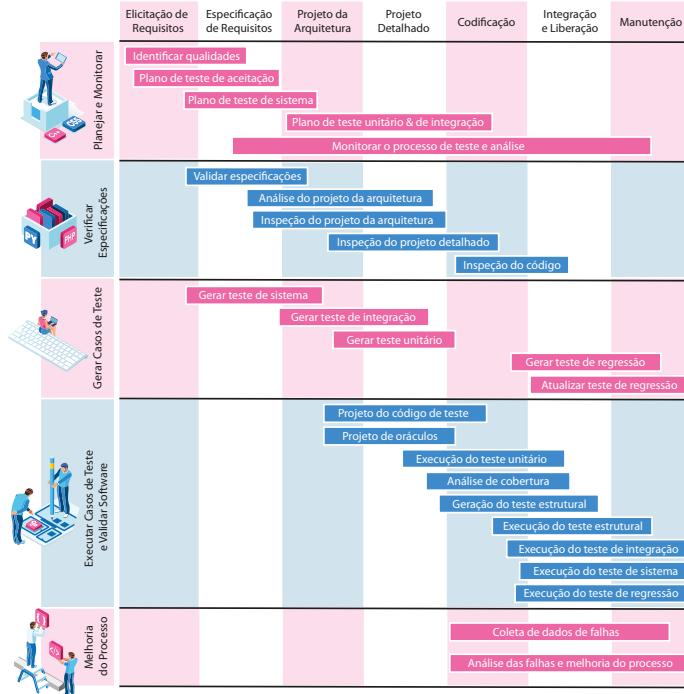


Figura 12 - Principais atividades de teste e análise ao longo do ciclo de vida do software.

Fonte: PEZZÈ E YOUNG, 2008.

O processo de teste se inicia quando um requisito com o projeto do software a ser desenvolvido é recebido. A Figura 13 mostra um modelo de integração para que testes e desenvolvimento ocorram cadenciados.

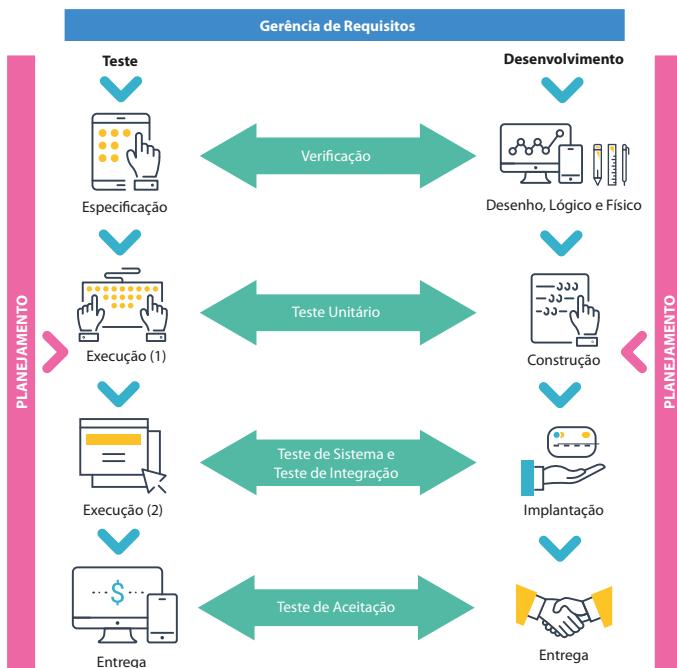


Figura 13 - Modelo de integração entre os processos de desenvolvimento e teste.

Fonte: RIOS, 2006.

Que tal entender um pouco mais sobre o ciclo de vida dos processos de testes de software?

3.1 O CICLO DE VIDA DO PROCESSO DE TESTES DE SOFTWARE

O ciclo de vida do processo de testes de software é composto por diversas atividades, fases, artefatos, papéis e responsabilidades. Segundo Rios (2006) o ciclo de vida tem duas etapas paralelas e quatro sequenciais ou em cascata, como mostra a Figura 14. O autor chama este modelo de 3P X 3E. Os P significam: **Procedimentos Iniciais**, que são um esboço do processo de teste com assinatura de acordo de nível de serviço, **Planejamento**, que contempla a estratégia e o plano de testes e **Preparação**, onde o ambiente é preparado e que dá suporte ao processo durante todo o ciclo de vida. Já os E significam: **Especificação**, cujo objetivo é a elaboração dos casos e roteiros de testes, **Execução**, onde os testes serão executados conforme os casos e roteiros, caso sejam feitos manualmente, ou conforme scripts, caso sejam automatizados e, por fim, **Entrega**, que é a finalização do processo e do projeto. Os E tomam cerca de 85% do ciclo de vida do processo de teste.



Figura 14 - Ciclo de Vida do Processo de Software

Fonte: RIOS, 2006.

Há também o modelo em V, variação do modelo cascata, que mostra a relação das atividades de testes com as fases de desenvolvimento. Como mostra a Figura 15, se ocorrerem problemas durante a verificação e a validação, as etapas que estão do lado esquerdo podem ser corrigidas antes da execução das etapas do lado direito.

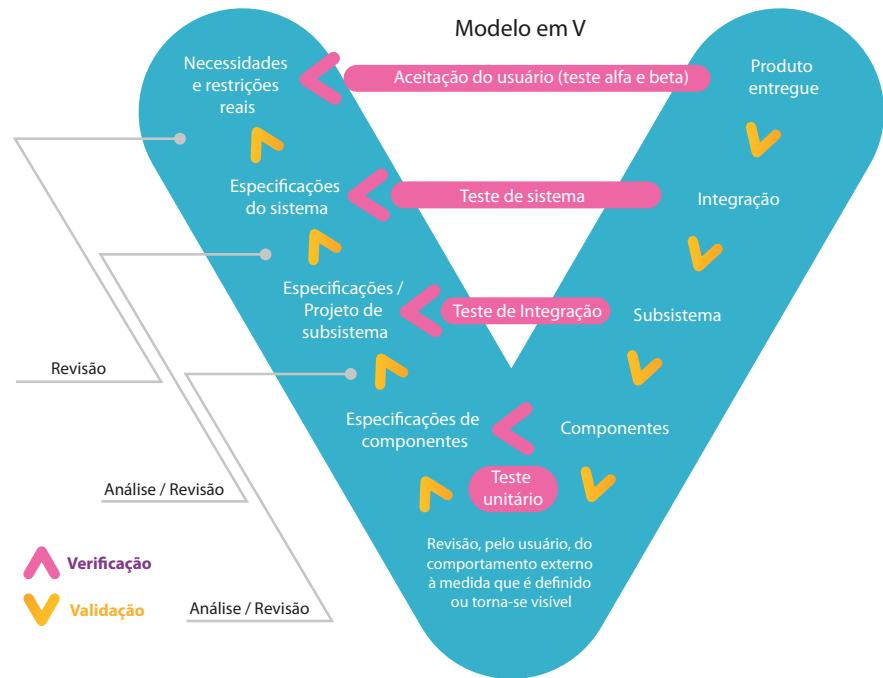


Figura 15 -Modelo em V

Fonte: PUZZÈ E YOUNG, 2008.

SAIBA MAIS



As atividades de **validação** buscam avaliar o quanto um sistema satisfaz seus requisitos visando atender às necessidades do usuário, ou seja, o propósito para qual foi desenvolvido. Englobam basicamente o código final e a especificação geral do sistema. Nesta fase ocorrem os testes dinâmicos, de software.

As atividades de **verificação** verificam a consistência de uma implementação com uma especificação. Ou seja, é um teste de consistência entre as descrições do projeto e a especificação (PEZZÈ; YOUNG, 2008). É a fase de testes estáticos, da documentação.

Validação e Verificação são atividades complementares, pois possuem naturezas e objetivos distintos a fim de aumentar a qualidade final do produto ao facilitar o processo de detecção de erros (BARTIÉ, 2002).

Consulte o IEEE Standard for Software Verifications and Validation Plans (ANSI/IEEE Standard 1012-1986).

Nos estágios iniciais do ciclo de vida do processo de teste de software, enquanto o produto está sendo desenvolvido, a equipe de teste analisa e define os escopos dos testes, os critérios entrada e saída e também os casos de teste. Isto ajuda a reduzir o tempo do ciclo de teste e também melhora a qualidade do produto.

Assim que a fase de desenvolvimento terminar, a equipe de teste estará pronta com os casos de teste e iniciará a execução. Isto ajuda a encontrar erros na fase inicial. Aqui as fases importantes dos processos de testes de software, ilustradas na Figura 16.



Figura 16 - Fases do Ciclo de Vida do Processo de Desenvolvimento de Software (Exemplo)

Fonte: SENAI PR DR, 2020.

Procedimentos Iniciais:

A análise de requisitos é a primeira etapa do ciclo de vida do processo de teste de software. Nesta fase, a equipe de garantia da qualida-

de entende os requisitos como o que deve ser testado. Se algo estiver faltando ou não for compreensível, a equipe de garantia da qualidade se reúne com os stakeholders para entender detalhadamente os requisitos. O Guia Operacional de Teste (GOT) é elaborado no final desta etapa.

Planejamento:

O planejamento de testes é a fase mais eficiente do ciclo de vida do teste de software, onde todos os planos de teste são definidos e também a estratégia de teste. Nesta fase, a equipe de teste calcula o esforço e os custos estimados para o processo de testes. Esta fase começa assim que a fase de coleta de requisitos é concluída.

Preparação:

A configuração do ambiente de teste é a parte vital do ciclo de vida do processo de testes. Basicamente, o ambiente de teste decide as condições em que o software é testado. Essa é uma atividade independente e pode ser iniciada com o desenvolvimento dos casos de testes.

Especificação:

Esta é a fase de desenvolvimento dos casos de testes, iniciada assim que a fase de planejamento do teste é concluída. Nesta fase a equipe

de testes anota detalhadamente os casos de teste. Também prepara os dados de entrada necessários para o teste. Quando os casos de teste são preparados, eles são revisados pela equipe de garantia da qualidade.

Execução:

Após o desenvolvimento do caso de teste e a configuração do ambiente de teste, a fase de execução dos testes é iniciada. Nesta fase, a equipe de testes começa a executar os casos de testes já preparados na etapa anterior.

Entrega:

Este é o último estágio do ciclo de vida do processo de testes de software, no qual o processo de teste é analisado e o projeto é dado como encerrado.

Cada uma das etapas acima gera documentos do processo de testes. Os mais importantes para o processo de testes são: o guia operacional de testes (GOT), estratégia de testes, planos de testes, roteiros de testes, casos de testes e scripts de testes (RIOS, 2006). Na Tabela 1 conseguimos entender melhor as etapas, subetapas, insumos e produtos do ciclo de vida do processo de software.

Fases dos testes de Validação			
Etapa	Subetapa	Insumo	Produto
1 Procedimentos Iniciais	1.1) Elaborar o Guia Operacional de Teste	- Requisitos do Negócio - Modelos de dados - Diagramas de fluxo de dados - Diagramas de contexto - Outros documentos de desenvolvimento	Guia Operacional de Teste
2 Planejamento	2.1) Estabelecer Estratégia de Testes	- Requisitos do Negócio - Modelos de dados - Diagramas de fluxo de dados - Diagramas de contexto - Documentos de planejamento do sistema - Guia Operacional de Teste - GOT	- Estratégia de Teste - Análise de Riscos do Projeto de Teste
	2.2) Estabelecer Plano de Testes	- Estratégias de Testes - Análise de Riscos do Projeto de Testes - Necessidades de dados de teste - Planejamento do sistema que está sendo desenvolvido	- Plano de testes - Nova versão - Análise de Riscos do Projeto de Testes
	2.3) Revisar Estratégia de Testes	- Requisitos de negócio do sistema - Estratégia de Testes	- Estratégia de Testes revisada
	2.4) Revisar Plano de Testes	- Estratégia de Testes - Plano de Testes	- Plano de Testes revisado
3 Preparação	3.1) Adequar o projeto de testes à Gerência de Configuração e/ou de controle de mudanças	- Arquitetura do ambiente de desenvolvimento - Arquitetura do ambiente de produção - Ferramentas e procedimentos de Gerência de Configuração e de mudança	Registro e controle das diversas versões do produto: funcional, desenvolvimento, produto e operacional
	3.2) Disponibilizar infraestrutura e ferramentas de teste	- Estratégia de testes - Arquitetura básica do ambiente de desenvolvimento - Arquitetura básica do ambiente de produção - Ferramentas de teste	Infraestrutura e ferramentas de teste disponíveis para a equipe de testes
	3.3) Disponibilizar pessoal	- Estratégias de testes - Plano de Testes - Ferramentas de testes - Definição do ambiente de teste	Equipe de testes definida e capacitada

 4 Especificação	4.1) Elaborar casos de testes <ul style="list-style-type: none"> - Estratégias de Testes - Plano de Testes - Documentação técnica do sistema - Necessidades de dados de teste - Posição quanto aos testes já realizados 	Casos de testes (atual e nova versão) Scripts de testes (se usar ferramentas) Especificação das necessidades de dados de testes
 5 Execução	4.2) Elaborar Roteiros de teste <ul style="list-style-type: none"> - Casos de Testes - Planos de Teste - Fluxo de execução dos programas previsto pela equipe de desenvolvimento 	Roteiros de Testes
	5.1) Preparar dados de testes <ul style="list-style-type: none"> - Casos de Testes e Scripts de testes - Roteiros de Testes - Documentação do sistema - Especificação das necessidades de dados de testes - Processos de criação de bases e/ou arquivos de teste 	Bases/Arquivos de teste disponíveis
	5.2) Executar testes <ul style="list-style-type: none"> - Roteiros de Testes - Casos de Testes - Scripts de Testes - Resultados esperados 	Resultados dos testes Relatórios de defeitos encontrados Ajustes no material de testes
	5.3) Solucionar ocorrências de testes <ul style="list-style-type: none"> - Relatórios de defeitos com status a resolver - Resultados dos testes 	Relatórios de defeitos encontrados com status resolvido ou a avaliar
	5.4) Acompanhar a execução dos Casos de Testes <ul style="list-style-type: none"> - Relatórios de defeitos (resumo) - Resultados dos testes (resumo) - Estratégia de Testes - Plano de Testes - Casos de Testes - Roteiros de Testes 	Análise do andamento dos Casos de Testes
	5.5) Elaborar Relatório Final <ul style="list-style-type: none"> - Análise dos resultados de teste - Estratégia de Testes - Resultados de testes - Relatórios de defeitos - Resumo - Plano de Testes 	Relatório Final dos testes

 6 Entrega	6.1) Avaliação e Arquivamento da Documentação	- Documentos de testes	- Relatórios de não conformidade - Relatório final de testes - Documentação arquivada
--	--	------------------------	---

Tabela 1 - Quadro resumido das etapas e subetapas do ciclo de desenvolvimento de testes de software.

Fonte: RIOS; MOREIRA, 2006.

A partir do próximo capítulo estudaremos com mais detalhes as fases que acabamos de conhecer. Mas antes, vamos ver quais são os membros de uma equipe de desenvolvimento de software e identificar seus papéis. A Figura 17 nos mostra os papéis dos profissionais nas etapas do desenvolvimento de software.

Para descobrir o que o cliente quer e documentar os requisitos, é preciso ter na equipe de desenvolvimento um ou mais analistas de requisitos. Assim que os requisitos são documentados, os projetistas entram em cena para descrever o que o sistema faz, possibilitando que os programadores gerem código que traduza o especificado nos requisitos. Logo que o código é gerado, começam os testes, feitos primeiramente pelos próprios programadores e depois por testadores, que buscam encontrar os defeitos que não foram descobertos nos primeiros testes. A equipe de testes, após aprovar as funcionalidades e qualidade do software, trabalha com os clientes para verificar se o sistema está de

acordo com o que ele pediu. Tendo a aceitação, os instrutores treinarão os usuários do sistema (PFLEEGER, 2004).



Figura 17 - Papeis da equipe de desenvolvimento.
Fonte: PFLEEGER, 2004.

Agora que já sabemos o que cada um faz na equipe de desenvolvimento, vamos entender o planejamento dos testes de software, pois assumimos que já temos o GOT em mãos.

4 PLANEJAMENTO DE TESTES DE SOFTWARE

Os testes de software são um processo complexo e, como qualquer outro similar, requerem planejamento. Envolve cronograma e alocação de recursos, planejamento de marcos de verificação para todas as atividades no processo de teste de software, considerando desde a definição do processo de testes até os stakeholders e tempo disponível, além de estabelecer padrões do processo e descrever os testes a serem realizados (SOMMERVILLE, 2018; PEZZÈ, YOUNG, 2008).

Cada etapa do processo de testes precisa ser cuidadosamente planejada para que tenhamos a segurança de que os testes estão sendo realizados de maneira adequada. Como o ciclo de vida do processo de software é independente, pode ser realizado paralelamente a outras atividades de desenvolvimento. Pfleeger (2004, pg 299) aponta seis etapas de teste que devem ser planejadas: "1) estabelecendo os objetivos do teste, 2) projetando os casos de teste, 3) escrevendo os casos de teste, 4) testando os casos de teste, 5) executando os testes e 6) avaliando os resultados dos testes".

O objetivo desta fase é elaborar a estratégia e o plano de testes de maneira a minimizar os principais riscos do negócio e guiar as próximas etapas. O planejamento ocorre de maneira dinâmica, evoluindo durante todo o processo de desenvolvimento do software. Aspectos de aproveitamento de cenários de testes, mecanismos para executar novamente testes e conferir resultados, redução de impacto das mudanças na documentação de testes e de esforço de manutenção devem ser valorizados (BARTIÉ, 2002).

[SAIBA MAIS](#)



Acesse este QR Code para saber mais sobre planejamento de testes de software



4.1 ESTRATÉGIA DE TESTES

Segundo Rios e Moreira (2006) estratégia de testes busca fornecer uma visão geral do projeto de testes e diretrizes para a execução do processo de testes. As prioridades devem ser acordadas com o usuário. A Tabela 2 mostra os insumos, produtos, atividades e os dados para indicadores que esta etapa origina.



- Requisitos de negócios dos sistemas
 - Documento de planejamento (se houver)
 - GOT
-
- Estratégia de testes
 - Análise de riscos do projeto de testes



ATIVIDADES

- Elaboração da análise de riscos do projeto de testes
- Análise das possibilidades de mitigar riscos de negócio
- Avaliação de modelo de dados preliminares, dentre outros
- Identificação de indicadores de qualidade e desempenho
- Estipulação dos produtos a serem verificados
- Elaboração do documento de estratégia de testes
- Estipulação dos tipos de testes a serem realizados na implantação e no desenvolvimento
- Definição das prioridades para os testes dos módulos, visando os de integração
- Identificação de necessidade de testes alfa e beta
- Definição de prioridades para os testes
- Definição da infraestrutura a ser utilizada
- Definição dos ambientes operacionais
- Identificação de necessidade de testes estáticos
- Identificação de modularizações para melhor gestão do processo de testes
- Validação das estratégias de testes com as áreas envolvidas
- Revisão técnica dos produtos da etapa
- Divulgação da estratégia de testes para a equipe
- Identificação das técnicas a serem adotadas para os testes



INDICADORES

- Relatório de defeitos
- Data inicial da elaboração da estratégia de testes
- Data final da elaboração da estratégia de testes
- Horas trabalhadas

Tabela 2 – Estabelecer estratégia de testes.

Fonte: ADAPTADO DE RIOS, MOREIRA, 2006.

O documento de estratégia de testes deve ter pelo menos os campos básicos listados a seguir, porém a organização pode inserir ou modificar campos para que se adaptem às suas diretrizes institucionais, conforme afirmam Rios e Moreira (2006).

Identificação do Projeto de Desenvolvimento

Projeto de desenvolvimento ao qual a atividade de execução dos testes está associada.

Definição do Centro de Custos

Qual centro de custo arcará com os custos do projeto dos testes.

Descrição da Aplicação

Descrição sucinta e em linhas gerais da aplicação que será testada e identificação, caso haja, dos módulos e subsistemas que originarão os casos de testes.

Propósito

Visão geral do projeto de testes conjugada com a forma de relacionamento com o projeto de desenvolvimento associado a ele. Também uma descrição sucinta do documento que define o projeto de testes a fim de que o leitor saiba qual trabalho será executado.

Objetivos

Objetivos e metas a serem alcançados.

Escopo

Definição do escopo dos testes.

Usuário

Identificar os usuários que aprovarão o plano de testes.

Riscos de Negócio

Os riscos do negócio, bem como suas criticidades, devem ser identificados aqui.

Cronograma Básico

Um cronograma em linhas gerais com prazos estipulados, de acordo com o projeto de desenvolvimento.

Ambiente de Teste

Todas as necessidades de ambiente, para realização dos testes, devem ser listadas.

Abordagem dos Testes

Devem ser acordadas as responsabilidades de cada parte para a realização dos testes, bem como a estratégia dos testes de integração (bottom up ou top down).

Datas

Datas de início e término da elaboração do documento de estratégia de testes.

Assinaturas

Assinaturas dos stakeholders e ou áreas envolvidas no projeto.

4.2 PLANO DE TESTES

O plano de testes é o documento principal dos testes de software, pois permite que testes sejam repetidos e controlados e serve de guia para orientar manutenções ou executar testes de regressão. Constam nele informações importantes sobre os stakeholders, os objetivos do teste, os componentes do software a serem testados, os critérios de aceitação e os passos necessários para executar os testes.

O objetivo é organizar as atividades do teste, levando em conta os objetivos deste e incorporando cronograma que foi definido pela estratégia de testes ou prazos do projeto. Também busca demonstrar para os clientes que o software funciona perfeitamente conforme os requisitos especificados (PFLEINGER, 2004).

Cada plano de testes contém as descrições de um ou mais casos de teste. Um caso de teste tem pelo menos um conjunto de atividades a serem executadas e um critério de aceitação, que demonstra se o teste teve sucesso ou não. Na maioria das vezes os planos de teste têm também critérios de entrada ou requisitos e vêm acompanhados de uma breve descrição do item a ser testado e dos objetivos a atingir. Ele é um guia para toda a atividade de testes, explicando quem faz os testes e o porquê de fazê-los, como eles são conduzidos e quando acontecerão.

O processo de testes de software vem avançando nos últimos anos e novos artefatos foram adicionados aos planos de teste, tais como sumário de alto nível e lições aprendidas. O intuito é o de situar o plano e a equipe de testes com relação a outras equipes e ao planejamento do projeto como um todo (SILVA et al, 2016).

O plano de teste também define os recursos de hardware e software necessários. Ele normalmente deve incluir quantidades significativas de contingência, a fim de que desvios no projeto e na implementação possam ser acomodados e a equipe reimplantada em outras atividades. Não é um documento estático, pois evolui durante o processo de desenvolvimento.

Para sistemas pequenos e médios, um plano de teste menos formal pode ser usado, porém há necessidade de um documento formal como apoio do planejamento do processo de teste. Para alguns processos ágeis, como programação extrema (XP), o teste é inseparável do desenvolvimento. Como outras atividades de planejamento, o de testes também é incremental. No XP, o cliente é responsável por decidir quanto esforço deve ser dedicado aos testes do sistema (SOMMERVILLE, 2018).



INSUMOS

- Estratégia de Testes
- Análise de Risco do projeto de testes
- Necessidades de dados de testes
- Planejamento do sistema em desenvolvimento



PRODUTOS

- Plano de testes
- Análise de riscos do projeto de testes



ATIVIDADES

- Elaboração do plano de testes
- Revisão da análise de riscos do projeto de testes
- Alineamento do Plano de Testes com o Planejamento do Sistema
- Aprovação do Plano de Testes
- Testes Unitários
- Testes de Integração
- Testes de Sistema
- Testes de Aceitação
- Definição de métricas para cálculo dos indicadores de desempenho e qualidade
- Identificação do tamanho do sistema a ser testado por pontos de função ou outra métrica
- Estimativa dos pontos de teste ou outra métrica
- Planejamento e preparação do cronograma para as revisões definidas na estratégia de testes
- Previsão de recursos de hardware, software, pessoal e ambiente e estratégia de aquisição e capacitação
- Definição de alocação de responsabilidades para atividades e testes específicos
- Definição de relatórios de acompanhamento do processo de testes e controle de correções de defeitos, caso haja necessidade de complementares aos padronizados
- Revisão técnica dos produtos desta etapa
- Divulgação do Plano de Testes para a equipe
- Definição das regras para a classificação de defeitos com sua severidade, caso não hajam padronizadas



INDICADORES

- Data inicial da elaboração do plano de testes
- Data final da elaboração do plano de testes
- Relatório de defeitos
- Horas trabalhadas
- Tamanho do projeto de teste em pontos de teste
- Tamanho do projeto de teste em pontos de função

Relacionar as atividades de teste e indicar as datas de início e término para os seguintes testes:

*Tabela 3 - Estabelecimento do Plano de Testes.
Fonte: ADAPTADO DE RIOS, MOREIRA, 2006.*

O Plano de Testes, exemplificado na Tabela 3, é o documento que contém o projeto de testes e está alinhado com a estratégia de testes, já definida anteriormente. Aqui começam a ser definidos os casos de testes. Segundo Rios e Moreira (2006), os campos que constam no plano são basicamente os seguintes:

Mensuração

Identifica o tamanho do sistema por pontos de função e é feito pela equipe de desenvolvimento. O tamanho do projeto de testes, em pontos de testes, é calculado utilizando os pontos de função.

Ambiente de Teste

Define o ambiente, que é constituído por: sistema operacional, arquitetura do sistema, identificação dos componentes, forma de acesso ao sistema, linguagens de programação utilizadas, conectividades entre ambientes, etc. Verificar se há outros padrões a serem cumpridos.

Necessidade de pessoal técnico

Mapeia o quantitativo do pessoal e seu perfil, a fim de identificar necessidades de treinamento.

Tipos de Teste

Identifica os tipos de testes que serão executados.

Método de Teste

Identifica como os testes serão executados e auxilia e fornecer informações sobre a necessidade de ferramentas automatizadas para o processo de testes.

Critérios de Aceitação

Documenta informações quantitativas que podem ser usadas como métricas para a aceitação do sistema. Algumas como de controle de qualidade e melhores práticas podem ser definidas como critério mínimo para aceitação do sistema.

Processos Associados

Define processos associados ao projeto de testes, tais como bancos de dados, sistemas legados, sistemas de gestão, etc., que podem afetar o processo de testes. Define para quem serão enviados os documentos gerados no processo de teste.

Cronograma detalhado de testes

Lista as atividades a serem executadas com as datas previstas de início e de conclusão da atividade, como exemplifica a Tabela 4 abaixo e na Figura 18, em software específico para cronograma de projetos.

Atividade	Data de início	Data de término
Estratégia de testes	dd/mm/aaaa	dd/mm/aaaa
Plano de testes	dd/mm/aaaa	dd/mm/aaaa
Casos de testes	dd/mm/aaaa	dd/mm/aaaa
Roteiros de testes	dd/mm/aaaa	dd/mm/aaaa

Tabela 4 - Exemplo de cronograma detalhado de testes.

Fonte: DAPTADO DE RIOS, MOREIRA, 2006.

Nome da tarefa	Duração	Início	Término	Predecessoras
Concepção	11 dias	Seg 17/08/09	Seg 31/08/09	
Definição da Equipe e Tema	3 dias	Seg 17/08/09	Qua 19/08/09	
Entrega da Equipe e Tema para o Professor	1 dia	Seg 24/08/09	Seg 24/08/09	2
Elaboração do Plano de Projeto	4 dias	Ter 25/08/09	Sex 28/08/09	3
Entrega do plano de Projeto	1 dia	Seg 31/08/09	Seg 31/08/09	4
Elaboração	17 dias	Ter 01/09/09	Qua 23/09/09	1
Elicitação de requisitos	5 dias	Ter 01/09/09	Seg 07/09/09	
Reunião para concluir elicitação de requisitos	1 dia	Ter 08/09/09	Ter 08/09/09	7
Reunião para definir escopo e prioridades do projeto	1 dia	Qua 09/09/09	Qua 09/09/09	8
Elaboração do documento de requisitos	9 dias	Qui 10/09/09	Ter 22/09/09	9
Entrega do documento de requisitos	1 dia	Qua 23/09/09	Qua 23/09/09	10
Teste 01	10 dias	Qui 24/09/09	Qua 07/10/09	6
Reunião sobre os testes para o sistema	1 dia	Qui 24/09/09	Qui 24/09/09	
Análise dos testes do sistema	3 dias	Sex 25/09/09	Ter 29/09/09	13
Reunião sobre possíveis falhas	1 dia	Qua 30/09/09	Qua 30/09/09	14

Análise sobre possíveis falhas	2 dias	Qui 01/10/09	Sex 02/10/09	15
Elaboração do plano e projeto de testes de sistema	2 dias	Seg 05/10/09	Ter 06/10/09	16
Entrega do plano e projeto de testes do sistema/aceitação	1 dia	Qua 07/10/09	Qua 07/10/09	17
Análise 01	10 dias	Qui 08/10/09	Qua 21/10/09	12
Definição a arquitetura do sistema	1 dia	Qui 08/10/09	Qui 08/10/09	
Reunião para definição da arquitetura	1 dia	Sex 09/10/09	Sex 09/10/09	20
Elaboração do documento de análise	7 dias	Seg 12/10/09	Ter 20/10/09	21
Entrega do documento de análise	1 dia	Qua 21/10/09	Qua 21/10/09	22
Construção	10 dias	Qui 08/10/09	Qua 21/10/09	12
Reunião para definição do documento do projeto	1 dia	Qui 08/10/09	Qui 08/10/09	
Elaboração do documento do projeto	8 dias	Sex 09/10/09	Ter 20/10/09	25
Entrega do documento do projeto	1 dia	Qua 21/10/09	Qua 21/10/09	26
Implementação	18 dias	Qui 22/10/09	Seg 16/11/09	24
Teste 02	5 dias	Ter 17/11/09	Seg 23/11/09	28
Teste de versão final do sistema	5 dias	Ter 17/11/09	Seg 23/11/09	
Reunião sobre apresentação do projeto final	1 dia	Ter 17/11/09	Ter 17/11/09	
Entrega da versão final do sistema	1 dia	Qua 25/11/09	Qua 25/11/09	

Figura 18 - Exemplo de cronograma.

Fonte: SENAI PR DR, 2020.

Identificação de riscos

Lista os possíveis riscos que podem causar impacto no processo de testes. Devem ser classificados por severidade e impactos causados, usando a mesma classificação: alta, média ou baixa. Os riscos que devem ser identificados são, por exemplo: disponibilidade de recursos, novos processos, nível dos requisitos e dos modelos de dados, falta de qualificação e/ou experiência da equipe de teste, disponibilidade de dados de teste, etc.

Abordagem de teste

Utiliza uma das quatro abordagens que podem ser utilizadas no processo de testes. **Produção completa** utiliza dados reais da produção, **regressão completa**, utiliza um subconjunto de dados da produção, **regressão concentrada** cria um conjunto de dados de teste e garantia de cumprimento do acordo, que cria dados necessários a partir dos reais.

Tipos de testes a serem executados

Identifica os tipos e níveis de testes a serem executados e papéis dos responsáveis em executá-los, conforme indicado pela Tabela 5.

Testes	Responsáveis
Unitário	Desenvolvedores
Integração	Desenvolvedores, DBAs, gerentes do projeto
Sistema	Testadores, desenvolvedores
Aceitação	Testadores, desenvolvedores, usuários
Stress	Testadores, analistas de produção
Performance	Testadores, analistas de produção

Tabela 5 - Testes e seus responsáveis.
Fonte: SENAI PR DR, 2020.

Requisitos de acesso

Define as necessidades de acesso para a equipe de processo de testes, como a bibliotecas, dados de produção e afins. Em cada caso de teste deve constar: equipe de testes, responsável, equipamentos, ferramentas de automação e critérios de aceitação.

Gerência de configurações ou alterações

Cria um fluxo de informações que notifica o testador caso algum programa seja modificado após ter sido testado.

Critérios de término de testes

Define quais serão os critérios que darão os testes como terminados.

Aprovação do plano de testes

Aprova o plano de testes pela área de desenvolvimento e pelo cliente.

Datas

Informa as datas de início e término da elaboração do Plano de Testes.

Assinatura

Coloca a assinatura dos stakeholders na elaboração e aprovação do Plano de Testes.

Na Figura 19 há um Plano de Teste customizado de acordo com as necessidades da empresa.

The screenshot shows a LibreOffice interface with a green header bar labeled 'QUALIDADE' and 'PLANO DE TESTE'. Below the header, there are sections for 'FUNCIONALIDADE' (Envelope), 'TEMPO DESPENDIDO (H)' (1h), 'CONTADOR' (02), and 'CRITICIDADE' (Baixa). The main content area is divided into three sections: 'OBJETO DE TESTE' (with bullet points for validating the assistant for creating an envelope and checking print quality), 'DESCRÍÇÃO DO CASO DE TESTE' (with bullet points for ensuring the application meets configuration requirements and successful envelope printing), and 'PRÉ-CONDIÇÃO' (with bullet points for user data entry and compatible printer selection). A table below lists test steps (P1-P7) and validation steps (V1-V2) for these conditions.

ID	Passo	Procedimento
1	P1	Executar o aplicativo LibreOffice 4.2 (Opção texto)
	P2	Selecionar a opção Texto
	P3	Acessar o menu a opção Envelope (Inserir > Envelope)
	V1	O aplicativo deverá exibir a tela do assistente para Envelope
	P4	Clicar no botão Inserir
2	P5	Se necessário ajustar a largura da caixa de texto do Destinatário e Remetente
	P7	Clicar no ícone da Impressora
	V2	Verificar se o envelope foi impresso corretamente.

Resultado Esperado: As operações deverão funcionar corretamente cumprindo todas as regras acima citadas.

Figura 19 - Plano de Teste de Software.

Fonte: LIBREOFFICE, 2014.

Agora que já sabemos bastantes coisas sobre planos de testes, vamos ver os casos de testes?

4.3 CASOS DE TESTES

Os casos de testes são métodos que buscam identificar todos os cenários possíveis de testes. São especificações mais detalhadas dos testes e cada qual representa uma situação diferenciada e única de comportamento no software, cujo objetivo é identificar um defeito não previsto durante todo o ciclo de vida do desenvolvimento. Detalham o que será testado, como será testado, quais os responsáveis pelos testes, entradas necessárias e saídas esperadas (BARTIÉ, 2002; CARVALHO, 2011).

As técnicas para gerar casos de testes sempre estarão associadas à abordagem funcional ou estrutural. Essa associação é fundamental para que a equipe de desenvolvimento rastreie a origem dos casos de testes e, avalie quais deles deverão ser aplicados quando alguma mudança ocorrer nos requisitos ou estruturas internas originais. As Figura 20 e Figura 21 exemplificam o processo de geração de casos de testes utilizando duas diferentes abordagens.

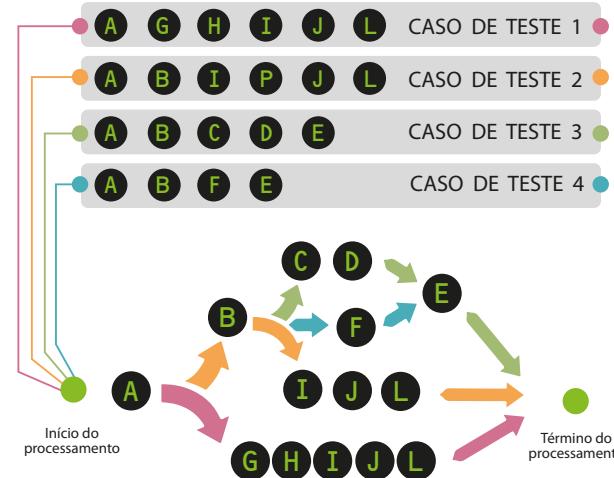


Figura 20 - Abordagem caixa branca para gerar casos de testes.

Fonte: BARTIÉ, 2002.

Os métodos a serem empregados, porém, mudam conforme a abordagem de testes que utilizamos. Se estivermos realizando testes funcionais, baseados em requisitos, utilizaremos certas técnicas de identificação dos casos de testes. Idem para testes estruturais, baseados nas estruturas internas do sistema. O papel dessas formas básicas é fundamental para garantir que o produto se comporte adequadamente. A equipe de testes deve produzir um número suficiente de casos de testes

para cada abordagem existente. Devemos buscar todas as alternativas possíveis e inseri-las em nosso processo de teste de software a fim de refinar e ampliar o nível de cobertura alcançado.

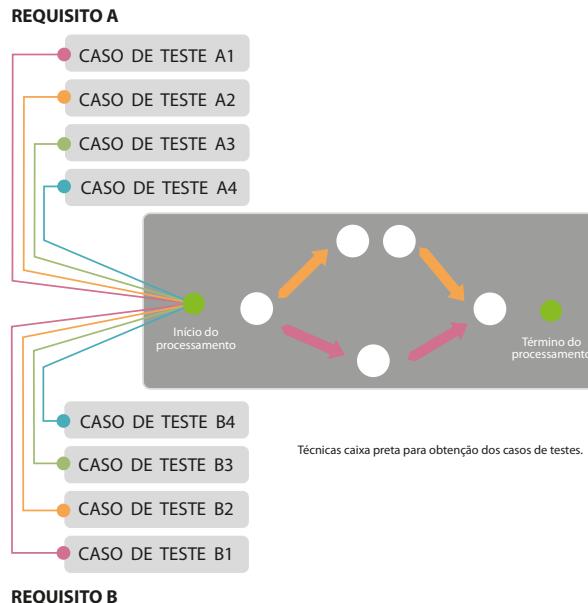


Figura 21 - Abordagem caixa preta para geração de casos de testes.

Fonte: BARTIÉ, 2002.

Pelos casos de testes conseguiremos monitorar os avanços da

qualidade de um software, avaliando os históricos de cobertura dos testes no decorrer dos sucessivos ciclos de interações do desenvolvimento do software. Assim, os casos de testes são o elemento essencial de um processo de teste de um software.

A equipe de testes deve conhecer as principais técnicas de obtenção de casos de testes para que o processo de validação do software tenha resultados efetivos. Devemos ter sempre em mente que a qualidade de um sistema é determinada pelo que conseguimos "garantir" e isso será possível se conseguirmos simular o maior número de cenários possíveis de execução (BARTIÉ, 2002).

Um bom caso de teste deve conter, segundo Carvalho (2011):

- identificação das condições de teste;
- o que testar (identificação dos casos);
- detalhamento da massa de entrada e de saída (dados);
- critérios especiais, quando necessários, para gerar os dados de entrada e saída do teste;
- especificação das configurações de ambiente no qual o teste será executado: sistema operacional, ferramentas necessárias, origem dos dados etc (onde testar);

- como testar: automático/manual (definir o tipo de implementação do teste);
- quando testar (cronograma, em qual fase o teste será executado);
- listar as interdependências, caso existam, entre os casos de teste.

Na Tabela 6, que exemplifica um caso de teste aplicado a um sistema de uma biblioteca, você nota que os casos de teste são descritos de forma bem detalhada. É bastante trabalhoso criar os cenários, pois precisam abranger todas as funcionalidades do software.

Caso de teste Biblioteca da Unidade - Sistema FIEP	
Caso de teste	CT 27 – Buscar para reserva. Botão visualizar o calendário de empréstimo.
Pré-condições	Ter clicado em “Buscar para reserva” na página de empréstimos e estar com a opção de livro disponível no acervo.
Procedimentos	<p>1) Usuário do sistema clica em “Visualizar calendário” representado pelo segundo ícone da direita, de cima para baixo na página de consulta ao acervo.</p> <p>2) O sistema apresenta um calendário do mês, com setas laterais para navegação entre os meses e com as datas acinzentadas dos dias em que o livro está emprestado ou reservado.</p>
Resultado esperado	Carregar o calendário com os dias do mês com as datas dos dias disponíveis em texto normal.
Critérios especiais	Não se aplicam.
Implementação	Manual
Iteração	1ª iteração

Tabela 6 - Exemplo de caso de teste

Fonte: SENAI DR PR, 2020

4.3.1 Análise de Documentos

A análise de documentos é um método bastante efetivo para a identificação dos casos de testes. Nele utilizam-se documentos produzidos com a finalidade de detalhar comportamentos e regras de negócios. Em qualquer documento pode-se encontrar elementos essenciais que auxiliem na identificação de novos casos de testes e no refinamento e ampliação do esforço de planejamento dos testes (BARTIÉ, 2002).

5 NORMAS DE TESTES DE SOFTWARE

Conforme Araújo (2013), há algumas normas que contemplam os testes de softwares. Apesar da IEEE 829 - *Standard for Software and System Test Documentation* - ainda ser usada, ela foi substituída pela ISO/IEC 29119 - *Software Testing Standard*. Também existem normas do International Software Testing Qualifications Board (ISTQB) e do Project Management Institute (PMI), que se encontram no *Project Management Book of Knowledge* (PMBok).

A importância de padronizar, ou normalizar, os testes de software é justamente para termos um padrão, como o próprio nome diz. Pense em uma organização com vários projetos. Cada equipe faz do seu jeito e no decorrer do tempo teríamos inúmeros planos de testes, um diferente do outro dentro da mesma organização. Uma pessoa precisa buscar informações de vários projetos, mas quando os lê tem muita dificuldade de encontrar os dados que precisa. A padronização uniformiza terminologias, documentos, artefatos (RIOS, 2010). Fica muito mais fácil para todos.

Vamos conhecer primeiro o padrão de teste que está em uso, o ISO/IEC/IEEE 29119. Na sequência veremos o IEEE 829, o TMMI e o ISQTB.

5.1. PADRÕES DE TESTE ISO/IEC/IEEE 29119

A *International Organization for Standardization* (ISO) e a *International Electrotechnical Commission* (IEC) são entidades mundiais especializadas em normalização. A Associação Brasileira de Normas Técnicas (ABNT) é o Foro Nacional de Normalização, membro fundador da ISO e membro da IEC. É responsável pela elaboração das Normas Brasileiras (ABNT NBR), que são concebidas pelos Comitês Brasileiros (ABNT/CB), Organizações de Normalização Setorial (ABNT/ONS) e Comissões de Estudos Especiais (ABNT/CEE) (ISO/IEC/IEEE 29119-1, 2013).

A série ISO/IEC/IEEE 29119 tem a finalidade de apresentar um conjunto de normas para padronizar qualquer tipo de teste de software, que

pode ser usado por qualquer organização de desenvolvimento de sistemas. Ela começou a ser elaborada em 2007 e foi baseada na sua predecessora, a norma IEEE 829, que ainda é utilizada e que veremos adiante. Atualmente ela consiste em 5 partes, sendo 3 publicadas em 2013, uma em 2015 e uma 2016, cada qual abordando uma parte específica da atividade de teste. No Brasil somente a parte 1 da norma foi traduzida. A Comissão de Estudo de Teste de Software (CE-21:0007.26) elaborou no Comitê Brasileiro de Computadores e Processamento de Dados (ABNT/CB-21) a norma ABNT NBR ISO/IEC/IEEE 29119-1, cujo conteúdo é idêntico à ISO/IEC/IEEE 29119-1:2013, elaborada pelo *Technical Committee Information Technology* (ISO/IEC JTC 1), *Subcommittee Software and Systems Engineering* (SC 7).

A parte 1 da norma apresenta o vocabulário utilizado e discute os conceitos gerais de testes de software, com exemplos de como aplicá-los. A parte 2 refere-se ao processo de testes, apresentando um modelo genérico a ser usado pelas organizações. A parte 3 trata da documentação dos testes, apresentando modelos e exemplos de documentos produzidos nesta fase. A parte 4 determina as técnicas de projeto de teste que serão utilizadas no processo de projeto e na implementação do teste (parte 2 da norma). Já a parte 5 estabelece uma abordagem baseada em palavras-chave na descrição dos casos de teste

de maneira modular durante o processo de projeto e implementação do teste (parte 2 da norma). Vamos vê-las em mais detalhes.

5.1.1. ISO/IEC/IEEE 29119-1: Conceitos e Definições

Na primeira parte da norma, chamada de ABNT NBR ISO/IEC/IEEE 29119-1 por ter a elaboração brasileira, são especificadas as definições e os conceitos sobre os testes de software, abrangendo os termos de teste e a discussão dos conceitos-chave para que se possa entender a série. São fornecidos também exemplos de aplicação de cada conceito

As considerações seguintes são apontadas nesta parte da norma. Testar um software é um processo composto de atividades que interagem entre si para transformar entradas em saídas. O processo de teste organizacional define e mantém políticas e estratégias de teste aplicáveis aos projetos e funcionalidades da organização. O teste deve ser planejado, controlado e monitorado. Os processos e subprocessos podem ser aplicados a quaisquer tipos, níveis ou fases de testes. Testam-se os itens uma um. Devem ser realizados testes estáticos, que, embora possam ser feitos por outras pessoas envolvidas no projeto, a norma recomenda que sejam feitos por testadores. Esses testes são extremamente importantes no ciclo

de vida do projeto e têm demonstrado serem essenciais para a detecção de defeitos o quanto antes, redução dos custos do projeto e cumprimento do cronograma. São realizados geralmente sem executar o software e podem utilizar ferramentas de análise estática a fim de encontrar defeitos. Já os testes dinâmicos, tratados na parte 2 da norma – Processo de Teste Dinâmico, executam os itens de teste e abrangem as atividades de preparação e acompanhamento. Tanto o teste estático quanto o dinâmico podem validar as atividades de validação e verificação, ao demonstrar evidências objetivas de que os requisitos foram cumpridos e que o sistema pode ser usado pelos usuários. (ISO/IEC/IEEE 29119-1, 2013).

Aqui se esclarece que é impossível testar um software exaustivamente e que os principais objetivos do teste são fornecer informações sobre a qualidade do item de teste e riscos residuais, encontrar defeitos neste item antes de ir para produção e mitigar os riscos de má qualidade do produto. Recomenda-se que o teste se concentre em encontrar o maior número de defeitos possível em um produto do software, o mais cedo possível no ciclo de vida de desenvolvimento, conforme custo e cronograma definidos. Também que múltiplas estratégias de teste sejam empregadas a fim de mitigar os riscos, pois a maioria deles é baseado em heurística, método que pode ser falho.

5.1.1.1. TESTE DE SOFTWARE EM UM CONTEXTO ORGANIZACIONAL E DE PROJETO

Organizações envolvidas no desenvolvimento ou aquisição de produtos de software têm interesse em desenvolvimento, utilizando processos eficazes, eficientes e recorrentes. Para conseguir isso, geralmente desenvolve-se um conjunto robusto de processos de ciclo de vida de software para serem aplicados aos projetos de desenvolvimento por eles realizados. A intenção da norma é ser útil tanto para adoção em toda a organização quanto para uso em projetos específicos. A organização pode adotar o padrão e complementá-lo com procedimentos adicionais, práticas, ferramentas e políticas, conforme a necessidade e também para estar em conformidade com as normas internas da organização.

A norma recomenda que toda e qualquer organização que produza software, independente de seu porte, engaje a alta gerência, expressando o comprometimento em uma Política Organizacional de Teste e também em uma ou mais Estratégias Organizacionais de Teste, que são pilares para todo o software desenvolvido na organização. Essa prática visa dar mais coerência ao teste para que o projeto seja mais eficaz e eficiente (ISO/IEC/IEEE 29119-1, 2013).

O teste de software, como já falado, deve ser planejado, monitorado e controlado. Pode ser tanto um projeto de desenvolvimento quan-

to uma manutenção contínua. Tudo isto ocorre dentro de um contexto, que envolve orçamento, cronograma, escopo, risco, criticidade, cultura organizacional, expectativas dos usuários, disponibilidade do ambiente para os testes. Devido à particularidade de cada projeto, é recomendável que a norma seja utilizada como referência para customização e refinamento dos testes, tendo em vista que nenhuma estratégia, plano, método ou processo funciona para todas as situações.

O plano completo do projeto deve incluir a consideração das atividades de teste a serem realizadas como parte do projeto. Um plano de teste do projeto deve refletir tanto a Política de Teste Organizacional, a Estratégia de Teste Organizacional e os desvios dessas diretrizes organizacionais. Ele também deve explicar as restrições estabelecidas no plano geral do projeto. Um elemento importante do planejamento de teste é a pesquisa das diversas necessidades de teste e o equilíbrio de recursos em todos os testes.

O teste de um projeto de software poderá ser executado por muitos subprocessos de testes, cada qual com um plano de teste que corresponda com a estratégia adotada. Como exemplo, um plano de subprocesso de teste, um plano de teste de sistema ou um plano de teste de desempenho. A Figura 22 mostra de que maneira o teste se adequa a um contexto de leis, políticas e normas da organização, em múltiplas camadas.

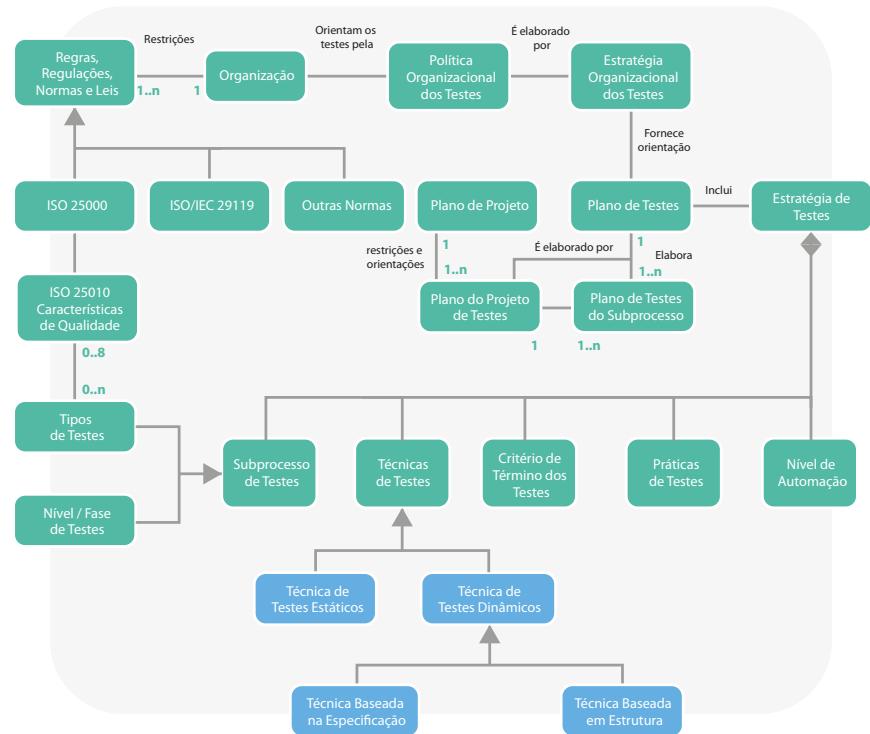


Figura 22 - Diagrama de contexto de teste por múltiplas camadas.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

O plano de teste do projeto descreve a estratégia total para testes e os processos de teste a serem usados. Isso estabelece o contexto de testes para o projeto pela definição dos objetivos, práticas, recursos e cronograma. Também identifica subprocessos de teste que serão aplicados, como por exemplo, testes de sistema e testes de desempenho, e as técnicas de testes, estática ou dinâmica, apropriadas para completar o teste.

Cada plano de teste do subprocesso pode abranger mais do que um nível de teste - por exemplo, o teste de segurança pode abranger diversas fases de testes - e pode tratar mais de um tipo de teste – por exemplo, um plano de testes de sistema que abranja teste funcional e teste de desempenho no nível de teste de sistemas. O plano de teste do subprocesso também descreverá a estratégia para a execução do teste (ISO/IEC/IEEE 29119-1, 2013).

A Figura 23 mostra a implementação dos processos de testes, em particular os níveis de teste e os tipos de testes. Cada nível de teste é uma aplicação específica do subprocesso de teste genérico - por exemplo, fase de teste de componentes, o nível de teste de aceitação. Cada tipo de teste é uma aplicação específica do subprocesso de teste genérico -

por exemplo, testes de desempenho, testes de usabilidade. O diagrama também ilustra a relação entre os tipos de teste e as características de qualidade. O subprocesso genérico pode ser aplicado como um nível/fase de teste (fase de teste de componente, fase de teste de aceitação, etc.), como um tipo de teste (teste de desempenho, teste de usabilidade, etc.), como um subprocesso de teste associado a um nível de teste que pode ter um ou mais subprocessos (por exemplo, teste funcional e de desempenho dentro do testes de sistemas) e um processo do projeto de teste de software, que pode ser composto de vários subprocessos de teste (subprocesso de teste de componente, teste de integração, teste de sistemas, teste de aceitação).

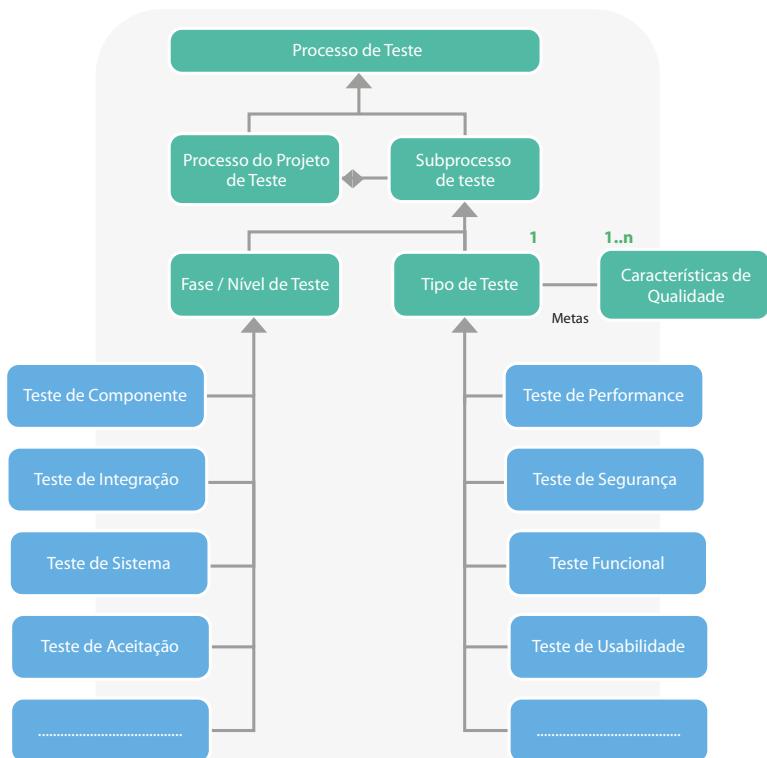


Figura 23 - Relação entre o subprocesso de teste genérico, níveis de teste e tipos de teste.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

A norma utiliza um modelo de processo de 3 camadas, descrito detalhadamente na parte 2 da norma e ilustrado na Figura 24. Há uma camada de gestão organizacional de alto nível que especifica as políticas e estratégias de testes. A camada do meio é referente ao gerenciamento do projeto, das fases e dos tipos de teste. Por fim, a camada inferior define a série de processos de testes dinâmicos.

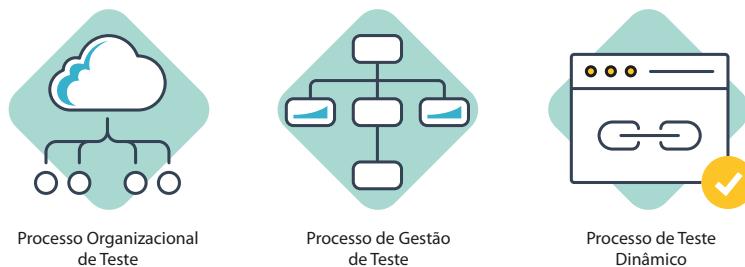


Figura 24 - O relacionamento multicamada entre processos de testes.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

A política de Teste é direcionada aos executivos e gerência sênior, pois ela traduz as expectativas da gestão e a abordagem do teste no viés de negócio e orienta a elaboração e execução da Estratégia Organizacional de Teste. Já esta define os requisitos e restrições dos processos de teste dinâmico e de gestão de testes que suportarão todos os projetos dentro da organização. Alinhada com a Política Organizacional de Teste,

descreve como o teste deve ser executado. O Processo de Gestão de Testes, ilustrado na Figura 25 mostra como deve ser realizada a gestão do teste. Atividades de monitoramento são executadas a fim de garantir que o teste está seguindo dentro do planejamento e também para mitigar riscos ou tratá-los de maneira adequada. Relatórios de Estado do teste são artefatos desta fase. O resultado geral do teste, porém, é documentado no Relatório de Conclusão do Projeto de Teste.



Figura 25 - Processo de Gestão de Teste.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

O teste de um projeto é geralmente dividido em subprocessos de teste e devem ser gerenciados da mesma maneira que o projeto de testes global. Os subprocessos contemplam tanto testes estáticos quanto dinâmicos, estes mostrados na Figura 26 e detalhados na parte 2 da norma.

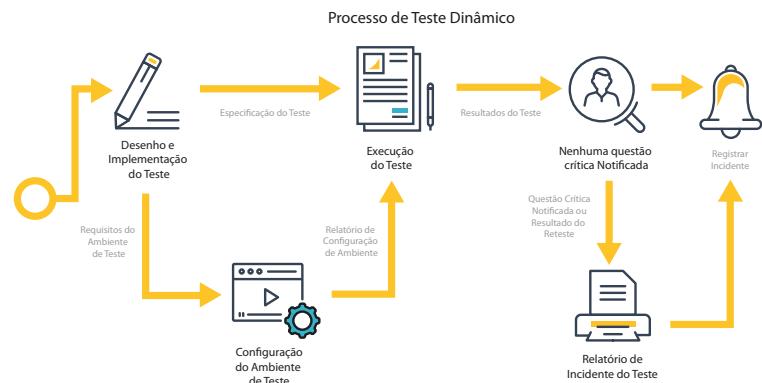


Figura 26 - Processo de Teste Dinâmico.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

5.1.1.2. PROCESSOS DE TESTE GENÉRICOS NO CICLO DE VIDA DO SOFTWARE

O ciclo de vida do software é esperado desde quando é concebido até quando chega ao fim. Ele compreende vários subciclos, de vida dentre os quais se encontra a atividade de testes. A Figura 27 mostra que este ciclo muitas vezes é composto de um ou mais ciclos de desenvolvimento. É uma estrutura que contém processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um software. Um ciclo de vida de desenvolvimento é gerenciado e controlado em um projeto de desenvolvimento (OLIVEIRA, 2017).

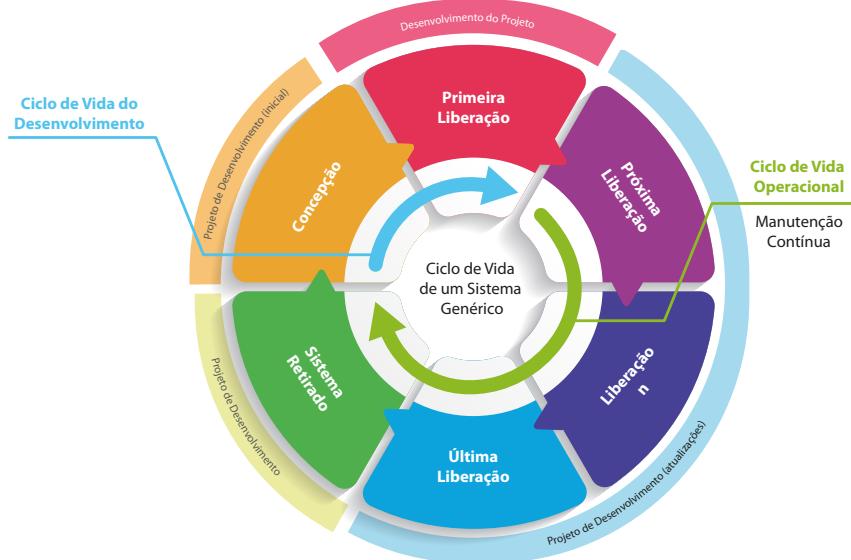


Figura 27 - Exemplo de Ciclo de Vida do Software.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

Em cada subprocesso de desenvolvimento algo é produzido. Desde um documento detalhado e altamente estruturado até um informal. Decisões também podem ser documentadas. Todo item produzido, do componente do sistema de software ao sistema finalizado, é um potencial item de teste.

A preocupação do processo de manutenção contínua é manter um nível aceitável de confiabilidade e disponibilidade do sistema. Isto engloba a produção de novas versões do sistema com prioridade nas correções de defeitos encontrados na operação e, possivelmente, a definição de prioridade nas alterações de funcionalidades.

Em uma organização, processos de apoio são necessários para suportar o ciclo de vida de desenvolvimento de software. Dentre eles estão: "garantia da qualidade, gestão de projeto, gestão de configuração e melhoria dos processos" (ISO/IEC/IEEE 29119-1, 2013 p 26).

A garantia de qualidade é um conjunto de processos planejados e atividades de suporte necessárias para proporcionar a confiabilidade adequada de que um processo ou produto de trabalho cumprirá requisitos técnicos ou de qualidade estabelecidos. Para isto é preciso aplicar métodos, padrões, ferramentas e habilidades que são reconhecidos como prática adequada para o contexto. O processo de garantia de qualidade utiliza os resultados do teste e outras informações para fins de investigação, classificação e para relatar quaisquer problemas na concepção, planejamento ou execução dos processos de engenharia de software (ISO/IEC/IEEE 29119-1, 2013).

A gestão do projeto refere-se aos processos de suporte que são utilizados para planejar e controlar o andamento de um projeto, inclusive a ges-

tão do projeto de teste dentro do projeto global, como ilustra a Figura 28. A estimativa, análise de risco e programação das atividades de teste devem ser consolidadas com o planejamento geral do projeto. O plano do projeto, item informativo do processo de gestão de projetos é uma entrada para o processo de gestão de testes, quando usado para gerenciar o projeto.



Figura 28 - Relação entre o projeto global e o de testes.

Fonte: ISO/IEC/IEEE 29119-1, 2013.

A finalidade da gestão de configuração é estabelecer e manter a integridade dos produtos de trabalho. É uma boa prática para testar um sistema de gerenciamento de configuração do projeto antes da sua utilização operacional para saber se ele atende aos requisitos organizacionais ou de projeto.

O processo de melhoria visa a mudança dos processos para que sejam mais eficazes e eficientes para os negócios da empresa. Os processos de teste e o de melhoria de processos interagem de duas maneiras: os processos de teste fornecem informações para direcionar ações do processo de melhoria e os processos de teste em si podem ser objeto do processo de melhoria (ISO/IEC/IEEE 29119-1, 2013).

Agora que já passamos pelos principais pontos da parte 1 da norma, vamos passar para a seguinte?

5.1.2. ISO/IEC/IEEE 29119-2: Processos de Teste

Esta parte inclui mais detalhes sobre os processos de testes ilustrados na Figura 24. Identifica os processos de teste que podem ser usados para gerenciar, controlar e implementar testes de software nas organizações. Ele fornece descrições comuns dos processos de teste com diagramas descritivos aplicáveis a todos os modelos de teste de software (ISO/IEC/IEEE 29119-2, 2013).

5.1.2.1. MODELO MULTICAMADAS DO PROCESSO DE TESTE

Como mostrado na Figura 24, o processo de teste é baseado em um modelo de processo de três camadas, abrangendo os processos de teste organizacional, de gerenciamento de teste e de teste dinâmico.

O processo de teste organizacional define um processo para a criação e manutenção de especificações de teste organizacional, tais como políticas de teste organizacional, estratégias, processos, procedimentos e outros ativos.

Os processos de gerenciamento de teste definem os processos que abrangem o gerenciamento de testes para o projeto de teste como um todo ou qualquer fase de teste ou tipo de teste dentro de um projeto de teste. Os projetos que fazem parte são: processo de planejamento de teste, processo de monitoramento e controle de teste e o processo de conclusão do teste.

Os processos de teste dinâmico definem os processos genéricos para realizar os testes. Pode ser realizado em uma fase específica de teste – por exemplo, unidade, aceitação –, ou para um tipo específico de teste – por exemplo teste de desempenho, teste funcional – no projeto de teste. Os processos dinâmicos de teste que fazem parte são: processo de projeto de teste e de implementação, processo de configuração e manutenção do

ambiente de teste, processo de execução do teste e processo de relatório de incidentes de teste (ISO/IEC/IEEE 29119-2, 2013).

As camadas do modelo de processo de teste compreendem números variáveis de processo de teste, como mostrado na Figura 29.



Figura 29 - Modelo multicamadas mostrando todos os processos de testes.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

5.1.2.2. PROCESSO DE TESTE ORGANIZACIONAL

O processo de teste organizacional é usado para desenvolver e gerenciar especificações de teste organizacional, que geralmente se aplicam a testes em toda a organização por serem genéricos e não se basearem em projetos específicos como, por exemplo, a Política de Teste e a Estratégia de Teste Organizacionais. (ISO/IEC/IEEE 29119-2, 2013).

A Política de Teste Organacional é um documento voltado para os executivos da organização e descreve o propósito, os objetivos e o escopo geral dos testes dentro da organização. Também define práticas de teste na organização e fornece um *framework* para a elaborar, revisar e melhorar continuamente a Política de Teste e também a Estratégia de Teste da organização e aborda o gerenciamento de projetos de teste.

A Estratégia de Teste Organacional é um documento técnico detalhado que define como o teste é realizado dentro da organização. É um documento genérico que fornece diretrizes para uma série de projetos na organização.

Como mostra a Figura 30, estratégia de teste organizacional deve ser alinhada com a Política de Teste Organacional. O feedback desta

atividade é direcionado para a Política de Teste para possível melhoria de processos. Da mesma forma, os processos de gestão de teste que serão utilizados em cada um dos projetos dentro da organização também precisam estar integrados com a estratégia e política organizacional de teste (ISO/IEC/IEEE 29119-2, 2013).



Figura 30 - Exemplo de Implementação do Teste de Processo Organizacional
Fonte: ISO/IEC/IEEE 29119-2, 2013.

O teste organizacional é representado duas vezes na Figura 30, uma vez para a criação e manutenção da política de teste e a segunda para a estratégia de teste organizacional. Ademais, o processo de gerenciamento de teste é iniciado para o desenvolvimento e implementação do plano de teste. A camada inferior é dedicada ao teste dinâmico sempre que o plano de teste necessitar. O teste dinâmico, por exemplo, abrange testes de unidade, testes de sistema e testes de desempenho, etc. A Figura 31 mostra uma visão geral dos Processos Organizacionais de Teste

As entradas para as atividades nesse processo podem incluir:

- Visões dos principais interessados;
- Conhecimento das práticas atuais de teste dentro da organização;
- Declaração de Missão da Organização;
- Política de TI;
- Política de Gerenciamento de Projetos de TI;
- Política de qualidade
- Política de Teste Organizacional;
- Estratégia para o Teste Organizacional;
- Feedback sobre especificações do teste;
- Planos Típicos de teste e organização;
- Padrões da indústria e / ou governo.

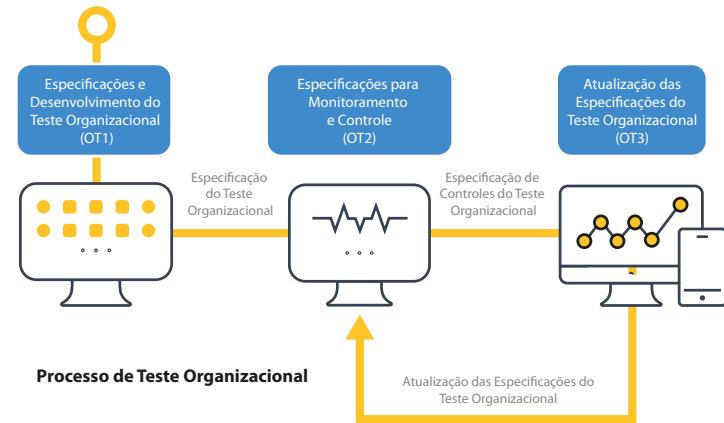


Figura 31 - Processo de Testes Organizacionais.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

SAIBA MAIS



Quer saber mais sobre Processos de Testes Organizacionais? Consulte a norma ISO/IEC/IEEE 29112-2.

5.1.2.3. PROCESSOS DE GERENCIAMENTO DE TESTES

Há três processos de gerenciamento de testes: planejamento de testes, monitoramento e controle de testes e conclusão do teste. Esses processos genéricos do gerenciamento de teste podem ser aplicados no nível do projeto, para gerenciamento de testes em diferentes fases de teste e para gerenciar vários tipos de testes – por exemplo, gerenciamento de teste de usabilidade. Quando aplicados no nível de gerenciamento de teste do projeto, são usados para gerenciar o teste para todo o projeto, baseado em um plano de teste do projeto. Para muitos projetos, cada um dos testes exigirá que os processos de gerenciamento de teste sejam aplicados separadamente ao seu gerenciamento. A Figura 32 mostra os relacionamentos entre os três processos de gerenciamento e suas interações. (ISO/IEC/IEEE 29119-2, 2013).

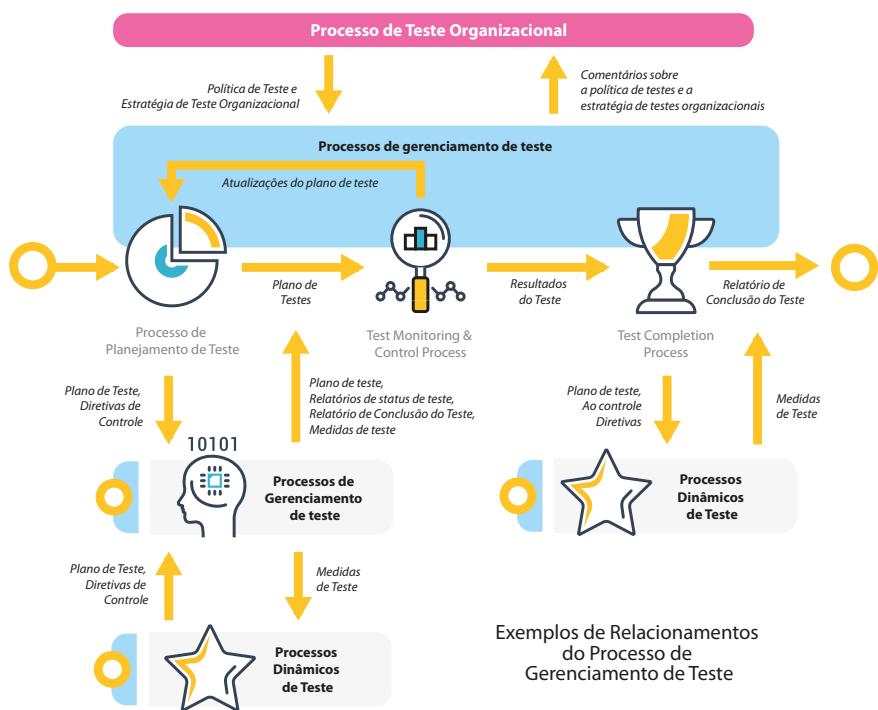


Figura 32 - Exemplo de Relacionamentos de Processos de Gerenciamento de Testes.
Fonte: ISO/IEC/IEEE 29119-2, 2013.

5.1.2.4. PROCESSO DE PLANEJAMENTO DE TESTE

A finalidade do processo de planejamento de teste é elaborar o Plano de Testes. Dependendo de em qual parte do projeto o processo é implementado, pode ser tanto um plano de projeto para uma fase específica quanto para o projeto como um todo.

As atividades mostradas na Figura 33 devem ser realizadas para a elaboração do Plano de Teste. À medida em que o conteúdo do plano de teste se torna disponível devido à realização das atividades definidas, um plano de teste será gradualmente elaborado até que o plano de teste completo seja gravado. Devido à natureza iterativa do processo, caso necessário as atividades mostradas na Figura 33 podem ser revistas antes que o plano de teste completo seja disponibilizado. Geralmente as atividades TP3, TP4, TP5 e TP6 precisarão ser executadas iterativamente a fim de obter um plano de teste aceitável (ISO/IEC/IEEE 29119-2, 2013).

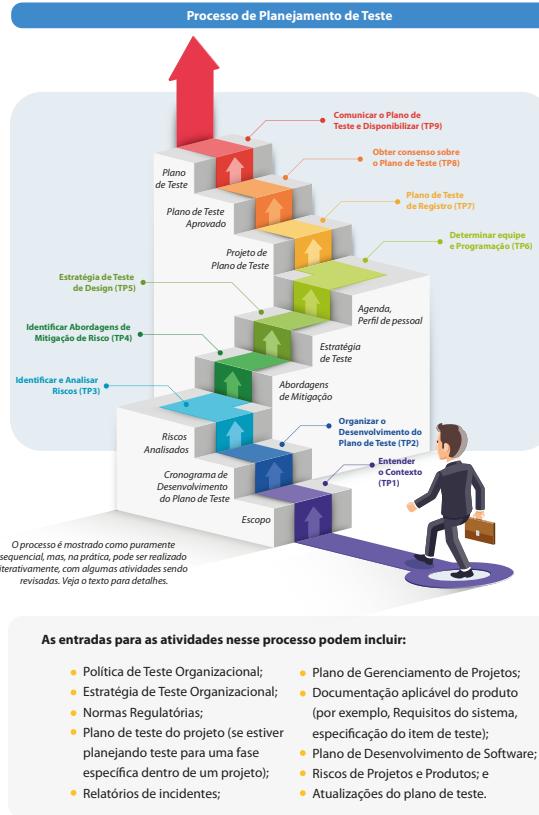


Figura 33 - O Processo de Planejamento de Teste.
Fonte: ISO/IEC/IEEE 29119-2, 2013.

Durante o teste, o plano de teste poderá ser modificado em virtude dos resultados da implementação do plano e do surgimento de novas informações. Por exemplo, podem surgir novos riscos que ameaçam o projeto ou o produto depois que o plano de teste foi disponibilizado. O processo, então, deve ser novamente inserido em Identificar e Analisar Riscos (TP3).

Como resultado da implementação bem-sucedida do Processo de Planejamento de Teste, o escopo do trabalho do projeto de teste é analisado e compreendido, os *stakeholders* que participarão no planejamento do teste serão identificados e informados. Os riscos que podem ser tratados por testes são reconhecidos, analisados e classificados com um nível acordado de exposição ao risco, identificam-se estratégias de teste, ambiente de teste, ferramenta de teste e necessidades de dados de teste. Cada atividade está programada, as estimativas são calculadas e as evidências para justificar as estimativas são registradas. O Plano de Teste é acordado e distribuído a todas as partes interessadas (OLIVEIRA, 2017).

5.1.2.5. PROCESSO DE MONITORAMENTO E CONTROLE DE TESTES

O processo de monitoramento e controle de testes examina se o progresso dos testes está alinhado com o Plano de Teste Organizacional e a política de teste da organização. Se a variação for detectada, o plano de correção deverá ser executado para remover a variação. Esse processo pode ser atribuído ao gerenciamento do projeto ou ao gerenciamento da fase ou tipo de teste único, por exemplo Teste do sistema ou teste de desempenho. Se a variação for detectada em estágios posteriores, ela será aplicada como parte do teste dinâmico. A pessoa encarregada de executar o plano de teste deve implementar as seguintes atividades e tarefas, conforme a Figura 34.

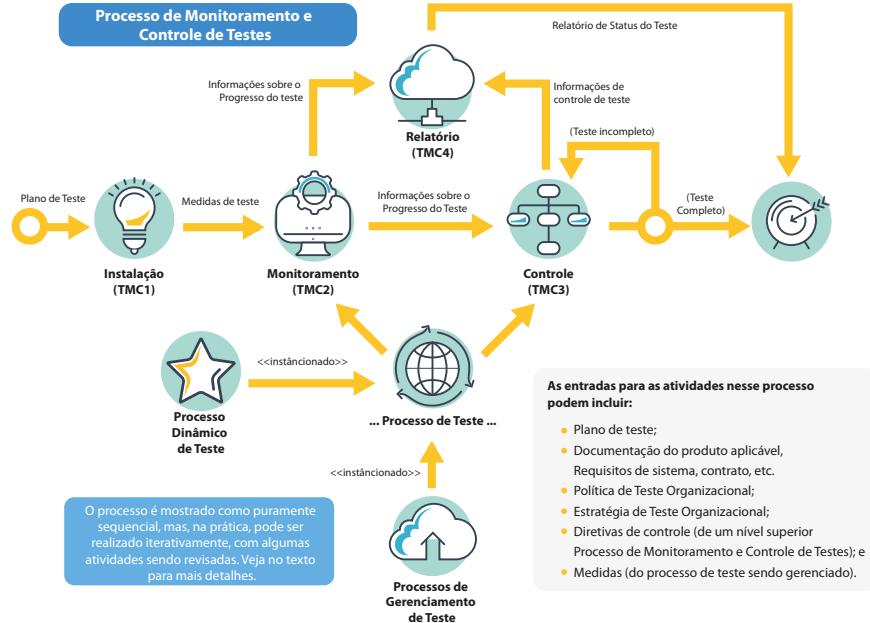


Figura 34 - Processo de monitoramento e controle de testes.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

As atividades indicadas na Figura 34 são as seguintes (ALAQAII; AHMED, 2018):

TMC1: Configuração. Cria medidas e meios adequados para atualizar os riscos com as novas mudanças.

TMC2: Monitoramento. Coleta e registra medidas e as compara com o plano de teste durante o andamento do teste.

TMC3: Controle. Garante a implementação das atividades necessárias para o plano de testes.

TMC4: Relatório. Testa o progresso em relação ao plano de teste e comunica os resultados aos stakeholders dentro de um período de tempo específico, como também os atualiza e relata sobre novos riscos (ISO/IEC/IEEE 29119-2, 2013).

5.1.2.6. PROCESSO DE CONCLUSÃO DO TESTE

O processo de conclusão do teste é um teste de verificação que será executado após a conclusão do teste de atividades. É usado como teste de verificação para realizar os testes feitos no sistema ou no teste de desempenho do projeto geral. É considerado como teste satisfatório, deixando o ambiente de teste em boa situação. A comunicação com os stakeholders deve ocorrer quando o teste geral estiver concluído. A Figura 35 resume as atividades e tarefas envolvidas neste processo.



Figura 35 - Processo de Conclusão dos Testes.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

As atividades indicadas na Figura 35 são as seguintes (ALAQAIL; AHMED, 2018):

TC1: Arquivamento de ativos de teste. Identifica e armazena adequadamente os casos de teste úteis para uso posterior.

TC2: Limpeza do ambiente de teste. Restaura o ambiente para seu estado inicial após a conclusão das atividades de testes.

TC3: Identificação das lições aprendidas. Registra as lições aprendidas durante o projeto.

TC4: Relatório de conclusão do teste. Informações relevantes como plano de teste, resultado de teste e relatório de conclusão de teste devem ser coletadas e relatadas aos stakeholders.

5.1.2.7. PROCESSO DE TESTE DINÂMICO

Este tipo de teste é usado para realizar o teste dinâmico em uma fase específica do teste, por exemplo, testes de unidade, sistema, integração e aceitação. Quatro tipos de processo de teste dinâmico estão disponíveis: projeto e implementação de teste, instalação e manutenção do ambiente de teste, execução e relatório de incidentes

de teste. Esses processos são normalmente considerados como parte da implementação da estratégia de teste dentro do plano de teste na fase de teste, por exemplo, teste do sistema ou teste de performance. A Figura 36 mostra como os quatro processos interagem (ISO/IEC/IEEE 29119-2, 2013).

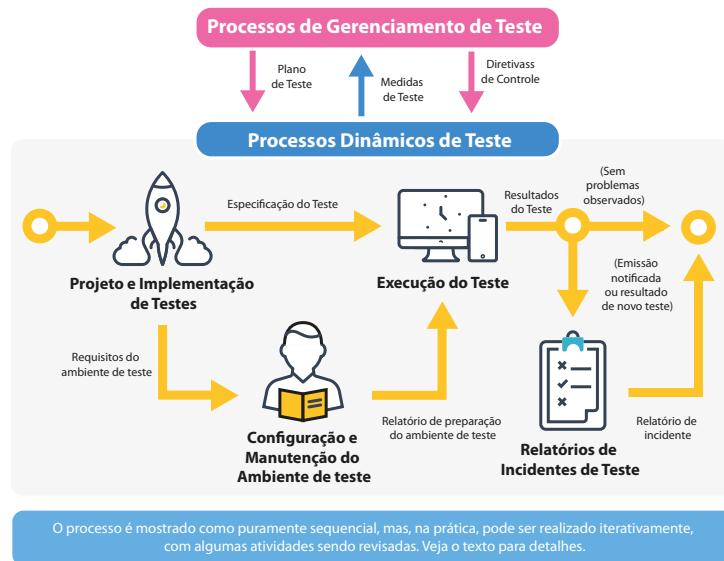


Figura 36 - Processo de Teste Dinâmico.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

5.1.2.8. PROCESSO DE CONFIGURAÇÃO E MANUTENÇÃO DOS TESTES

Nesse processo, casos e procedimentos de teste são obtidos. Normalmente são documentados nas especificações de teste, mas podem ser executados imediatamente. Possivelmente ativos de teste armazenados podem ser usados durante esse processo como teste de regressão. Esse processo pode parar ou reiniciar caso um novo incidente seja relatado. Ademais, exige que o testador aplique uma ou mais técnicas de teste para obter casos ou procedimentos de teste com o objetivo de atingir os critérios de conclusão do teste. Além disso, a iteração pode ocorrer entre as atividades. A Figura 37 resume as atividades e tarefas deste processo.

As entradas para as atividades nesse processo podem incluir:

- Base de teste;
- Plano de teste;
- Estratégia de teste;
- Itens de teste;
- Técnicas de design de teste.

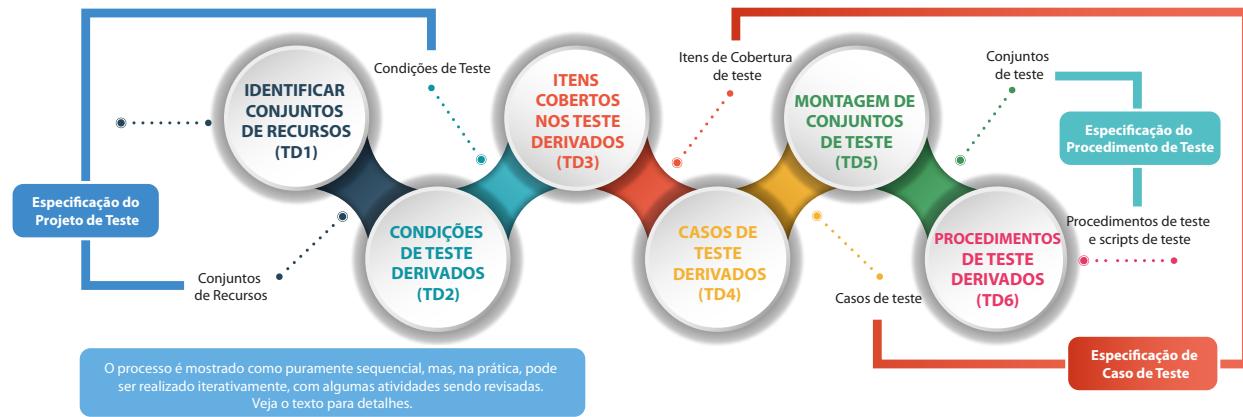


Figura 37 - Processo de Manutenção e Configuração dos Testes.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

As atividades indicadas na Figura 37 são as seguintes (ALAQAIL; AHMED, 2018):

TD1: Identificação de conjuntos de recursos. Analisa a base de teste para entender os requisitos do item de teste, combina os recursos de teste no conjunto de testes e prioriza pelo valor do risco, documenta o conjunto de recursos e comunique-o às partes interessadas.

TD2: Obtenção das condições de teste. Determina a condição de teste para cada caso de teste; prioriza a melhor condição com base no risco, regista a condição do teste na especificação do projeto de teste.

TD3: Obtenção de itens de cobertura de teste. Obtém itens de cobertura de teste através da aplicação de técnicas de design de teste; prioriza itens de cobertura de teste com base no risco, regista itens de cobertura de teste na especificação do projeto de teste e regista a rastreabilidade.

TD4: Obtenção de casos de teste. Determina os valores de pré-condição e entrada para um ou mais casos de teste e prioriza por valor de risco o resultado esperado, registra os itens dos casos de teste na especificação do projeto de teste. Registra a rastreabilidade, obtém a aprovação dos *stakeholders*.

TD5: Montagem de conjuntos de teste. Distribui os casos de teste em um ou mais conjuntos de testes com base em restrições e execução, registra os casos de teste na especificação de procedimentos e também a rastreabilidade.

TD6: Procedimentos de teste de obtenção. Obtém procedimentos de teste do conjunto de testes, ordenado com base nas condições pré e pós condições e dependências. Identifica dados de teste excluídos, prioriza procedimentos de teste com base no risco, registra procedimentos de teste na especificação de procedimentos, registra a rastreabilidade, obtém a aprovação dos *stakeholders*.

5.1.2.9. PROCESSO DE CONFIGURAÇÃO E MANUTENÇÃO DO AMBIENTE DE TESTE

Estabelece e mantém o ambiente em que os testes são executados. Manutenção do ambiente de teste pode envolver alterações com base nos resultados de testes anteriores, onde existem processos de mudança e gerenciamento de configuração, alterações no teste ambientes podem

ser gerenciadas usando esses processos. O objetivo é estabelecer e manter o necessário no ambiente de teste e para comunicar seu status para todas as partes interessadas. Como resultado da realização deste processo, os seguintes itens de informação devem ser produzidos: ambiente de teste; dados de teste e relatório de preparação do ambiente de teste.

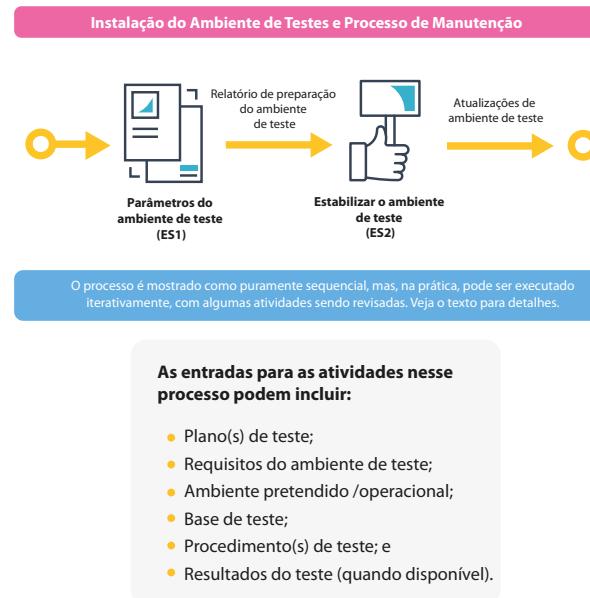


Figura 38 - Processo de Configuração de Ambiente de Teste e Manutenção.
Fonte: ISO/IEC/IEEE 29119-2, 2013.

As atividades indicadas na Figura 38 são as seguintes (ISO/IEC/IEEE 29119-2, 2013):

ES1: Estabelecimento de ambiente de teste. Planeja a configuração do ambiente de teste. Registra e comunica o status do ambiente de teste e os dados aos *stakeholders* relevantes. Mantém o ambiente de teste conforme definido pelos requisitos do ambiente de teste.

ES2: Manutenção de ambiente de teste. Mantém o ambiente de teste como definido nos requisitos do ambiente de teste. Comunica mudanças de status do ambiente de testes aos *stakeholders* relevantes.

5.1.2.10. PROCESSO DE EXECUÇÃO DE TESTE

Esse processo é utilizado para executar os procedimentos de teste gerados como resultado do processo de design e implementação do teste no ambiente de testes definido. Pode ser necessário executar o processo de execução do teste várias vezes, pois todos os procedimentos de teste disponíveis não podem ser executados em iteração única. Além disso, quando um problema é corrigido, o processo de execução do teste deve ser executado novamente. As atividades envolvidas nesse processo são: execução do procedimento de teste, comparação dos resultados do teste e registro da execução do teste, como mostra a Figura 39.



Figura 39 - Processo de Execução dos Testes.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

As atividades indicadas na Figura 39 são as seguintes (ISO/IEC/IEEE 29119-2, 2013):

TE1: Execução dos procedimentos do teste. Um ou mais procedimentos de testes devem ser executados no ambiente preparado. Os resultados atuais para cada caso de teste devem ser observados. Os resultados atuais devem ser registrados.

TE2: Comparação de resultados de testes. Os resultados atuais e esperados de cada caso de teste no procedimento de teste devem ser comparados. O resultado do teste da execução dos casos de teste no procedimento de teste deve ser determinado. Se um reteste for aprovado, deverá ser atualizado o relatório de incidente pelo processo de relatório de incidentes de teste que será especificado no próximo tópico.

TE3: Execução dos registros de teste. A execução do teste deve ser registrada como especificado no plano de testes.

5.1.2.11. PROCESSO DE RELATÓRIO DE INCIDENTES DE TESTE

Esse processo é usado para relatórios de incidentes de teste como resultado da detecção de falhas, itens com comportamento inesperado ou incomum durante a execução do teste ou em caso de reteste. As atividades envolvidas são: analisar o resultado do teste e criar ou atualizar resultados do incidente.

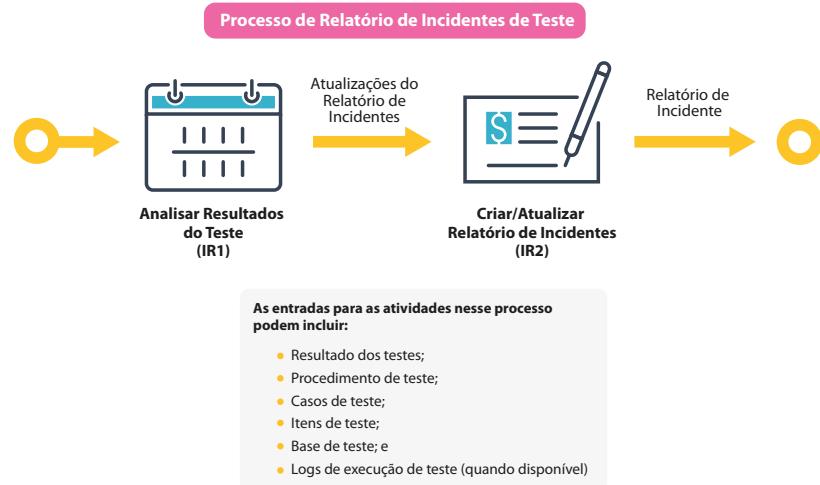


Figura 40 - Processo de Relatório de Incidentes de Teste.

Fonte: ISO/IEC/IEEE 29119-2, 2013.

As atividades indicadas na Figura 40 são as seguintes (ISO/IEC/IEEE 29119-2, 2013):

IR1: Análise os resultados de teste. Reanálise de resultado de teste que se relaciona a um incidente levantado anteriormente e atualização dos detalhes do incidente. Quando um resultado do teste indicar que uma nova questão foi identificada, o resultado do teste

deve ser analisado e será determinado se é um incidente que exige relatórios, um item de ação que será resolvido sem os relatórios de incidentes ou se não será necessária nenhuma ação adicional. Itens que precisam de atenção devem ser destinados à pessoa pertinente para resolução.

IR2: Criação/atualização do relatório de incidentes. As informações que precisam ser registradas sobre o incidente devem ser identificadas e relatadas/atualizadas. O status de incidentes novos/atualizados deve ser comunicado aos *stakeholders* relevantes.

5.1.3. ISO/IEC/IEEE 29119-3: Documentação de teste

Esta parte da norma fala sobre documentação de teste. Ela determina os formulários e modelos de teste de software que podem ser usados pelas organizações, projetos específicos ou atividade única de teste. Contém documentos definidos que são considerados uma saída dos processos de teste. Esses documentos podem ter várias versões e o conjunto de documentação é útil para os profissionais de teste. A Figura 41 lista os documentos do processo de teste.

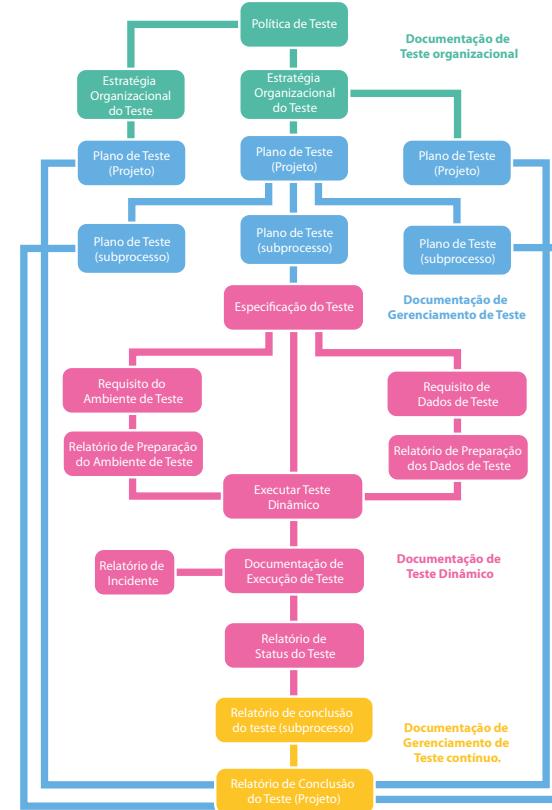


Figura 41 - A Hierarquia da Documentação de Teste.

Fonte: ISO/IEC/IEEE 29119-3, 2013.

A documentação que faz parte desta norma é a seguinte, na ordem apresentada por ela:

- 01. Política de teste
- 02. Estratégia de teste organizacional
- 03. Plano de teste
- 04. Relatório de status do teste
- 05. Relatório de conclusão do teste
- 06. Especificação do projeto de teste
- 07. Especificação de caso de teste
- 08. Especificação do procedimento de teste
- 09. Requisito de dados de teste
- 10. Requisito de ambiente de teste
- 11. Relatório de preparação de dados do teste
- 12. Relatório de prontidão do ambiente de teste
- 13. Log de execução de teste
- 14. Relatório de incidente de teste

Os documentos descritos nesta parte da norma podem ser emitidos em várias versões ao longo do tempo. A norma, porém, não trata as várias versões de documentos tendo em vista que é uma questão de gestão de configuração.

5.1.3.1. DOCUMENTAÇÃO DO PROCESSO DE TESTE ORGANIZACIONAL

As especificações de teste organizacional descrevem informações sobre testes em nível da organizacional e são independentes de projeto. São exemplos de especificações de teste organizacional desenvolvidas no processo de teste organizacional: a política de teste e estratégia de teste organizacional.

A política de teste define os objetivos e os princípios do teste de software que serão aplicados na organização. Especifica o que deve ser feito testando, porém não detalha como o teste é realizado. A política fornece um *framework* para estabelecer, revisar e melhorar continuamente a política de teste da organização.

O conteúdo da Política de Teste traz informações específicas do documento, descrevendo suas origens e histórico. Identifica exclusivamente uma versão do documento. Especifica a organização responsável por elaborar o documento e fornece informações explicativas sobre ele, seu escopo e descreve quaisquer inclusões, exclusões, premissas ou limitações. Apresenta as declarações de política de teste, descrevendo a finalidade, os objetivos, o escopo geral dos

testes dentro da organização, identificando o processo de teste que a organização adotará. Contém também treinamentos e o código de ética organizacional a ser suportado pelos testadores, entre outras informações relevantes.

A Estratégia de Teste Organizacional é um documento técnico, como pode-se ver na Figura 42, que fornece diretrizes sobre como o teste deve ser realizado dentro da organização a fim de alcançar os objetivos estabelecidos na Política de Teste. Para organizações pequenas ou altamente homogêneas, uma única Estratégia de Teste Organizacional pode abranger todos os testes. Uma organização pode ter mais de uma estratégia de teste organizacional tendo em vista que o desenvolvimento pode ser executado de maneiras significativamente diferentes, como por exemplo, produtos críticos de segurança e produtos não críticos, ou se estiver usando modelos de desenvolvimento ágil e modelo V, ou se seus programas são grandes o suficiente para merecer sua própria estratégia (ISO/IEC/IEEE 29119-3, 2013).

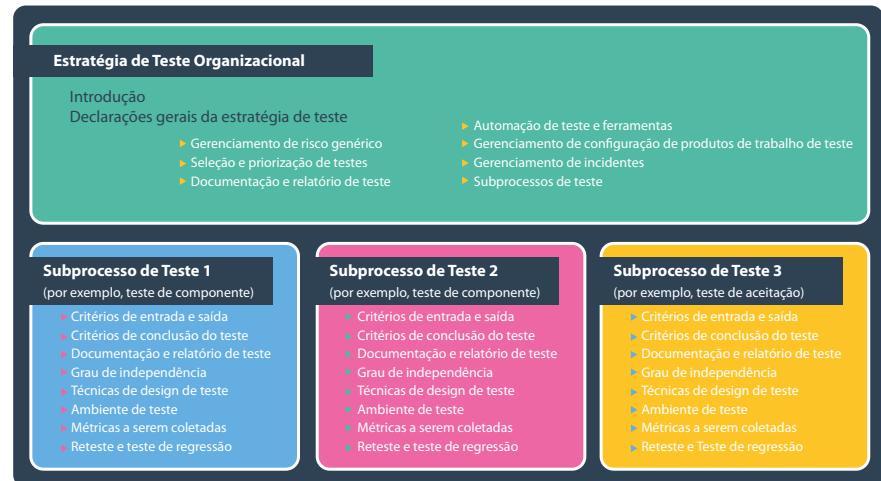


Figura 42 - Exemplo de Estrutura de Estratégia de Teste Organizacional.

Fonte: ISO/IEC/IEEE 29119-3, 2013.

Declarações de estratégia de teste organizacional em todo o projeto, o gerenciamento de risco genérico, identificação da abordagem genérica do gerenciamento de riscos que se espera que seja usada para direcionar as atividades de teste, estão contidas na Estratégia de Testes Organizacional. Ela também descreve a abordagem da organização para selecionar e priorizar a execução do teste. Os procedimentos de teste

consistem em casos de teste prioritários, advindos de conjuntos de recursos priorizados por meio de condições de teste, também priorizadas, e itens de cobertura. A documentação e relatório de teste identificam os documentos que devem ser produzidos durante o teste para o projeto de teste como um todo. A abordagem para testes automatizados dentro da organização também é detalhada e identifica as ferramentas de teste que serão usadas durante o teste, dentre outras informações relevantes (OLIVEIRA, 2017).

5.1.3.2. DOCUMENTAÇÃO DE PROCESSOS DE GERENCIAMENTO DE TESTE

Os documentos desenvolvidos nos processos de gerenciamento de testes compreendem: plano de teste, relatório de status do teste e relatório de conclusão do teste.

O plano de teste fornece um planejamento de teste e documento de gerenciamento de testes. Alguns projetos podem ter um único plano de teste, enquanto projetos maiores podem ter vários. Os planos podem ser aplicados em vários projetos, em um único projeto ou teste específico de subprocesso. Ele descreve decisões tomadas durante o planejamento

inicial e evolui conforme o replanejamento é realizado como parte da atividade de controle. O conteúdo do plano de teste inclui informações específicas do documento, identifica os projetos ou os subprocesso de teste para o qual o plano está sendo escrito e outras Informações contextuais relevantes. Também identifica quaisquer características dos itens de teste que devem ser excluídos especificamente do teste e a justificativa de sua exclusão. Identifica os riscos e fornece um nível de exposição para cada risco com base em seu impacto e probabilidade, apresentando recomendações para tratar os riscos (ISO/IEC/IEEE 29119-3, 2013).

O relatório de status do teste descreve informações sobre a condição do teste que é realizado em um período de relatório específico. Em um projeto ágil, o relatório de status do teste pode não ser um documento escrito, mas discussões em reuniões de iteração complementadas por informações armazenadas em fóruns de atividades e gráficos. Apresenta o progresso que tem sido feito contra o plano de teste. Quaisquer desvios do plano devem ser destacados, com explicações sobre as razões para a ocorrência e descrição de quaisquer ações corretivas. Identifica os fatores que impediam o progresso durante o período do relatório e as soluções correspondentes que foram implementadas para removê-los. Lista os

novos riscos que foram identificados como resultado do monitoramento e controle de teste, bem como alterações aos riscos existentes durante o período de reporte. Descreve ainda, o teste previsto para o próximo período de referência (OLIVEIRA, 2017).

O relatório de conclusão do teste fornece um resumo do teste que foi realizado. Isto pode ser para o projeto como um todo ou para um subprocesso de teste específico. O conteúdo do relatório de conclusão do teste descreve detalhes sobre o que foi testado e quaisquer limitações sobre a maneira como o teste foi realizado.

5.1.3.3. DOCUMENTAÇÃO DE PROCESSOS DE TESTE DINÂMICO

Os documentos desenvolvidos nos processos de teste dinâmico compreendem os seguintes tipos: Especificação de Teste, dividido em: Especificação do Projeto de Teste, Especificação do Caso de Teste e Especificação do Procedimento de Teste, Requisitos de Dados de Teste, Requisitos de Ambiente de Teste, Relatório de Preparação de Dados de Teste; Relatório de Preparação de Ambiente de Teste e Documentação de Execução de Teste, dividida em: Resultados Reais, Resultados do Teste e Log de Execução de Teste e Relatório de Incidentes.

A especificação de projeto de teste identifica os recursos a serem testados e as condições de teste obtidas do teste base para cada uma das características como o primeiro passo para a definição de casos de teste e procedimentos de teste para serem executados. Um conjunto de recursos é um agrupamento lógico dos recursos a serem testados para os itens de teste, que são especificados no Plano de Teste. Podem ser descritos por um ou mais conjuntos de recursos, organizados hierarquicamente, bem como corresponder à arquitetura dos itens de teste, observando a melhor maneira para um teste mais eficiente.

A especificação do caso de teste evidencia os itens de cobertura de teste e os casos de teste correspondentes obtidos da base de teste para um ou mais conjuntos de recursos. A cobertura do teste refere-se ao que é esperado para ser englobado por um caso de teste, de acordo com a técnica de design usada durante a sua obtenção. Um caso de teste especifica como um ou mais itens de cobertura de teste são executados para ajudar a determinar se a parte do item de teste foi ou não implementada corretamente. Especifica os resultados esperados e os comportamentos necessários do item de teste em resposta às entradas que são dadas (OLIVEIRA, 2017).

A especificação do procedimento de teste descreve os casos de teste nos conjuntos de teste selecionados na ordem de execução com as ações associadas e talvez necessárias para configurar as condições prévias iniciais e quaisquer atividades de encerramento de execução posterior.

Os requisitos de dados de teste apresentam as propriedades dos dados de teste necessários para executar os procedimentos de teste definidos na Especificação do Procedimento de Teste. Os requisitos do ambiente de teste caracterizam as propriedades do ambiente de teste necessário para executar os procedimentos de teste definidos na Especificação do Procedimento de Teste.

O relatório de preparação de dados de teste descreve o cumprimento de cada requisito de dados de teste. Por sua vez, o relatório de preparação do ambiente de teste apresenta o cumprimento de cada requisito de ambiente de teste.

Os resultados reais são um registro do resultado da execução de um caso de teste em um procedimento de teste. Eles são comparados com os resultados esperados para determinar o resultado do teste. Os resultados reais nem sempre são registrados formalmente, porém pode haver esta necessidade de documentação dos resultados reais ou a gravação integral dos mesmos. A gravação pode ser feita por uma

ferramenta automatizada durante a execução do teste (OLIVEIRA, 2017).

O resultado do teste é um registro de se uma execução específica caso de teste foi aprovada ou rejeitada, ou seja, se os resultados reais correspondem aos resultados esperados ou se desvios foram observados, ou se a prevista execução de caso de teste não foi possível. O resultado do teste para um caso de teste geralmente é registrado diretamente no procedimento de teste em um espaço reservado para este propósito. O resultado do teste, portanto, não é geralmente considerado como um documento independente.

O registro de execução, ou *log*, de teste grava detalhes da execução de um ou mais procedimentos de teste. Os procedimentos de teste podem ser descritos em listas ou tabelas em um documento, como exemplificado na Figura 43, ou produzidos por uma ferramenta de teste e um banco de dados.

Date	Log entry
01-10-2010	---
02-10-2010	---
03-10-2010	---
04-10-2010	Call from Support - many calls about missing calls for rule-engine - 1294 - 7358715
05-10-2010	Start-up today. There are problems with the communication with TDF. bgh: Problems with internal gf gives problems in the test in udv2
06-10-2010	Finally finished with P0B for this time, now we wait for defects to be handle so that we can get the cases corrected or explained so that we can close them. That is BGH has continued work with message runs and deletion runs and PUJ has
07-10-2010	UDV2 down because of CPR from around noon.
08-10-2010	UDV2 down until appr. 11 because of cpr - it turned out to be a change in interface, which was not communicated. bgh: problems with internal gf gives problems i'n the test in udv2
09-10-2010	P0B has calmed down again, so we are still in. Evaluation meeting went well, hope the results are taken seriously and brough into future projects.
10-10-2010	---
11-10-2010	tstxxx.com is down. And I cannot connect without xxx. Prod is down because of xxx. It is dificult to get something done in xxx, start 0800
12-10-2010	xxx has still got problems - but came up during the day deploy in the evening is very small - but it went well
13-10-2010	Benny is ill until after week 42.
14-10-2010	---

Figura 43 - Exemplo de um log de Execução de um Teste de Sistemas.

Fonte: ISO/IEC/IEEE 29119-3, 2013.

Um incidente de teste é qualquer problema encontrado durante o teste e deve ser documentado. Eles são registrados em relatórios de incidentes. Haverá um relatório de incidente para cada incidente único. Também é chamado de relatórios de defeitos, relatórios de erros, relatórios de falhas, etc. (ISO/IEC/IEEE 29119-3, 2013).

Agora que já vimos toda a parte 3 da norma, que trata da documentação, vamos para a parte 4, que normaliza as técnicas de teste.

5.1.4. ISO/IEC/IEEE 29114-4: Técnicas de teste

Esta parte da norma especifica e identifica técnicas de teste que podem ser usadas com processos de teste na parte 2. O público-alvo desta parte são testadores, gerentes de teste e desenvolvedores, especialmente aqueles que são responsáveis pelo gerenciamento e implementação de software. Aqui, as técnicas de design de teste são definidas para testes baseados em especificação, testes baseados na estrutura e testes baseados na experiência. Nos testes baseados em especificação, a principal fonte de informação usada para projetar casos de teste é a base de teste, por exemplo, necessidades, requisitos, especificações e modelos do usuário. Nos testes baseados em estrutura, o código fonte ou a estrutura do modelo é usada como fonte de informações para desenvolver casos de teste. Nos testes baseados na experiência, a principal fonte de informação é a experiência e o conhecimento dos testadores. Ademais, todos esses tipos de testes são usados para gerar os resultados

finais esperados. Essas técnicas de design de teste não são essenciais, mas consideradas complementares. No entanto, eles são eficazes se aplicados em combinação.

O objetivo das Técnicas de Testes da norma é definir um padrão internacional que abranja técnicas de teste de software que podem ser usadas durante o projeto de teste e processo de implementação, dentro de qualquer organização ou modelo de ciclo de vida de desenvolvimento de software, fornecendo aos stakeholders a capacidade de projetar casos de teste para o teste de software em qualquer organização (ISO/IEC/IEEE 29119-4, 2015).

Os termos: teste baseado em especificação, teste baseado em estrutura também são conhecidos como teste de caixa preta e teste de caixa branca. Tanto o teste de caixa preta quanto o teste de caixa branca estão relacionados à visibilidade de itens de teste na estrutura do software. Como já vimos anteriormente, nos testes de caixa preta, a visibilidade da estrutura interna do item de teste não está presente, enquanto nos testes de caixa branca ela é visível. A Figura 44 mostra todas as técnicas referentes a testes baseados em especificação, testes baseados em estrutura e testes baseados em experiência (ALQAIL; AHMED, 2018).

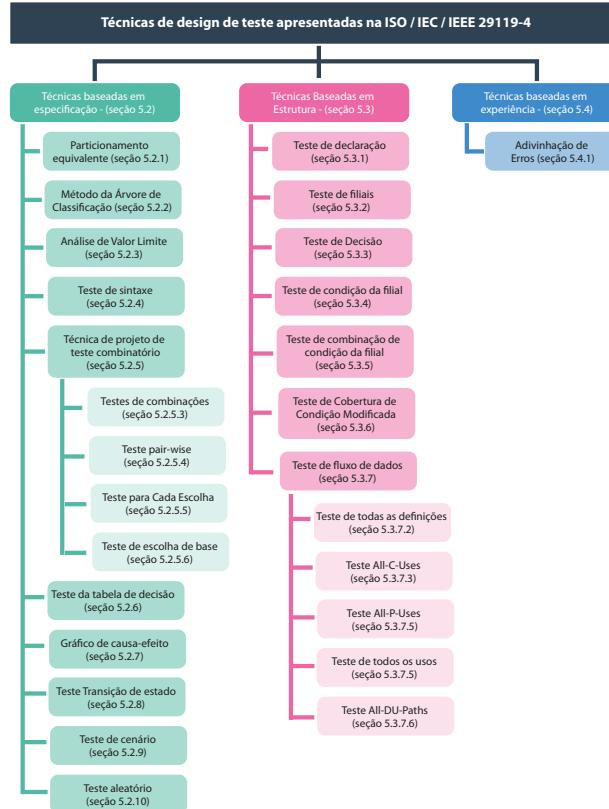


Figura 44 - Conjunto de Técnicas de Projeto de Teste.

Fonte: ISO/IEC/IEEE 20119-4, 2013.

5.1.4.1. TÉCNICAS DE PROJETO DE TESTE BASEADAS EM ESPECIFICAÇÕES

Vejamos com mais detalhes o que a norma fala sobre as técnicas vendo algumas definições dos processos da Figura 44.

5.1.4.1.1. Partição de equivalência

A partição de equivalência usa um modelo do item de teste que divide as entradas e saídas do item de teste em partições de equivalência, também chamadas de "partições" ou "classes de equivalência", onde cada partição de equivalência deve ser declarada como uma condição de teste. Essas partições de equivalência devem ser obtidas da base de teste, onde cada partição é escolhida de modo que todos os valores dentro da partição de equivalência possam razoavelmente ser tratados de maneira semelhante pelo item de teste. Elas podem ser obtidas para entradas e saídas válidas e inválidas (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.2. Método da árvore de classificação

O método da árvore de classificação usa um modelo do item de teste que partitiona as entradas do item de teste para representá-las graficamente como uma árvore chamada árvore de classificação.

As entradas dos itens de teste são divididas em “classificações” (grifo da norma) em que cada uma consiste em um conjunto de classes desmembradas, sem serem sobrepostas. Cada classificação deve ser uma condição de teste. As classes que resultam da decomposição das classificações podem ser divididas ainda mais em subclasses dependendo do nível de rigor exigido no teste. As relações hierárquicas entre classificações, classes e classes secundárias são modeladas como uma árvore, na qual o domínio de entrada do item de teste é colocado como o nó da raiz, as classificações como nós de ramificação e as classes ou subclasses como nós de folha (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.3. Análise do valor limite

A análise do valor de limite usa um modelo do item de teste que partitiona as entradas e saídas do item de teste em uma série de conjuntos e subconjuntos (partições e subpartições) com limites identificáveis, onde cada limite é uma condição de teste. Os limites devem ser obtidos da base do teste. Para os limites de saída, as partições de entrada correspondentes são obtidas com base no processamento descrito na especificação dos itens de teste. As entradas de teste são selecionadas das partições de entrada (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.4. Teste de sintaxe

Teste de sintaxe usa um modelo formal das entradas para um item de teste como base para o projeto do teste. Este modelo de sintaxe é representado como uma série de regras, onde cada uma define o formato de um parâmetro de entrada em termos de "sequências de", "iterações de" ou "seleções entre" (grifos da norma) elementos na sintaxe. A sintaxe pode ser representada em um formato textual ou diagramado. A condição de teste no teste de sintaxe deve ser o modelo total ou parcial das entradas para o item de teste (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.5. Técnicas de projeto de teste combinatório

As técnicas de projeto de teste combinatório são usadas para obter sistematicamente um subconjunto significativo e gerenciável de casos de teste que cubram as condições de teste e os itens de cobertura de teste que são obtidos durante o teste. As combinações de interesse são definidas em termos de parâmetros do item de teste e os valores que esses parâmetros podem tomar. Onde inúmeros parâmetros devem interagir, esta técnica permite uma redução significativa no número de casos de teste necessários sem comprometer a funcionalidade. É composto por quatro testes: todos os testes combinados, teste em pares, cada teste de escolha e teste de escolha de base (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.6. Teste da tabela de decisão

Teste de tabela de decisão usa um modelo de relações lógicas (regras de decisão) entre as condições (causas) e as ações (efeitos) para o item de teste em uma tabela de decisão, onde cada condição booleana define um par de partições de equivalência de entrada para o item de teste: caso "verdadeiro", e caso "falso". Cada ação é um resultado esperado ou uma combinação de resultados para o item de teste expressado como um booleano. E um conjunto de regras de decisão define as relações necessárias entre as condições e as ações (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.7. Representação de grafo de causa e efeito

O grafo de causa e efeito usa um modelo de relações lógicas (regras de decisão) entre causas (por exemplo, entradas) e efeitos (saídas) para o item de teste em forma de grafo em que cada causa booleana define um par de partições de equivalência de entrada para o item de teste: caso "verdadeiro" caso "falso". Cada efeito define uma condição de saída esperada ou combinação de condições de saída para o teste, expresso como booleano (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.8. Teste de transição de estado

O teste de transição de estado usa um modelo dos estados que o item de teste pode ocupar, as transições entre os estados, os eventos que causam transições e as ações que podem resultar das transições. Os estados do modelo devem ser discretos, identificáveis e finitos em número. Uma transição individual pode ser restrita por um guarda de eventos, que define um conjunto de condições que devem ser verdadeiras quando o evento ocorre, para que a transição aconteça. Nos testes de transição do estado, as condições do teste podem ser todos os estados do modelo de estado, todas as transições do modelo de estado ou todo o modelo de estado, dependendo dos requisitos de cobertura dos testes. O modelo pode ser representado como um diagrama de transição de estado ou uma tabela de estado (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.9. Testes de cenário

O teste de cenários usa um modelo das sequências de interações entre o item de teste e outros sistemas. O objetivo é testar os fluxos de uso envolvendo o item de teste. As condições de teste devem ser uma sequência de interações, ou seja, um cenário, ou todas as sequências de interações. O teste de cenário deve incluir a identificação do cenário

“principal”, que é a sequência típica de ações que se espera do item de teste ou uma escolha arbitrária quando nenhuma sequência típica de ações é conhecida e de um cenário “alternativo”, que representa os cenários que não são o principal (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.1.10. Teste aleatório

Teste aleatório usa um modelo do domínio de entrada do item de teste que define o conjunto de todos os valores de entrada possíveis. Deve ser escolhida uma distribuição de entrada para a geração de valores de entrada aleatórios. Todo o domínio de entrada deve ser a condição de teste para testes aleatórios (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.2. TÉCNICAS DE PROJETO DE TESTE BASEADAS EM ESTRUTURA

Vejamos com mais detalhes o que a norma fala sobre as técnicas baseadas em estrutura, vendo as principais definições dos processos da Figura 44.

5.1.4.2.1. Teste de declaração

Um modelo do código fonte do item de teste que identifica declarações como executáveis ou não executáveis deve ser obtido. Cada declaração executável deve ser uma condição de teste. Cada declaração executável deve ser um item de cobertura de teste, ou seja, os itens de cobertura de teste são os mesmos que as condições de teste. Portanto, nenhuma ação adicional é necessária nesta etapa para esta técnica (ISO/IEC/IEEE 29119-4, 2015).

SAIBA MAIS



A cobertura do teste é definida como uma métrica no Teste de Software que mede a quantidade de testes executados por um conjunto de testes. Isso inclui a coleta de informações sobre quais partes de um programa são executadas quando se executa um conjunto de testes para determinar quais ramificações de instruções condicionais foram tomadas. Ou seja, é uma técnica para garantir que os testes estejam realmente testando o código ou quanto do seu código você usou ao executar o teste.

5.1.4.2.2. Teste de ramificação

Um modelo de fluxo de controle que identifica as ramificações no fluxo de controle do item de teste deve ser obtido. Cada ramificação no modelo de fluxo de controle deve ser uma condição de teste. Uma ramificação é uma transferência condicional ou incondicional de controle de qualquer nó no modelo de fluxo de controle para qualquer outro nó ou uma transferência de controle de ponto de entrada, quando há mais de um. O teste de ramificação completo que cobre 100% de todas as ramificações requer todos os arcos (links ou bordas) no gráfico de fluxo de controle a ser testado, incluindo instruções sequenciais entre um ponto de entrada e saída que não contém decisões. O teste de filial pode exigir testes de ramificações condicionais e incondicionais, incluindo pontos de entrada e saída para um item de teste, dependendo do nível de cobertura de teste exigido (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.2.3. Teste de decisão

Um modelo de fluxo de controle do item de teste que identifica as decisões deve ser obtido. As decisões são pontos no item de teste em que dois ou mais resultados possíveis podem ser tomados pelo fluxo de controle. As decisões típicas são usadas para seleções simples,

para decidir quando sair de laços de repetição e em declarações de casos. Neste teste, cada decisão no modelo de fluxo de controle deve ser uma condição de teste (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.2.4. Teste de condição de ramificação

Um modelo de fluxo de controle do item de teste que identifique decisões e condições deve ser obtido. Decisões são pontos nos itens de teste quando dois ou mais resultados entram no fluxo de controle. No teste de condição de ramificação, todos os valores booleanos (verdadeiro ou falso) das condições e os resultados das decisões devem ser identificados como itens de cobertura. Tanto o Teste de Combinação de Condição de Ramificação quanto o Teste de Cobertura de Decisão Modificado seguem os mesmos princípios do teste nesta seção abordado (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.2.5. Teste de fluxo de dados

No teste de fluxo de dados, um modelo do item de teste deve ser obtido que identifique caminhos adjacentes do fluxo de controle a partir do item de teste, dentro do qual cada definição de uma determinada variável está conectada ao seu uso subsequente da mesma variável e dentro de que não houve uma redefinição que interferiu no valor da variável. Um "uso" é

uma ocorrência de uma variável que não possui um novo valor. Os "usos" podem ser mais distinguidos como "p-uses" (uso-predicado) ou "c-uses" (uso-computação). Uma p-use indica o uso de uma variável na determinação do resultado de uma condição (predicado) dentro de uma decisão, como, por exemplo, um laço while, se então, etc. Um uso c ocorre quando uma variável é usada como uma entrada para a computação da definição de qualquer variável ou de uma saída (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.3. TÉCNICAS DE PROJETO DE TESTE BASEADAS NA EXPERIÊNCIA

Veremos aqui com mais detalhes o que a norma fala sobre as técnicas baseadas em experiência, vendo as principais definições dos processos da Figura 44.

5.1.4.3.1. Erro de adivinhação

A adivinhação de erros envolve o projeto de uma lista de verificação de tipos de defeito que podem existir no item de teste, permitindo que o testador identifique entradas para o item de teste que podem causar falhas, se esses defeitos existem no teste item. Cada tipo de defeito deve ser uma condição de teste. A lista de verificação dos

tipos de defeito pode ser obtida por taxonomias de erros conhecidos, informações contidas nos sistemas de gerenciamento de incidentes, do conhecimento ou experiência de um membro da equipe de testes ou outros *stakeholders*, compreensão dos testadores sobre os itens de teste ou itens de teste semelhantes (ISO/IEC/IEEE 29119-4, 2015).

5.1.4.4. MEDIDA DE COBERTURA DE TESTE

As medidas de cobertura definidas nesta parte da norma são baseadas em diferentes graus de cobertura que podem ser alcançados por técnicas de projeto de teste. Os níveis de cobertura podem variar de 0% a 100%. Em cada cálculo de cobertura, uma série de itens de cobertura de teste pode ser inviável, se não for executável ou impossível de ser coberto por um caso de teste (ISO/IEC/IEEE 29119-4, 2015).

SAIBA MAIS



Consulte a parte 4 da norma ISO/IEC/IEEE 29119 para ver exemplos de vários testes que vimos aqui. Eles estão nos anexos da norma.

5.1.5. ISO/IEC/IEEE 29119-5: Teste orientado por palavra-chave

O Teste Orientado por Palavra-chave é uma abordagem de especificação de teste geralmente usada para auxiliar a automação de teste e a criação da sua estrutura. Pode ser usado se nenhuma abordagem de automação de teste estiver planejada ou existir. Esse tipo de teste pode ser aplicado a todos os níveis de teste, como componentes e sistemas, ou mesmo para tipos de testes, como testes funcionais e testes de confiabilidade. As vantagens de aplicar o teste orientado por palavras-chave para o sistema são: torna o sistema mais amigável, comprehensível, sustentável, reutilizável, dá suporte à automação e diminui os custos. (ISO/IEC/IEEE 29119-5, 2016)

Esta parte da norma fornece definições e restrições eficientes para testes orientados por palavras-chave. Introduz abordagens como referência para implementar testes orientados por palavras-chave. Ela também traz um *framework* para a estruturação de testes orientados por palavras-chave. Isso permite que os testadores compartilhem seus itens de trabalho, casos de teste, dados de teste, palavras-chave ou especificações de teste. Na figura pode-se ver os relacionamentos entre as entidades que compõe o teste orientado.

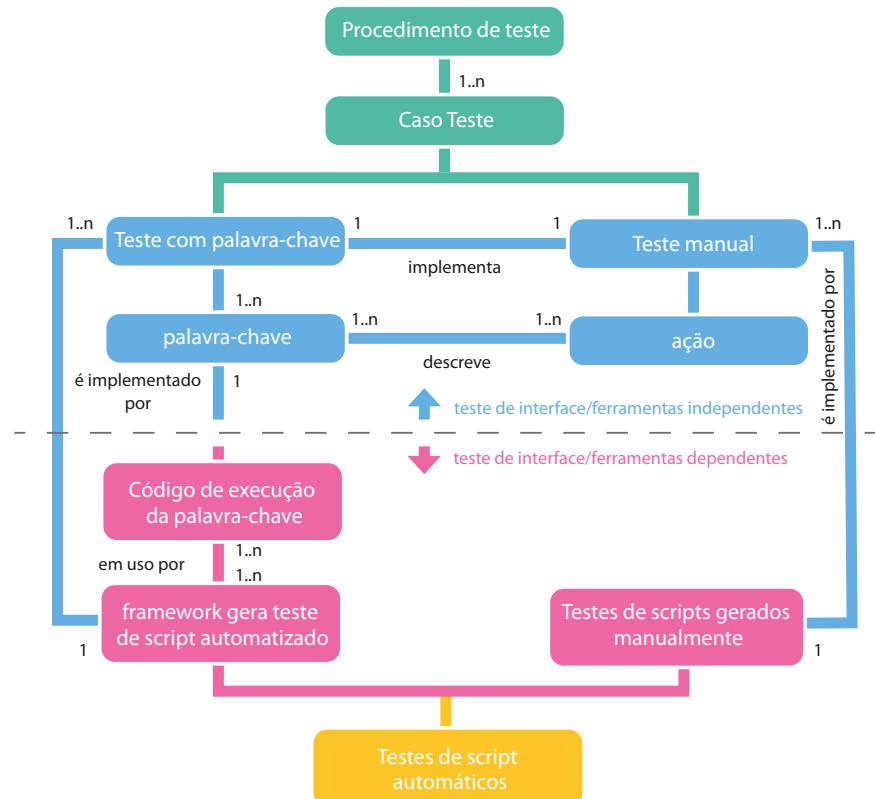


Figura 45 - Relações entre Entidades do Teste orientado por Palavras-Chave.

Fonte: ISO/IEC/IEEE 29119-5, 2016.

Além disso, esta parte fornece definições de interfaces e formato de troca de dados de diferentes fornecedores para garantir uma integração do sistema. Inclui, também, definições de diferentes níveis de palavras-chave hierárquicas, com explicações e orientações sobre como usá-las. Por fim, é fornecida uma lista de exemplos e palavras-chave técnicas de baixo nível, por exemplo “InputData” ou “checkValue” para determinar o caso de teste em qualquer nível técnico específico. Também pode ser combinado a partir da palavra-chave em nível de negócios, se necessário (ALQAIL; AHMED, 2018).

5.2. IEEE 829

A norma, ou padrão, IEEE 829 era amplamente utilizada para a documentação de software, antes da entrada da ISO/IEC/IEEE 29119, em especial a parte 3, que trata especificamente de documentação. Como ela ainda não foi traduzida para o português, muitas empresas ainda adotam a IEEE 829. Na versão da norma anterior a 2008 o foco era apenas os documentos de testes, porém nesta ele passa a ser o processo de teste em si.

O objetivo do padrão é descrever recursos, escopo, abordagem e cronograma das atividades de teste de software, identificar os itens e funcionalidades a serem testados, tarefas a serem executadas e pessoas responsáveis por cada tarefa, além dos riscos associados ao projeto. Ele baseou-se nas diretrizes de plano de projeto do *Project Management Institute* (PMI), ao considerar que o teste de software precisa ser encarado como um projeto (RIOS, 2010).

O escopo do padrão é dar suporte aos processos relacionados ao desenvolvimento e ao teste de software, tais como fornecimento, aquisição, produção, manutenção e desenvolvimento nas organizações. Para isto, ele propõe que sejam utilizados documentos padronizados no **planejamento dos testes**, na **especificação dos testes** e nos **relatórios dos testes**, nesta ordem, conforme o ciclo de vida da Figura 46.

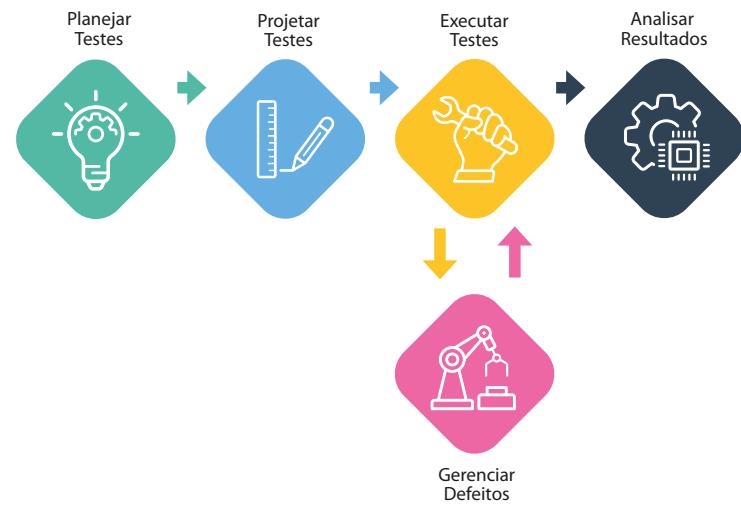


Figura 46 - Ciclo de Vida do Projeto de Teste de Software.

Fonte: RIOS, 2010.

5.2.1. Documentação da norma

A norma descreve os documentos básicos que são utilizados em um processo de teste. A organização pode elaborar outros documentos, além destes, para adequação às suas políticas internas, bem como customizar partes dos documentos da norma. A aplicação da norma independe da metodologia de teste escolhida para uso pela organização.

São os seguintes documentos que a norma define, segundo Rios (2010), e que podem ser vistos na Figura 47:

- Plano Máster de Teste
- Plano de Teste
 - Plano de Teste de Aceitação
 - Plano de Teste de Sistema
 - Plano de Teste de Integração de Componentes
 - Plano de Teste de Componente ou de Teste Unitário

- Projeto de Teste de Aceitação, Sistema, Integração e Unitário
- Casos de Teste de Aceitação, Sistema, Integração e Unitário
- Procedimentos de Teste de Aceitação, Sistema, Integração e Unitário
- Relatórios de Execução dos Testes
 - Relatório de Estado de Teste
 - Relatório de Teste (sumário)
 - Relatório de log de Teste
 - Relatório de Anomalias
- Relatório Máster de Teste (integra os relatórios listados abaixo)
 - Relatório de Teste de Componente
 - Relatório de Teste de Integração de Componentes
 - Relatório de Teste de Sistema
 - Relatório de Teste de Aceitação

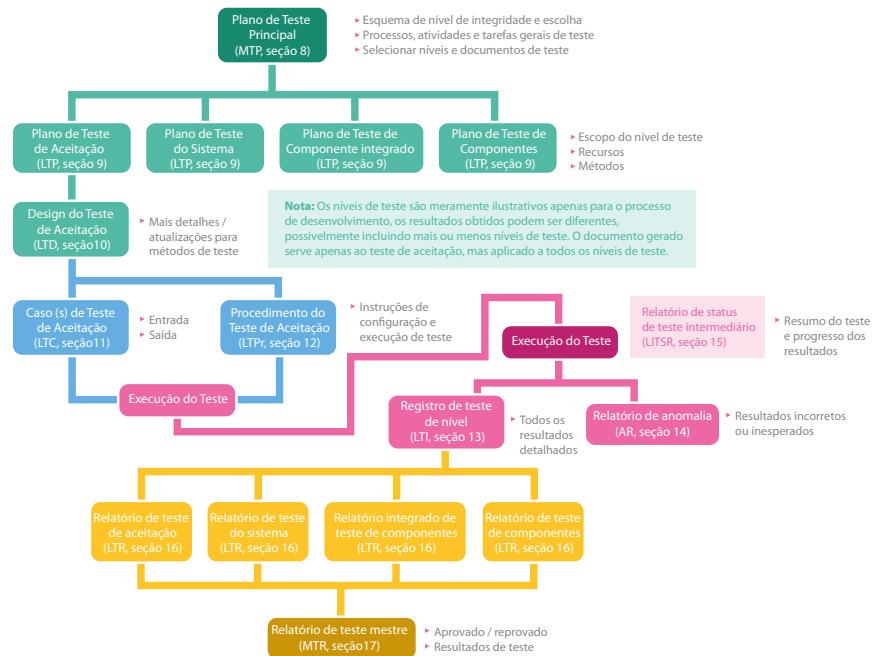


Figura 47 - Visão Geral da Documentação de Teste.

Fonte: IEEE 829, 2008.

Vamos ver com mais detalhes a documentação.

SAIBA MAIS



Quer entender mais sobre as abreviações usadas na Figura 47? Consulte a norma IEEE 829-2008. Todas as informações detalhadas estão nela.

5.2.1.1. PLANO MÁSTER DE TESTE (MTP)

Seu objetivo é planejar e gerenciar, de maneira estratégica e tática, os testes em vários níveis, mantendo seu monitoramento e controle. Ele descreve cada parte de um projeto de testes. Rios(2010), porém, afirma que não há necessidade de existir um Plano Máster quando há somente um Plano de Teste, pois a finalidade dele é de organizar o projeto de teste e não burocratizá-lo. Na Figura 48 pode-se ver todas as partes que compõem o documento.

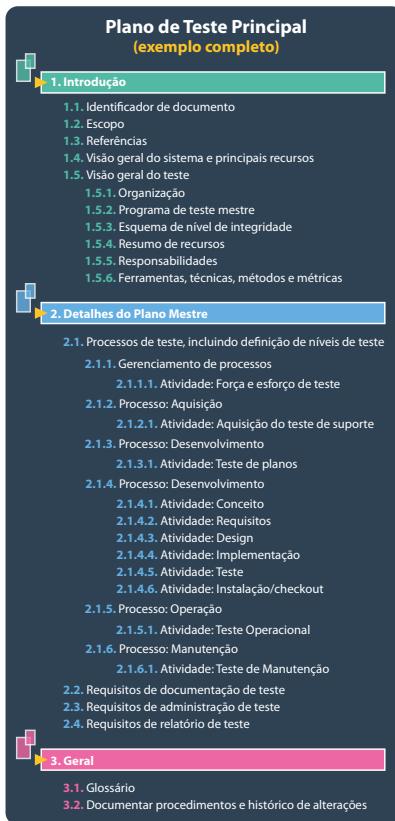


Figura 48 - Exemplo de Esboço do Plano MÁSTER de Teste.

Fonte: IEEE 829, 2008.

5.2.1.2. NÍVEL DE PLANO DE TESTE (LTP)

É um documento que especifica o escopo, abordagem, recursos e cronograma das atividades para cada nível de testes especificado. Conforme o tamanho do projeto, pode haver inúmeros planos de teste. Geralmente são separados por módulos, requisitos ou funcionalidades. Eles podem ser também ser classificados por níveis de testes, por exemplo Plano de Teste de Componentes, Plano de Teste de Integração de Componentes, Plano de Teste do Sistema e Plano de Teste de Aceitação. Nele é preciso identificar os itens que estão sendo testados, os recursos que serão testados, as tarefas de teste que serão executadas, os responsáveis por cada tarefa e os riscos associados (IEEE 829, 2008; RIOS, 2010).

Na maioria dos projetos, existem diferentes níveis de teste que requerem diferentes recursos, métodos e ambientes. É aconselhável descrever cada nível em um plano separado. Planos de teste de nível diferente podem exigir um uso diferente dos tópicos de conteúdo da documentação, cujo exemplo consta na Figura 49 .

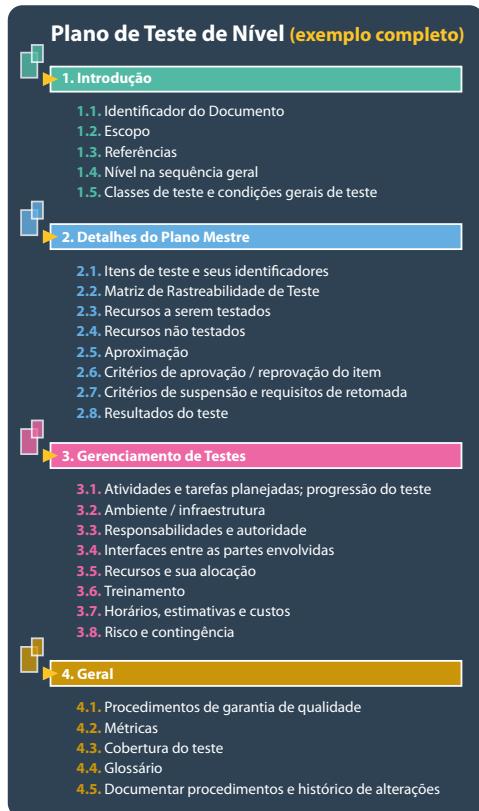


Figura 49 - Exemplo de Esboço de Nível de Plano de Teste.

Fonte: IEEE 829, 2008.

5.2.1.3. NÍVEL DE PROJETO DE TESTE (LTD)

O objetivo deste documento é especificar quaisquer refinamentos da abordagem de teste, já definida no plano de teste, a fim de identificar requisitos e funcionalidades a serem testados além de outros testes associados necessários. Este documento pode estar agrupado no Plano de Testes, em projetos pequenos. A Figura 50 mostra um exemplo do documento.



Figura 50 - Exemplo de Esboço de Nível de Projeto de Teste.

Fonte: IEEE 829, 2008.

5.2.1.4. NÍVEL DE CASOS DE TESTE (LTC)

A finalidade deste documento é definir, com detalhes, as informações necessárias no que se refere às entradas e saídas do software que será testado. Ele deve conter todos os casos de teste identificados pelo segmento nível de caso de teste. Define a unidade a ser testada pelo testador, independente do teste ser automático ou manual.

Tendo em vista que um caso de teste pode ser referenciado por vários projetos de teste de nível usados por diferentes grupos durante muito tempo, informações específicas suficientes devem ser incluídas no caso de teste para permitir a reutilização (IEEE 829, 2008; RIOS, 2010). A Figura 51 mostra um exemplo do documento.



Figura 51 - Exemplo de Esboço de Nível de Casos de Teste.

Fonte: IEEE 829, 2008.

5.2.1.5. NÍVEL DE PROCEDIMENTO DE TESTES (LTPR)

O propósito do nível de procedimento de testes é especificar os passos necessários para executar um conjunto de casos de teste ou a fim de avaliar um conjunto de recursos. A Figura 52 traz um exemplo do documento.

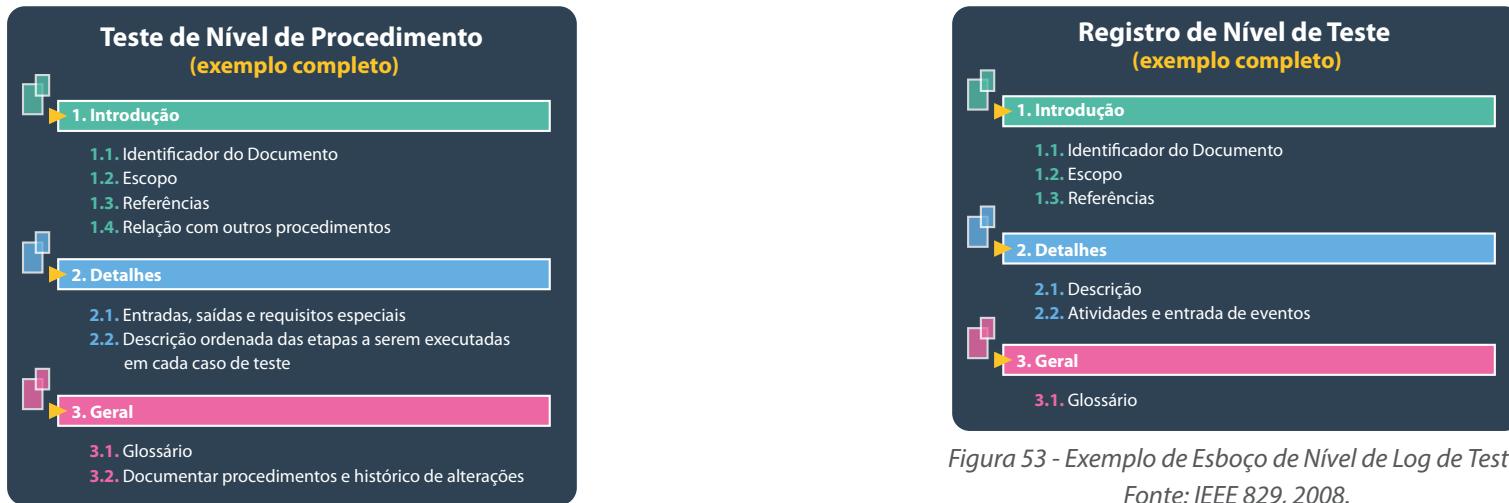


Figura 52 - Exemplo de Esboço de Nível de Procedimento de Teste.

Fonte: IEEE 829, 2008.

5.2.1.6. NÍVEL DE LOG DE TESTE (LTL)

O objetivo do nível de log de teste é fornecer um registro cronológico das informações relevantes e ocorrências, podendo identificar quem fez algo e quando foi feito (IEEE 829, 2008; RIOS, 2010). A Figura 53 é um exemplo deste documento.

Figura 53 - Exemplo de Esboço de Nível de Log de Teste.
Fonte: IEEE 829, 2008.

5.2.1.7. RELATÓRIO DE ANOMALIAS (AR)

Seu propósito é documentar qualquer evento que ocorra durante o processo de testes e que requeira algum tipo de investigação. O evento a que a norma se refere pode ser um problema, defeito, incidente de teste, anomalia, relato de erro ou um ponto controverso. Quando os registros destes defeitos são feitos por ferramentas automatizadas, o relatório pode sair em outro padrão que não o IEEE 829-2008 (IEEE 829, 2008; RIOS, 2010). A Figura 54 mostra um exemplo do relatório.

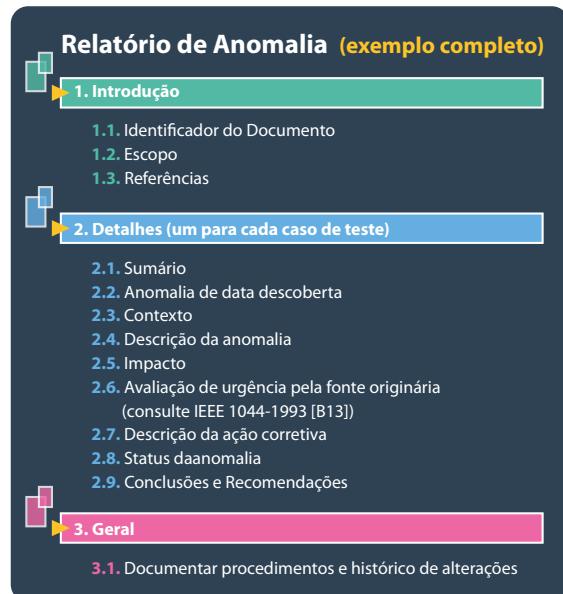


Figura 54 - Exemplo de Esboço de Relatório de Anomalias.

Fonte: IEEE 829, 2008.

5.2.1.8. RELATÓRIO DE STATUS DE TESTES PROVISÓRIOS (LITSR)

O objetivo deste relatório é resumir os resultados das atividades de teste designadas. Também pode fornecer avaliações e recomendações

com base nesses resultados. Geralmente substitui-se a palavra "Nível" no título do documento pelo nome da organização e indica-se o nível de teste específico, por exemplo, Relatório de status provisório de aceitação do teste. Por haver um formato definido do relatório para cada nível de teste identificado pela organização, pode haver uma grande variação de detalhe (IEEE 829, 2008). A Figura 55 mostra um exemplo do relatório.



Figura 55 - Exemplo de Esboço de Relatório de Status de Testes Provisório.

Fonte: IEEE 8290, 2008.

5.2.1.9. RELATÓRIO DE NÍVEL DE TESTE (LTR)

A finalidade do relatório de nível de teste é resumir os resultados das atividades de teste designadas e fornecer avaliações e recomendações baseadas neles. Ele deve ser feito por nível de teste, por exemplo, teste de integração, teste de sistema, teste de aceitação (IEEE 829, 2008; RIOS, 2010). É dividido em três partes, como mostra a Figura 56.



Figura 56 - Exemplo de Relatório de Nível de Teste.

Fonte: IEEE 829, 2008.

5.2.1.10. RELATÓRIO MÁSTER DE TESTE (MTR)

O relatório máster de teste tem como objetivo resumir os resultados dos níveis de atividades de teste e fornecer avaliações com base nesses resultados. Sempre que o plano de teste máster for gerado ou implementado, é necessário que os resultados descritos no relatório correspondam ao plano de teste máster. O relatório de teste máster é dividido em três partes, como mostra a Figura 57, e para a primeira ser completa, deve seguir estrutura tal como no relatório de testes de status provisório.

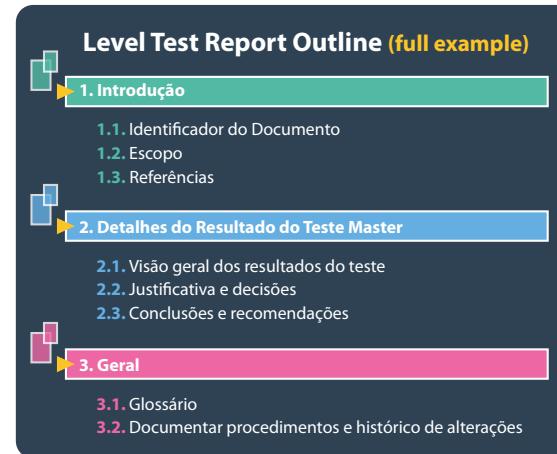


Figura 57 - Exemplo de Relatório Máster de Teste.

Fonte: IEEE 829, 2008.

6 EXECUÇÃO DOS TESTES DE SOFTWARE

Durante esta fase, a equipe de testes realizará a execução com base nos planos de teste e nos casos de teste preparados, observando as normas vigentes. Os erros serão relatados de volta à equipe de desenvolvimento para correção e depois serão novamente testados.

As atividades desta fase são: executar os casos de testes conforme planejado e especificado, documentar os resultados dos testes, documentar os erros encontrados, testar novamente os componentes

corrigidos. Os produtos são os resultados dos testes e os relatórios de testes, incluindo o de defeitos. A Figura 58 deixa mais claro como ocorre este processo.



Figura 58 - Processo de testes de software.

Fonte: SOMMERVILLE, 2018.

FIQUE ALERTA!



O processo de testes é caro e demorado. É muito importante escolher casos de testes eficientes e eficazes para os testes. Entenda mais a fundo o porquê em Sommerville (2018).

Os resultados dos testes devem ser analisados a cada etapa executada do processo de teste. Todos os registros da execução dos testes devem ser guardados em uma ferramenta específica para a gestão dos testes. Assim, o gestor do teste poderá saber o que ainda precisa ser corrigido pela equipe desenvolvimento, o que está em processo de teste e o que já foi dado como concluído, ou seja, já foi testado e aprovado (CARVALHO, 2011).

Um fluxo mais detalhado do processo de execução de testes é mostrado na Figura 59.

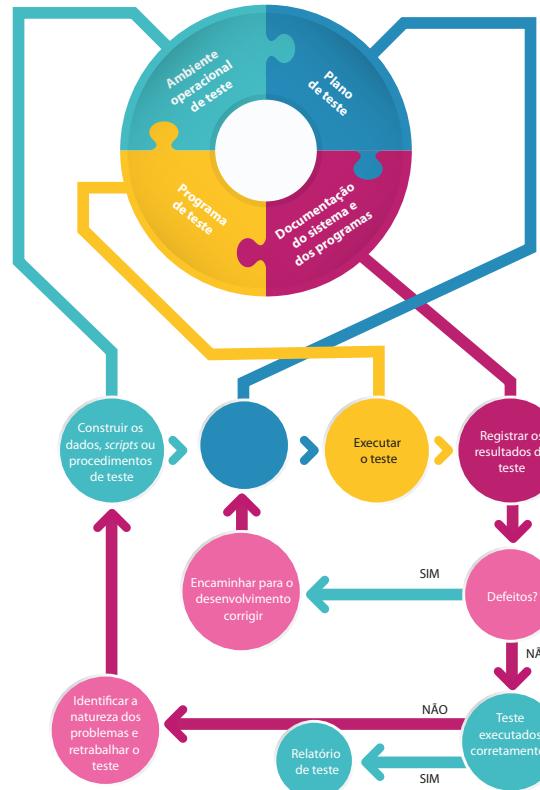


Figura 59 - Fluxo de execução dos testes.

Fonte: CARVALHO, 2011.

Há diversos métodos para testar um software, como pontua Carvalho (2011):

- **Teste de autorização** (funcionalidade): tem a finalidade de garantir o cumprimento das regras de autorização. As pessoas podem executar somente funções para as quais são autorizadas.
- **Teste de integridade dos arquivos** (funcionalidade): tem como objetivo garantir que os arquivos foram atualizados após um módulo do software ser executado.
- **Teste de recuperação** (continuidade): garante que todos os procedimentos de reinicialização do sistema após uma falha sejam testados.
- **Teste de estresse** (performance): visa testar se o software funciona corretamente sob estresse, ou seja, com uma grande carga de processamento.
- **Teste manual** (usabilidade): tem o intuito de testar manualmente o software, em um ambiente de teste mais próximo possível do de produção, para verificar se é amigável, fácil de usar.
- **Teste de segurança** (segurança): sua finalidade é verificar a possibilidade de falhas de segurança, que podem levar pessoas não autorizadas a acessarem informações indevidas. Pode haver a necessidade de um especialista em segurança para a realização deste teste.
- **Teste de inspeções** (manutenibilidade): seu objetivo é testar a facilidade de efetuar a manutenção do software. Para os desenvolvedores, que conhecem o sistema, é fácil realizar alterações. Se outras pessoas, porém, tiverem que fazer a manutenção, precisam entender facilmente a lógica.
- **Teste de conexão** (conectividade): visa garantir que os sistemas conversam uns com os outros.
- **Teste de performance** (performance): sua finalidade é testar se os requisitos de desempenho definidos, tais como transações por hora e número de acessos simultâneos, foram atendidos.
- **Teste operacional** (operacionalidade): visa avaliar toda a documentação de operação do software. Os encarregados de executá-lo são as equipes de produção e de operação do sistema.

A tabela 7 é um exemplo de checklist que pode ser utilizado na execução dos testes. Como você pode notar, há vários arquivos batch(.bat) nos pré-requisitos dos cenários, cujo objetivo é fornecer a entrada de dados para a simulação da situação real no processo de teste. Este, como alguns outros documentos são padronizados pelas normas que vimos no capítulo anterior cujo objetivo é estabelecer o formato dos artefatos que serão gerados durante o processo, para aumentar a qualidade do software.

CURIOSIDADE



Os arquivos batch (.bat) são arquivos de sistema, que executam comandos em lote que “falam” diretamente com o sistema operacional. Ele foi concebido para ser usado com o MS-DOS em linha de comando. Procure mais sobre ele na Internet e veja quanta coisa bacana se pode fazer com ele.

Processo: Importa Resultado da Análise de Crédito	
Cenário #4	
Executar a importação de pedidos com dois dias em pendência de análise de crédito	
Pré-requisitos	<ul style="list-style-type: none"> ► Garantir que existam pedidos aguardando análise de crédito há dois dias. ► Garantir que exista arquivo de simulação Simula-02.TXT.
Ações	<ul style="list-style-type: none"> ► Executar importação da análise de crédito.
Conferências	<ul style="list-style-type: none"> ► Mensagem <i>Alguns pedidos estão em análise há dois dias ou mais!</i> ► Confirmar se o arquivo foi excluído. ► Confirmar exibição da lista de pedidos com dois dias em análise ao usuário. ► Confirmar e-mail para o Gerente de Vendas, de Crédito e Cobrança. ► Confirmar se o e-mail possui lista de todos os pedidos com dois dias em análise.
Cenário #5	
Executar a importação da análise de crédito com problemas em alguns itens	
Pré-requisitos	<ul style="list-style-type: none"> ► Garantir que existam pedidos que aguardam análise de crédito. ► Garantir que exista arquivo de simulação Simula-03.TXT.
Ações	<ul style="list-style-type: none"> ► Executar importação da análise de crédito.
Conferências	<ul style="list-style-type: none"> ► Mensagem <i>Alguns erros ocorreram durante o processamento!</i> ► Confirmar se o arquivo foi excluído. ► Confirmar a existência do log de erros. ► Verificar se o log de erros foi apresentado ao final do processo. ► Confirmar se todos os itens “ruins” estão no log de erros. ► Confirmar se itens “bons” foram corretamente atualizados.

Tabela 7 - Procedimentos de testes individualizados por cenário.

Fonte: BARTIÉ, 2002.

Pezzé e Young (2008) afirmam que a execução dos testes deve ser bastante automatizada para que as atividades, que são repetidas inúmeras vezes, sejam executadas com o mínimo de intervenção humana

possível, o que pode melhorar a eficiência de atividades de qualidade. Vamos aprender um pouco mais sobre automatização de testes.

6.1. AUTOMAÇÃO DOS TESTES

Durante muito tempo os testes de software foram feitos com simulações de situações, criadas pelos próprios programadores e analistas, que usavam dados fictícios ou uma amostra de dados reais, que eram gerados manualmente ou utilizavam programas pontuais para este fim. Quase não havia ferramentas disponíveis para ajudar nestas atividades. E os softwares tornam-se cada dia mais complexos de serem testados, devido à infraestrutura de tecnologia, que evolui rapidamente, trazendo novos ambientes e ferramentas. Há testes muito complexos, com alto grau de dificuldade de execução sem o auxílio de uma ferramenta de automatização, como exemplo os testes funcionais e de regressão (RIOS; MOREIRA, 2006).

A automação dos testes evita que erros passem despercebidos aos olhos da equipe de testes. Como muitos testes chegam a ser repetidos centenas de vezes, tornam-se cansativos, caso feitos manualmente, e a probabilidade de deixar erros passarem aumenta a cada execução. Seres

humanos são muito lentos e sujeitos a erros com tarefas repetitivas. Por outro lado, são excelentes para identificar cenários relevantes de execução correspondentes aos casos d teste (PEZZÈ; YOUNG, 2008). Podemos dizer que automatizar testes assegura uma maior qualidade do software (BRAGA, 2016). Ademais, automatizar os testes pode reduzir significativamente os custos de teste (SOMMERVILLE, 2007). Por outro lado, a automação dos testes tem um custo alto e é de longo prazo, pois requerem desenvolvimento de ferramentas específicas que substituam testes manuais complexos. (RIOS; MOREIRA, 2006).

FIQUE ALERTA!



É imprescindível que a organização tenha um processo formal de testes bastante sólido para poder implantar a automação.

6.1.1. Ferramentas automatizadas de testes

Sommerville (2007) conceitua o *workbench* de testes como um conjunto de ferramentas integradas, exemplificado na Figura 60, que apoiam

o processo de testes. É usado em conjunto com frameworks de teste, cuja finalidade é apoiar a execução automatizada dos testes. O *workbench* gera dados de teste incluindo ferramentas que simulam outras partes do sistema.

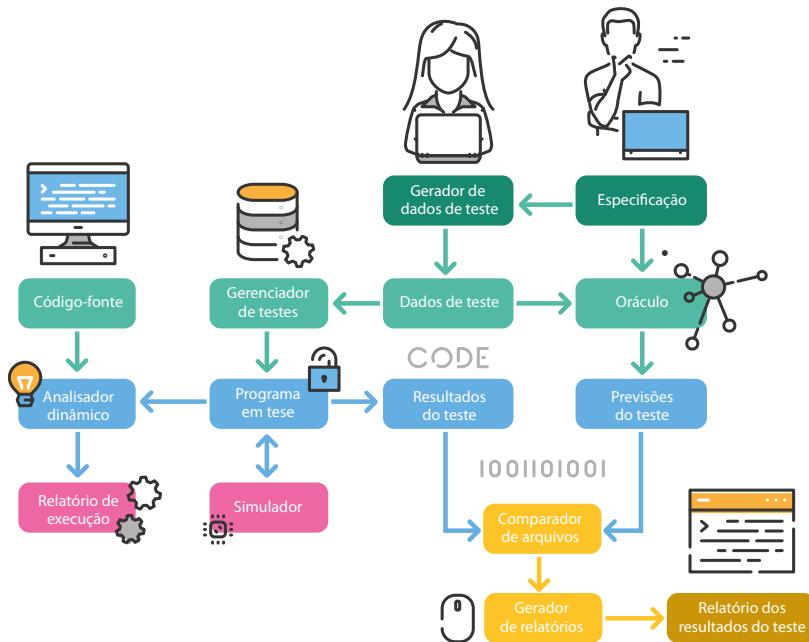


Figura 60 - Algumas ferramentas que podem compor um *workbench* de testes.
Fonte: SOMMERVILLE, 2007.

O **gerenciador de testes** gerencia a execução de testes de programa, acompanha os dados de teste, resultados esperados e recursos do programa testados. Um exemplo de gerenciador é o JUnit, um framework de automação.

O **gerador de dados de teste** gera os dados para o programa que será testado, podendo ser realizado por uma seleção de dados em um banco de dados ou padrões de geração de dados aleatórios.

O **oráculo** gera previsões dos resultados esperados no teste. Pode ser uma versão anterior do programa ou um protótipo dele.

O **comparador de arquivos** compara os resultados dos testes atuais com os anteriores, ressaltando as diferenças encontradas. São muito utilizados em teste de regressão, comparando resultados de diferentes versões do sistema.

O **gerador de relatórios** fornece recursos para a definição e a geração de relatórios para os resultados de teste.

O **analisador dinâmico** insere no programa código que conta o número de vezes que cada declaração foi executada e, ao final do teste, gera um perfil de execução com o resultado da contagem.

O **simulador** pode ser de diferentes tipos. O de **alvo** simula a máquina em que será executado o sistema, o de **interface** as inúmeras interações com o usuário, dirigidas por script e o uso do de **E/S** (entrada e saída) é para quando o timing de sequências de transações pode ser repetido.

Há uma quantidade significativa de tempo e esforço para a criação de um *workbench* abrangente. Somente sistemas de grande porte utilizam um tão completo quanto o da Figura 60. Nesse caso, os custos com teste podem chegar a 50% do custo total de desenvolvimento do sistema (SOMMERVILLE, 2007).

Pfleeger (2004) separa as ferramentas para análise de código em duas categorias: **análise estática**, realizada quando o software não está sendo executado e a **análise dinâmica**, quando ele está sendo executado. As ferramentas que fazem uma análise estática do programa-fonte para investigar se está correto podem ser agrupadas em quatro tipos:

O analisador de código, que avalia automaticamente a sintaxe dos componentes, ressaltando os erros.

O verificador de estrutura, que gera um grafo retratando o fluxo lógico e verifica os problemas estruturais.

O analisador de dados, que verifica as estruturas e declarações de dados, interfaces entre os componentes e registra os erros encontrados.

O verificador de sequência, que avalia as sequências de eventos, ressaltando os erros.

Já na análise dinâmica, as ferramentas automatizadas, por vezes chamadas de monitores de programa, acompanham e relatam o comportamento do sistema, permitindo que a equipe de testes obtenha os estados dos eventos durante sua execução. Elas podem, por exemplo, mostrar o número de vezes que um componente é chamado, que uma linha de código é executada, que um ponto de decisão é ramificado.

O autor ainda fala em ferramentas que automatizam o planejamento e execução dos testes, como:

Captura e repetição, que capturam as teclas digitadas, entradas e saídas e comparam com os resultados esperados, relatando os erros e possibilitando o rastreamento deles. São extremamente úteis no reteste.

Stubs e drivers. Os *stubs* são programas que simulam o comportamento de uma unidade de *hardware* ou *software*, eliminando as dependências entre eles, conforme mostra a Figura 61. Cram respostas

simuladas em uma comunicação entre o *hardware* e o *software* do sistema, possibilitando inúmeras e variadas situações de testes para identificação de erros. Também são excelentes para testar softwares com muitos sistemas legados, reduzindo a complexidade de testar as interfaces. Os drivers são programas desenvolvidos com a finalidade específica de testar uma unidade de *software*. Eles podem definir as variáveis de estado para os casos de testes e executá-los, simular a inserção de dados pelo teclado, comparar o resultado do teste com o esperado e relatar as diferenças, possibilitar o rastreamento dos caminhos percorridos enquanto o programa é executado, redefinir variáveis para um novo caso de teste interagir com a depuração. Podem tanto provocar cenários de testes quanto exercitar o código-fonte, o que possibilita a medição dos testes de caixa branca. A Figura 62 traz um exemplo dos drivers (BARTIÈ, 2002; PFLEEGER, 2004).

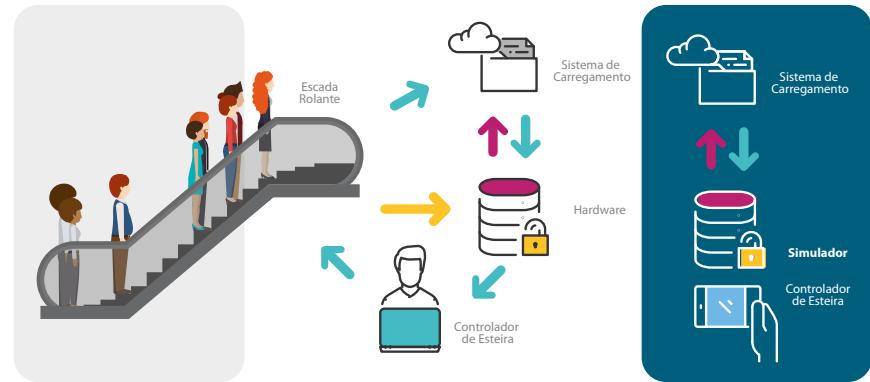


Figura 61 - Funcionamento dos stubs.

Fonte: BARTIÈ, 2002.

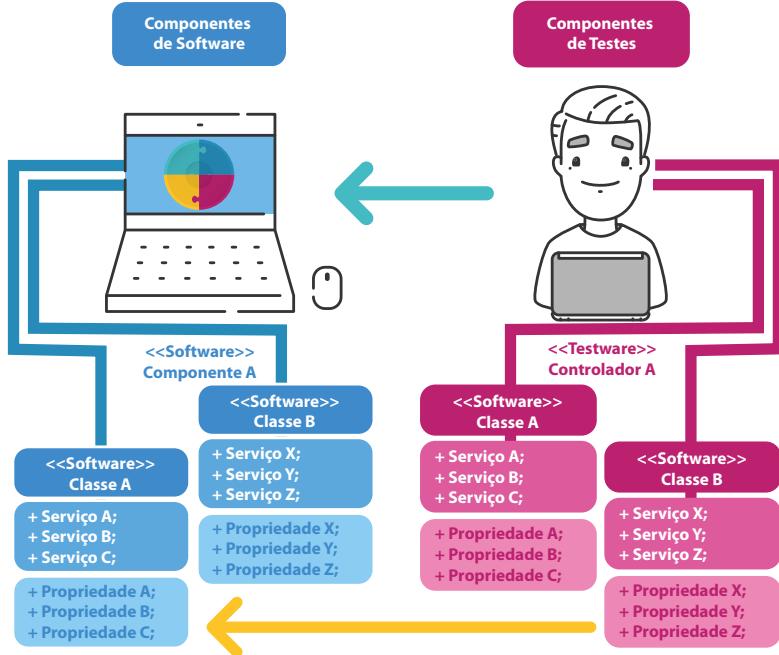


Figura 62 - Funcionamento dos drives ao testar cada unidade de software.

Fonte: BARTIÉ, 2002.

SAIBA MAIS



Testware é o nome que se dá ao conjunto de todo os artefatos gerados nas fases de verificação e validação. São as checklists, os documentos de teste – estratégias, planejamento, roteiros, casos de testes, especificações de testes, roteiros, scripts, relatórios, rotinas automatizadas de execução de testes (BARTIÉ, 2002; RIOS, MOREIRA, 2006).

Ambientes automatizados de teste podem integrar ferramentas para execução de testes a fim de formar um ambiente bastante amplo para a execução. Estas ferramentas automatizam os testes tanto quanto possível e estão conectadas a bancos de dados de testes, ferramentas de medição, de análise de código, de simulação, de modelagem, além de editores de texto.

Há muitas ferramentas no mercado para a automatização de testes. Na tabela vamos ver algumas que existem, relacionadas aos níveis de testes que elas cobrem.

	Ferramentas		Comentários
Testes Unitários	JUnit Visual Studio Unit testing Monitor XUnit Unit.js PHPUnit PyUnit	Java .Net Javascript PHP Python	
Testes de integração	Postman DB Unit As mesmas dos testes unitários		Integração com APIs Integração com banco de dados
Testes de Interface	Selenium Espresso XCode UI Tests Visual Studio Coded UI Test	Web Android iOS Windows	
Testes de Aceitação	Cucumber SpecFlow Jasmine PHP Spec Behave	Java .Net Javascript PHP Python	
Testes Não Funcionais	JMeter	Apache	Testes de carga e performance
Testes Automatizados	Jenkins TravisCI		Scaneiam o repositório e se algum arquivo foi modificado executam a bateria de testes

Tabela 8 - Ferramentas para Testes de Software e Respectivas Linguagens.

Fonte: ADAPTADO DE FERREIRA, 2018.

6.1.2. Ambiente de testes

Para executarmos os testes precisamos criar um ambiente que simule o ambiente de produção, onde o sistema rodará. O ambiente de testes deve simular diversos cenários de utilização para que tenhamos uma maior qualidade do produto desenvolvido (PFLEEGER, 2004).

O objetivo do ambiente de testes é fornecer a infraestrutura necessária de hardware e software para a realização dos testes, garantindo um ambiente exclusivo e ideal de simulação, isolado de qualquer interferência dos demais ambientes para que não haja comprometimento da qualidade dos testes e que as informações e processos sejam confiáveis. Sua infraestrutura deve emular o ambiente de produção para que ele seja o mais próximo possível do de produção e das condições reais de uso do sistema. Esse ambiente, mostrado nas Figura 63 e Figura 64, contempla equipamentos dedicados, ferramentas de testes já instaladas e sistemas parcialmente instalados com seus respectivos softwares de apoio. É recomendável que dados de produção que precisem ser usados para os testes sejam criptografados (BARTIÈ, 2002).



Figura 63 –Ambientes de Sistemas.
Fonte: BARTIÈ, 2002.

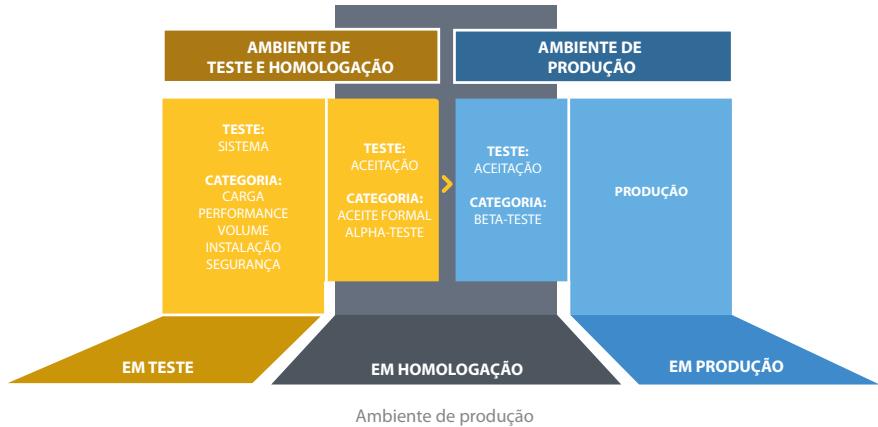


Figura 64 - Ambientes de Teste.
Fonte: BARTIÈ, 2002.

Devido à gestão isolada do ambiente de testes, a equipe tem autonomia para decidir quando e como os testes de determinada aplicação serão executados. Assim, poderemos submeter o software a rigorosos procedimentos de testes sem que haja risco de manipulação ou interferência de atividades externas, caso ocorram eventuais modificações emergenciais de uma aplicação. Logo, mesmo que testes específicos sejam executados no ambiente de desenvolvimento, outras categorias de testes que tenham objetivos diferentes, deverão ser

processadas no ambiente de testes, com total isolamento entre eles.

Os testes são realizados com as técnicas de caixa preta, onde não se conhece o funcionamento interno do sistema. A finalidade é provocar variações específicas de regras de negócio e avaliar o resultado, o que dirá se os testes foram bem-sucedidos ou não (BARTIÈ, 2002).

SAIBA MAIS



Teste alfa e beta fazem parte da homologação, aceite da solução, do sistema pelo cliente. Os testes alfa são executados pelos clientes, dentro de um ambiente simulado. Já o teste beta utiliza a infraestrutura em que o software rodará em definitivo. Vocês já devem ter ouvido falar em beta testers. Busquem mais informações sobre o assunto. Nas referências há vários autores renomados que falam sobre o assunto.

6.1.3. Quando parar os testes

Como saberemos quando devemos parar com os testes? Pressman e Maxim (2015) dizem que não há uma resposta definitiva para esta

questão, mas algumas tentativas empíricas. Coletar métricas durante o teste de software e usando modelos estatísticos é a maneira que parece mais plausível para responder a essa questão.

CURIOSIDADES



Tom de Marco disse: "se você não consegue medir só existe uma razão para acreditar que você ainda tem o projeto sob o seu controle: otimismo histérico."

Sommerville (2007) cita a previsão de confiabilidade como métrica para estimar a duração e também parar os testes, pois, devido ao fato de serem bastante dispendiosos, é preciso encerrá-los o mais breve possível. O teste pode ser interrompido quando atingir o nível de confiabilidade necessário. O sistema funciona com a variação da confiabilidade. À medida que as falhas descobertas no processo de testes do sistema e são

corrigidas, a confiabilidade aumenta. Calculamos a confiabilidade por modelos matemáticos. Entende-se como confiabilidade a probabilidade de um sistema operar sem defeitos durante um período de tempo e sob certas condições durante seu tempo de execução (PFLEEGER, 2004).

SAIBA MAIS



Há vários autores que abordam os modelos matemáticos para a previsão de confiabilidade, caso queira se aprofundar nos cálculos. São eles: Abdel-Ghaly et al (1986), Kan (2003), Littlewood (1990), Musa (1998).

Outra medição para projetos de testes de software é a análise de ponto de teste (APT). Sua finalidade também é estimar o tempo necessário para a atividade de testes. O tamanho do sistema é considerado para fazer a medição. Rios e Moreira (2006, p 84) elencam fatores que também afetam os testes de software:

Grau de complexidade do processo de teste;

Nível de qualidade que se pretende alcançar com os testes;

Grau de envolvimento dos usuários com os testes;

Interfaces que as funções que estão sendo testadas têm com os arquivos;

Qualidade do sistema que está sendo testado (ciclo de reincidência de defeitos);

Nível de cobertura esperado com os testes;

Experiência e produtividade da equipe de teste (medidos através de indicadores históricos);

Grau de automação dos testes;

Qualidade do ambiente de teste, inclusive sua capacidade de simular o ambiente de produção;

Qualidade da documentação do sistema e, especialmente dos requisitos.

A Figura 65 mostra como calcular os pontos de teste.

CONCLUSÃO

Finalizamos a unidade de testes de sistemas! Agora você entende melhor a importância de testar os softwares. Entende o que são testes de software e por que eles apresentam defeitos. Sabe que para ter um produto final de qualidade tem que padronizar a atividade de testes, utilizando normas pertinentes e que, usando uma metodologia, consegue fazer o planejamento dos testes. Consegue identificar técnicas, níveis e tipos de testes de software. Conhece as normas mais utilizadas no mercado. Sabe como um ambiente de testes deve ser para que eles sejam bem sucedidos. E também sabe como é a automação dos testes e quando eles devem ser interrompidos, tendo em vista que o processo é bastante dispendioso.

Agora é encarar o mercado de trabalho!

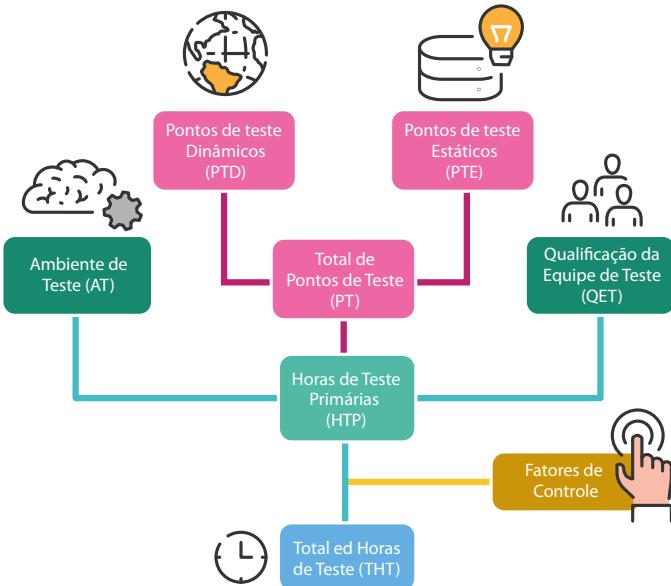


Figura 65 - Análise de Pontos de Teste.

Fonte: RIOS E MOREIRA, 2006.

Alguns autores apontam ainda mais algumas métricas, porém as apresentadas aqui são as mais utilizadas.

REFERÊNCIAS

ABDEL-GHALY, Abdalla A.; CHAN, P.Y; LITTLEWOOD, Bev. **Evaluation of Competing Software Reliability Prediction.** in IEEE Transactions on Software Engineering, v.12, n.9, p.950-967, set. 1986.

ALAQAIL, Hesham; AHMED, Shakeel. **Overview of Software Testing Standard ISO/IEC/IEEE 29119.** International Journal of Computer Science and Network Security, v.18, n.2, 2018.

ARAÚJO, Luciana de Barros. **A Importância do Teste de Software para a Qualidade do Projeto.** Disponível em: < <https://pmkb.com.br/wp-content/uploads/2013/11/a-importancia-do-teste-de-software.pdf>>. Acesso em 6 dez 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR ISO/IEC/IEEE 29119-1:2013.** Engenharia de software e sistemas - Teste de software - Parte 1: Conceitos e definições. Rio de Janeiro: ABNT, 2014.

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier, 2002.

BRAGA, Pedro Henrique C. **Teste de Software.** São Paulo: Pearson, 2016.

CARVALHO, Carlos Eduardo. **Teste de Software.** Florianópolis: SENAI/SC/DR, 2011.

CRESPO, Adalberto Nobato; SILVA, Odair Jacinto da; BORGES, Carlos Alberto; SALVIANO, Clênio Figueiredo; ARGOLLO JR., Miguel de Teive e; JINO, Mario. **Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo.** Disponível em: < <http://www.lbd-dcc.ufmg.br/colecoes/sbqs/2004/024.pdf>>. Acesso em 13 nov. 2019.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. **Introdução ao Teste de Software.** 2. ed. Rio de Janeiro: Elsevier, 2016.

DIAS NETO, Arilo Cláudio. Introdução a Testes de Software. **Engenharia de Software Magazine**, n.1, 14 jun 2010. Disponível em < https://edisciplinas.usp.br/pluginfile.php/3503764/mod_reFonte/content/3/Introducao_a_Teste_de_Software.pdf>. Acesso em 10 nov. 2019.

FERREIRA, Avelino. **Como garantir a qualidade do software: Testes automatizados**. Disponível em: < <https://www.knowledge21.com.br/blog/qualidade-do-software-testes-automatizados/>>. Acesso em 10 mar. de 2020.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **29119-2 ISO/IEC/IEEE International Standard**. Software and systems engineering - Software testing - Part 2: Test Processes. Piscataway: IEEE, 2013.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **29119-3 ISO/IEC/IEEE International Standard**. Software and systems engineering - Software testing - Part 3: Test Documentation. Piscataway: IEEE, 2013.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **29119-4 ISO/IEC/IEEE International Standard**. Software and systems engineering - Software testing - Part 4: Test Techniques. Piscataway: IEEE, 2015

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **29119-5 ISO/IEC/IEEE International Standard** - Software and systems engineering -- Software testing - Part 5: Keyword-Driven Testing. Piscataway: IEEE, 2016.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **829 IEEE Standard for Software and System Test Documentation** - Software and systems engineering. Piscataway: IEEE, 2016.

KAN, Stephen H. **Metrics and Modules in Software Quality Engineering**. Boston: Addison-Wesley, 2003.

LABOON, Bill. **A Friendly Introduction to Software Testing**. Scotts Valley: CreateSpace, 2016.

LITTLEWOOD, Bev. **Software Reliability Measurement**. IEE Colloquium on Software Metrics, p. 3/1-3/2. London, 1990.

MUSA, John D. **Software Reliability Engineered Testing**. New York: McGraw-Hill, 1998.

OLIVEIRA, Adriana Fernanda. **Um Estudo sobre a Aplicabilidade**

da Série de Padrões ISO/IEC/IEEE 29119 em Métodos Ágeis de Desenvolvimento de Software. 2017. 102f. Monografia – Universidade Federal de Ouro Preto, João Monlevade, 2017.

OLIVEIRA, Ana Paula Joslin de. Ferramenta Web para Criação de Plano de Testes Baseada na Norma IEEE 829-2008. 2017. 77f. TCC – Universidade Regional de Blumenau, Blumenau, 2011.

PEZZÈ, Mauro; YOUNG, Michael. Teste e Análise de Software: processos, princípios e técnicas. Porto Alegre: Bookman, 2008.

PFLEINGER, Shari Lawrence. Engenharia de Software: teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, Roger S.; MAXIM, Bruce R. Software Engineering: a practitioner's approach. 8. Ed. New York: McGraw-Hill Education, 2015.

RIOS, Emerson; MOREIRA, Trayahú. Teste de Software. 2. ed. Rio de Janeiro: Alta Books, 2006.

RIOS, Emerson. Documentação de Teste de Software: dissecando o padrão IEEE 829. 2. ed. Niterói: Imagem Art Studio, 2010.

RUNGTÀ, Krishna. **Software Testing: learn in 1 day.** Seattle: Kindle Edition, 2017.

SANTOS, Anderson Pereira; SILVA, Gabriela Oliveira Mota da. **TEGSTI: uma ferramenta de apoio à documentação de testes de software baseado[sic] na IEEE 829-2008.** Científico. V. 17, n. 36, jul./dez. 2017.

SILVA, Josenilson dos Santos et al. **O Processo de Teste de Software.** Tecnologias em Projeção. v 7, n. 2, 2016 p.99.

SOMMERVILLE, Ian. **Engenharia de Software.** 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

SOMMERVILLE, Ian. **Engenharia de Software.** 10. ed. São Paulo: Pearson Prentice Hall, 2018.

MINICURRÍCULO

Iris Jerusa D'Amico Burger

Iris Jerusa D'Amico Burger é mestre em Administração Estratégica pela PUC-PR (2011), especialista em Software Livre pela UFPR (2005), especialista em Telecomunicações pela UFPR (2001), especialista em Redes e Sistemas Distribuídos pela PUC-PR (1998) e bacharel em Análise de Sistemas pela PUC-PR (1997).

Atua na área de infraestrutura de TI, sistemas e inovação há mais de vinte anos, com atuação na Argentina e nos Estados Unidos. Foi gerente de projetos em fábrica de software. É atuante no ecossistema de inovação e community manager do grupo Women in Blockchain. Atualmente tutora o curso semipresencial bilíngue do SENAI PR, mentora a Trilha de Inovação do Sistema FIEP e inúmeros Hackatons.

CRÉDITOS

SENAI – DEPARTAMENTO REGIONAL DO PARANÁ

José Antônio Fares

Diretor Regional

Giovana Chimentao Punhagui

Gerente Executiva de Educação

Vanessa Sorda Frason

Gerente de Educação Profissional

Sandra Cristina Brasil Toloto

Coordenadora Pedagógica

Iris Jerusa D'Amico Burger

Elaboração

Cesar Ricardo Stati

Revisão Técnica

Alexandre Luis Kloch

Coordenação Técnica

Erica Luz de Souza

Orientação Pedagógica

Karem Morigi

Revisão Ortográfica e Gramatical

Daniel Gustavo Hella

Estela Pereira

Coordenação de Projeto

Willian Bill

Ilustrações e Tratamento de Imagens

Anderson Calixto de Carvalho

Diagramação, Revisão de Arte e Fechamento de Arquivo

Ricardo Luiz Freire de Menezes

Projeto Gráfico

