

Engenharia de *Software*



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico



Teste e manutenção de *software*

Responsável pelo Conteúdo:

Prof.^a Dr.^a Ana Paula do Carmo Marchetti Ferraz

Revisão Textual:

Prof.^a Me. Luciene Oliveira da Costa Santos



- Introdução
- Estratégia de testes
- Inspeção de *software*
- Técnicas ou métodos de testes



Nesta unidade, serão apresentadas as estratégias de testes e tipos de manutenção que existem.

Testar um software é fundamental para garantir a qualidade do produto de software comercializado, por isso, em grandes projetos, existem equipes independentes que realizam esta atividade.

Novamente, gostaríamos de lembrar que este Material teórico não esgotará todos os aspectos relacionados a esses conceitos. Portanto, sugerimos sempre pesquisas adicionais.

Contextualização

Discutirmos as questões relativas a teste e manutenção não é tarefa fácil. Todos nós sabemos da importância dessas atividades, mas muito pouco tempo dedicamos a elas.

Hoje, entendemos que as atividades relacionadas a testes e manutenção são fundamentais no processo de desenvolvimento de produto de software.

A seguir, apresentamos duas charges relacionadas a essa atividade. Uma sobre a evolução que essa atividade teve nos últimos tempos e a outra acerca da postura (divertida e, em algumas situações, real) que alguns gerentes têm em relação à equipe de teste.



Testar um produto de software não é tarefa fácil, mas é necessária. Esperamos que ao final desta unidade você também tenha consciência disso.

Introdução



Não temos como falar em desenvolvimento de um produto de *software* sem mencionar fatores relacionados à qualidade.

Qualidade pode ser um conceito subjetivo – o que é qualidade para um, pode não ser a outro. Entretanto, para medirmos a qualidade de *software*, conceitos subjetivos não podem ser considerados. É necessário que tenhamos uma definição precisa do que é um *software* de qualidade, segundo princípios e propriedades, para podermos atestar seu nível.

Entretanto, para garantirmos um bom produto de *software*, é necessário que diferentes testes sejam executados, e é nessa direção que trabalharemos neste capítulo.

Mesmo depois de todo o cuidado durante o desenvolvimento, com revisões periódicas e inspeções para remoção de defeitos, necessitamos realizar a etapa de teste para detectar os defeitos que escaparam das revisões e para avaliar o grau de qualidade de um produto e seus componentes.

O fluxo de teste é extremamente importante para o desenvolvimento de *software*. Segundo Pressman (1996) e Sommerville (2011), o custo de correção de um problema na fase de manutenção pode chegar até a 60% do custo total do projeto. Em outras palavras, realizar testes é fundamental não apenas para atestar qualidade do produto, mas também para diminuição de custos totais do projeto.

Na fase de teste, precisamos detectar a maior quantidade possível de defeitos que não foram apanhados pelas revisões, considerando os limites de custos e prazos (PAULA FILHO, 2005).

Estratégia de testes



Segundo Pressman (2007) e Paula Filho (2005), teste é considerado um conjunto de atividades que deve ser planejado antecipadamente e realizado de forma sistemática.

O planejamento e a realização das atividades de teste planejadas fazem parte do que definimos por **estratégia de teste de software**.

Como sempre falamos, a Engenharia de Software coloca ordem em diversas atividades do desenvolvimento de um produto de software. Esse “colocar ordem” implica planejar antecipadamente atividades e gerenciá-las. No caso de teste, isso não é diferente. Temos que montar uma estratégia de teste de software antes de iniciarmos qualquer atividade relacionada a ela.

Uma estratégia, ou política de teste de software, define técnicas de projeto de casos de teste e métodos de teste específico para cada etapa do processo de Engenharia de Software (PRESSMAN, 1996).

Nenhum produto é testado apenas ao final do seu desenvolvimento, por isso que, para cada etapa do processo de Engenharia de Software, é necessária uma estratégia específica de teste.

Por empirismo, sabemos que:

- A atividade inicia no nível de módulos e prossegue “para fora”, na direção da integração de todo o sistema. Sempre que você fez um programa, durante sua vida acadêmica, que continha módulos ou procedimentos, normalmente você os testava separadamente e depois testava sua conexão. É a isso que este item se refere.
- Diferentes técnicas de teste são apropriadas a diferentes pontos de tempo. Em cada fase do projeto, testes devem ser aplicados e existem técnicas específicas ou que podem trazer melhores resultados em pontos específicos do projeto.

Lembre-se sempre de que “a qualidade das partes garante a qualidade do todo”.

Considerando uma estratégia de teste, ela deve:

- Acomodar testes de **baixo nível** e teste de **alto nível**.
- Oferecer orientação ao profissional.
- Oferecer um conjunto de marcos de referência ao administrador do projeto e de cada fase do sistema.
- Ser mensurável. Existem testes estatísticos que são utilizados para testar desempenho e confiabilidade do programa para checar como ele trabalha sob determinadas condições operacionais.

Vale ressaltar que a atividade de teste num produto de software não é a mesma realizada pela equipe de desenvolvimento.

A equipe de desenvolvimento realiza atividades de revisão e depuração e, somente após atestar que o projeto está adequado, é que a equipe de teste entra em ação.

Toda empresa de desenvolvimento de produto de software de grande porte possui equipes de testes contratadas para os projetos desenvolvidos. Ela é formada por um **grupo independente** de pessoas para atestar a qualidade da avaliação e verificação do que está sendo realizado.

Geralmente, a ação chega a um nível em que desenvolvedores e analistas não encontram mais erros, mas isso não significa que eles não existam. É nesse momento que novos agentes são chamados (equipe de teste que não tenha participado do processo de desenho e implementação do projeto).

Segundo Sommerville (2007), um Grupo Independente de Teste (*Independent Test Group – ITG*) apesar de ser, como o próprio nome já diz, independente, ele faz parte da equipe de projeto de desenvolvimento de software, no sentido que está envolvido durante o processo de especificação e continua envolvido planejando e especificando estratégias e políticas de teste ao longo do projeto.

Devemos mencionar que, embora as atividades de teste e depuração sejam diferentes e independentes, a depuração deve ser sempre acomodada em qualquer estratégia de teste.

Antes de começar a explorar as estratégias de teste, dois conceitos devem ser bem compreendidos: Validação e Verificação.

“Validação e Verificação (V&V) é o nome dado aos processos de verificação e análise que asseguram que o *software* cumpra com as especificações e atenda às necessidades dos clientes”. (SOMMERVILLE, 2005, p. 358)

VALIDAÇÃO: refere-se ao conjunto de atividades que garante que o *software* que foi construído seja rastreável às exigências do cliente.

Basicamente responde à pergunta: estamos fazendo o produto **certo**?

VERIFICAÇÃO: refere-se ao conjunto de atividades que garante que o *software* seja construído da forma correta.

Basicamente responde à pergunta: estamos fazendo o **certo** produto?

Ambas as definições estão diretamente relacionadas à garantia de qualidade de *software*, como podemos observar na Figura 1:



Figura 1 - Relação das atividades relacionadas à garantia de qualidade do processo de *software*.

As principais definições referenciam sobre teste e são conduzidas pelo padrão IEEE (*Software Engineering Standards*). Utilizamo-nos desse padrão, referenciados pelos autores Sommerville (2005, 2007) e Paula Filho (2005), para discutirmos esses assuntos nesta Unidade. Entretanto, não deixe de acessar o site na nossa bibliografia, para conhecer um pouco mais sobre padrões IEEE relacionados aos assuntos de Engenharia de *Software*.



Explore

Faça uma pesquisa sobre IEEE e suas normas.

O processo de V&V ocorre em várias fases do projeto, começa com as revisões de requisitos e continua ao longo de todo o processo de desenvolvimento (codificação) até a fase final de teste.

Muitas pessoas confundem verificação de *software* com validação, mas, como podemos observar, são duas atividades distintas, que devem ser trabalhadas de forma complementar e nunca exclusiva.

O objetivo principal do processo de V&V, segundo Sommerville (2007), é estabelecer um vínculo de confiança entre o sistema de desenvolvimento e o projeto aprovado pelo usuário.

O nível de confiabilidade exigida depende, em grande escala, do propósito do cliente, das expectativas do usuário (que pode ou não ser a mesma pessoa que o cliente) e o atual ambiente/segmento do mercado no qual o *software* será utilizado.

Considerando o processo de V&V, há duas abordagens, conforme Sommerville (2007, p. 342):

1 - Inspeção de software ou revisão por pares: analisa e verifica representações de sistemas como documentos de requisitos, diagramas de projeto e código-fonte de programa. Você pode usar inspeções em todos os estágios do processo. Inspeções podem ser suplementadas por alguma análise automática de texto-fonte de um sistema ou documentos associados. Inspeções de software e análise automatizada são técnicas de V&V estáticas, usadas quando você não precisa executar o *software* em um computador.

2 - Teste de software: envolve executar uma implementação do *software* com dados de teste. Você examina as saídas do *software* e seu computador operacional para verificar se seu desempenho está conforme o necessário. O teste é uma técnica dinâmica de verificação e validação.

Os processos de inspeção e testes desempenham um papel complementar no desenvolvimento do *software* e há técnicas que podem ser utilizadas em momentos específicos da produção. Uma das vantagens de se desenvolver um produto de *software* por meio da técnica incremental (revisar conceitos de Engenharia de Software) é que, ao finalizarmos uma parte do *software*, ele pode ser testado em sua totalidade, naquele incremento (parte), pois testar um sistema só é possível quando há uma versão executável, ou seja, ele não precisa estar completo, mas deve haver partes executáveis e passíveis de teste.

Considerando o abordado no item 1 (inspeção de programas), embora seja amplamente utilizado nos dias atuais, o teste de programa é a principal técnica de verificação e validação.

Testar um *software* significa experimentá-lo, usando dados reais do usuário e examinar a saída esperada para confirmar anomalias.

Segundo Sommerville (2007), há dois tipos de testes que podem ser utilizados:

1 - Teste de validação: que tem a finalidade de mostrar se o *software* é o que o cliente deseja – se atende aos requisitos. Como parte deste processo, podem ser utilizados testes estatísticos para verificar confiabilidade e desempenho.

2 - Teste de defeito: que é destinado a revelar defeitos no sistema. O objetivo é procurar inconsistência entre o programa testado e suas especificações iniciais.

Considerando as definições, podemos perceber que não há uma definição clara entre um e outro. À medida que descobrimos defeitos pelos testes de validação, normalmente, acontecem adequações para sanar inconsistências por meio da análise dos requisitos.

Sendo assim, toda vez que falarmos sobre testes, estaremos falando de verificação e validação de sistemas.

Uma confusão normal acontece em relação ao *debugging* e como este processo interage com as definições de testes.

Primeiro, precisamos entender que o processo de *debugging* é aquele que localiza e corrige os defeitos observados pelo processo de teste (V&V).

Para realizar o processo de *debugging*, podemos utilizar a técnica de verificação de variáveis, estruturas de controle, de repetição etc.

Localizar um defeito em um programa não é tarefa fácil, uma vez que geralmente ele se encontra longe do processo de saída do resultado. Mesmo assim, é necessário realizar esta etapa (*debugging*) com total responsabilidade para que a próxima etapa de V&V possa ser executada de forma segura. Na Figura 2, mostramos o processo de *debugging* de forma simplificada.

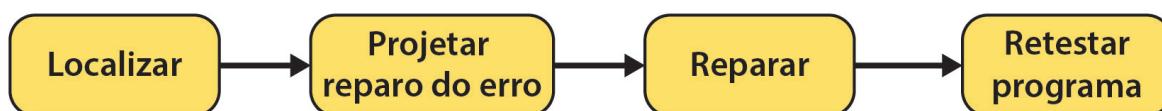


Figura 2 - Processo ilustrativo de debugging (SOMMERVILLE, 2007, p. 52)

Segundo Sommerville (2007), como no desenvolvimento de software, planos de testes devem ser projetados e seguidos para minimizar falhas após entrega do produto ao cliente.

Inspeção de software



Inspeção de software e teste são processos distintos.

Inspeção é processo de V&V estático, no qual um sistema de *software* é revisto para podermos encontrar erros, omissões e anomalias. Geralmente falamos de inspeção relacionada ao código-fonte, mas qualquer representação legível do *software* (requisitos, modelagem, planos e projetos) pode ser inspecionada.

Segundo Sommerville (2007), há três vantagens sobre inspeção de teste:

- Durante os testes, erros podem mascarar (ocultar) outros erros. Uma vez que um erro é descoberto, você nunca pode estar seguro se outras anomalias de saída são devidas a um novo erro ou se são efeitos colaterais do erro já descoberto. Uma única seção de inspeção pode descobrir vários erros.

- b) Versões incompletas de sistemas podem ser inspecionadas sem custos adicionais.
- c) Assim como procurar defeitos de programa, uma inspeção pode também considerar atributos de qualidade mais amplos como conformidade com padrões, portabilidade e facilidade de manutenção. Nesse momento, podem ser localizados, inclusive, algoritmos inapropriados e estilo de programação “pobre”, que dificultaria o processo de manutenção.

A inspeção de software com o objetivo de detecção de defeitos de programação foi formalmente desenvolvida pela IBM na década de 1970.

O processo é realizado por uma equipe independente da de desenvolvedores do produto a ser verificado e com diferentes experiências, que faz revisões linha a linha do código-fonte.

Segundo Sommerville (2007), a diferença fundamental entre inspeção de um programa e outros tipos de revisão de qualidade é que o objetivo especificado na inspeção é encontrar os defeitos de programa. Os defeitos podem ser erros de lógica, anomalias no código que possam indicar situações de não conformidade do software aos padrões do projeto idealizado. Outros tipos de testes, que ainda conheceremos, estão mais relacionados ao cronograma, custos, progresso, marcos de projeto, atendimento de requisitos de clientes etc.

Geralmente a equipe é formada por, pelo menos, quatro pessoas que analisam sistematicamente o código à procura de erros ou defeitos. Esse processo deve ser curto e deve focar detecção de defeitos, conformidade aos padrões. A equipe **não** deve sugerir como resolver inconformidades encontradas, afinal, não é a sua missão.

Todo processo de inspeção deve ser criteriosamente documentado.

Técnicas ou métodos de testes



Para serem eficazes os testes devem ser cuidadosamente planejados e estruturados (estratégia/política de teste).

Durante a realização dos testes, seus resultados devem ser cuidadosamente relatados (documentados) e inspecionados, pois nem sempre os resultados previstos são os obtidos e nem sempre é tão óbvio quando se detecta um erro. Lembre-se de que os erros óbvios foram diagnosticados e adequados durante o processo de depuração feito pelos programadores.

Segundo Paula Filho (2005, p. 184):

[...] testes são indicadores de qualidade do produto, mais do que meios de detecção e correção de erros. Quanto maior o número de defeitos detectados em um software, provavelmente maior também o número de defeitos não detectados. A ocorrência de um número anormal de defeitos em uma bateria de teste indica uma provável necessidade de redesenho dos itens testados.

Devemos lembrar sempre que, em se tratando de testes:

- Se erros graves forem encontrados com regularidade, então a qualidade e a confiabilidade do software são suspeitas.
- Se erros facilmente corrigíveis forem encontrados, então a qualidade e a confiabilidade do software estão aceitáveis, ou os testes são inadequados para revelar erros graves desse software em particular.
- Se não forem encontrados erros, então a configuração de testes não foi suficientemente elaborada e os erros estão escondidos no software.

Baseando-nos nesses parágrafos, podemos então dizer que:

- A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro.
- Se a atividade de teste for conduzida com sucesso, ela descobrirá erros no software.
- Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto.
- Um teste bem-sucedido é aquele que revela um erro ainda não descoberto.

Na Figura 3, podemos observar a estrutura das atividades de teste conforme apresentada.

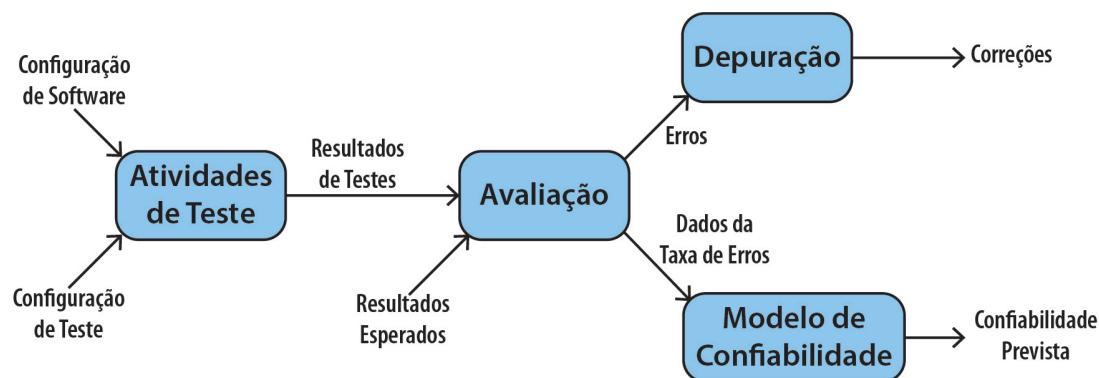


Figura 3 - Estrutura das atividades de teste.

As políticas de configuração de teste agregadas à configuração e documento do software gerarão uma política de atividades de teste que, após executadas, serão verificados os resultados obtidos com os dados esperados e realizada uma avaliação sobre esses dados. A partir daí, a taxa de erros entre o esperado e o obtido será analisada por meio de uma política de confiabilidade (também criada anteriormente no momento da definição da estratégia de teste). Caso a taxa de erro encontrada não seja adequada para o modelo de confiabilidade descrito, os erros serão novamente analisados, depurados e corrigidos.

Estratégias de testes



A partir de agora, veremos modelos de testes de sistemas. Portanto, vale relembrar que **validação** é importante para assegurar que o produto final corresponda aos requisitos do usuário; **verificação** assegura consistência, completitude e corretitude do produto em cada fase e entre fases consecutivas do ciclo de vida do *software*; teste examina o comportamento do produto por meio de sua execução.

Em todo o processo, temos que ter em mente isto:

- **Defeito**, num *software*, é uma deficiência mecânica ou algorítmica que, se ativada, pode levar a uma falha.
- **Falha**, num *software*, é um evento notável em que o sistema viola suas especificações. Geralmente aparece durante a utilização do produto pelos usuários.
- **Erro**, num *software*, é um item de informação ou estado de execução inconsistente. Deve ser corrigido durante a fase de manutenção.

Toda estratégia de teste é válida, desde que considere a verificação de testes de baixo nível (lógica) e testes de alto nível (usabilidade). Deve oferecer a orientação ao profissional, um conjunto de marcos de referência ao administrador, além de ter seu resultado mensurável.

Tipos de estratégias de testes

A realização, com sucesso, da etapa de teste de um *software* deve ter, como ponto de partida, uma atividade de construção de um projeto (dos casos) de teste deste *software*.

Para isso é necessário que:

- Seja conhecida a função a ser desempenhada pelo produto. Os testes devem ser executados para demonstrar que cada função é completamente operacional e sem erro. Esse primeiro princípio, segundo Pressman (2006), deu origem a uma importante abordagem de teste, conhecida como o **teste de caixa preta (black box)**;
- Com base no conhecimento do funcionamento interno do produto, realizam-se testes para assegurar que todas as peças dele estejam completamente ajustadas e realizando a contento sua função. Este segundo princípio, segundo Pressman (2006), é denominado **teste de caixa branca (white box)**, devido ao fato de que maior ênfase é dada ao desempenho interno do sistema (ou do produto).

A. Teste da Caixa Preta

Também chamado de Teste Funcional, é uma abordagem na qual o teste é derivado da especificação de programas ou de componentes. Ele analisa o sistema como uma caixa preta em que seu comportamento só pode ser avaliado por meio da análise das entradas e saídas relacionadas, ou seja, refere-se aos testes que são realizados nas interfaces do software.

O teste é usado para demonstrar que as funções dos softwares são operacionais, que a entrada é adequadamente aceita, a saída é corretamente produzida e que a integridade das informações externas é mantida.

Essa técnica examina aspectos de um sistema **sem** se preocupar muito com a estrutura lógica do software, pois concentra-se nos requisitos funcionais.

O teste procura descobrir erros nas seguintes categorias:

- Funções incorretas ou ausentes.
- Erros de interface.
- Erros nas estruturas de dados ou no acesso a Bancos de Dados (BD) externos.
- Erros de desempenho.
- Erros de inicialização e término.

B. Teste da Caixa Branca ou de Estrutura

O teste tem por objetivo determinar defeitos na estrutura interna do produto com técnicas que exercitem possíveis caminhos e erros de execução.

Nessa técnica, são testados caminhos lógicos através do software, fornecendo casos de testes que colocam à prova conjuntos específicos de condições e/ou laços definidos no sistema.

Ele é também chamado de Teste de Caixa Clara, Teste de Caixa de Vidro e Teste Estrutural. Ele fornece o uso de mais informações sobre o objeto testado do que Teste de Caixa Preta. Esse teste é dispendioso e tem limitada cobertura semântica.

Apesar da importância do Teste de Caixa Branca, não se deve guardar a falsa ideia de que sua realização num produto de software vai oferecer a garantia de 100% de correção deste ao seu final. Isso porque, mesmo no caso de programas de pequeno e médio porte, a diversidade de caminhos lógicos pode atingir um número bastante elevado, representando um grande obstáculo para o sucesso completo desta atividade (PRESSMAN, 2006).

Esse teste baseia-se num minucioso exame dos detalhes procedimentais no qual o “status do programa” pode ser examinado em vários pontos para determinar se o resultado esperado ou estabelecido corresponde ao real.

Devemos ter em mente que testes exaustivos apresentam certos problemas logísticos, porque, mesmo para pequenos programas, o número de caminhos lógicos possíveis pode ser muito grande. Mesmo assim, essa técnica nunca deverá ser descartada, pois cada caso de teste tem sua funcionalidade em momentos específicos estabelecidos pela estratégia de teste definida.

Estratégia de teste ou também chamado de baterias de testes

As políticas de teste podem ser criadas seguindo estas estratégias:

- A) Teste de unidade.
- B) Teste de integridade.
- C) Teste de validação.
- D) Teste de sistema.

Observação: no padrão nomenclatura IEEE, a bateria de teste de validação é chamada de teste de sistema, reservando o termo “teste de validação” para aquele realizado pelo cliente, como parte de um procedimento de aceitação do produto. Por efeitos didáticos, resolvemos definir, separadamente, o que é o teste de validação e de sistemas. Entretanto, você deverá perceber que os conteúdos desses dois grupos são semelhantes.

Nas figuras 4 e 5 podemos observar a estrutura e a relação das estratégias de testes.

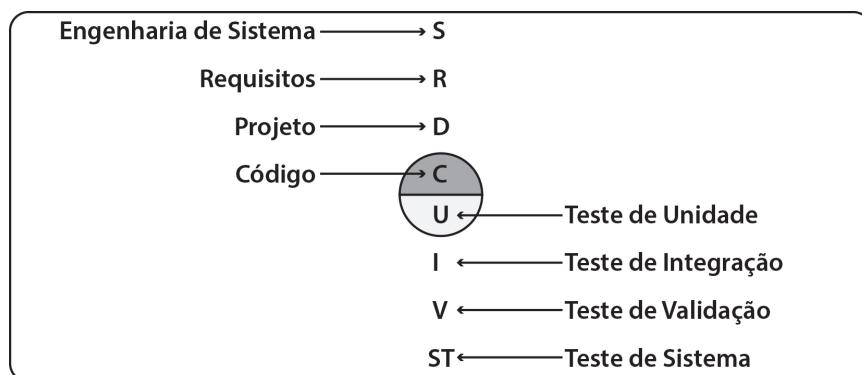


Figura 4 - Estrutura das estratégias de teste.

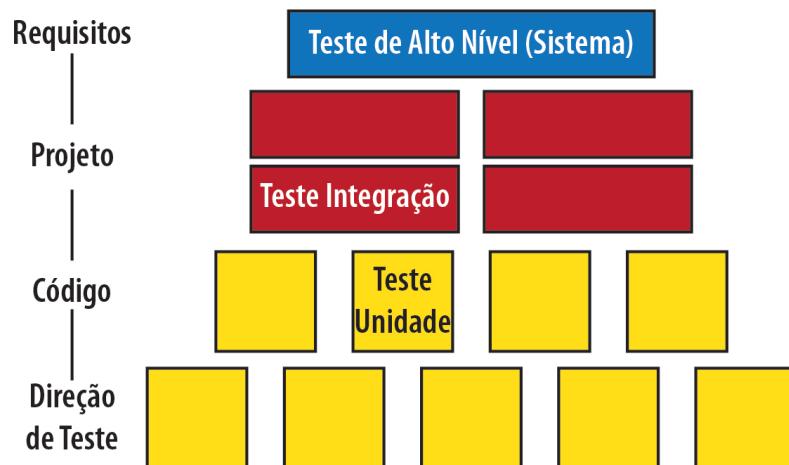


Figura 5 - Estratégia de Teste. No teste de unidade de teste, é verificado o código desenvolvido; no teste de integração, o projeto; e no teste de alto nível, se todos os requisitos funcionais e não funcionais foram satisfeitos.

A. Teste de Unidade

Esse teste tem por objetivo **verificar** os elementos que podem ser testados logicamente. Geralmente, concentra-se em cada unidade do *software*, de acordo com o que é implementado no código-fonte; cada módulo é testado individualmente, garantindo que ele funcione adequadamente.

Geralmente, são utilizadas técnicas de Caixa Branca.

B. Teste de Integração

Tem por objetivo **verificar** as interfaces entre as partes de uma arquitetura do produto. É uma técnica sistemática para a construção da estrutura de programa, realizando, ao mesmo tempo, testes para descobrir erros associados a interfaces e para verificar erros de integração.

O objetivo é, a partir dos módulos testados em nível de unidade, construir uma estrutura de programa que foi determinada pelo projeto.

Utiliza, principalmente, as técnicas de Teste de Caixa Preta.

Os Testes de Integração podem ser feitos por:

Integração incremental: o programa é construído e testado em pequenos segmentos, onde os erros são mais fáceis de serem isolados e corrigidos; as interfaces têm maior probabilidade de serem testadas completamente e uma abordagem sistemática ao teste pode ser aplicada.

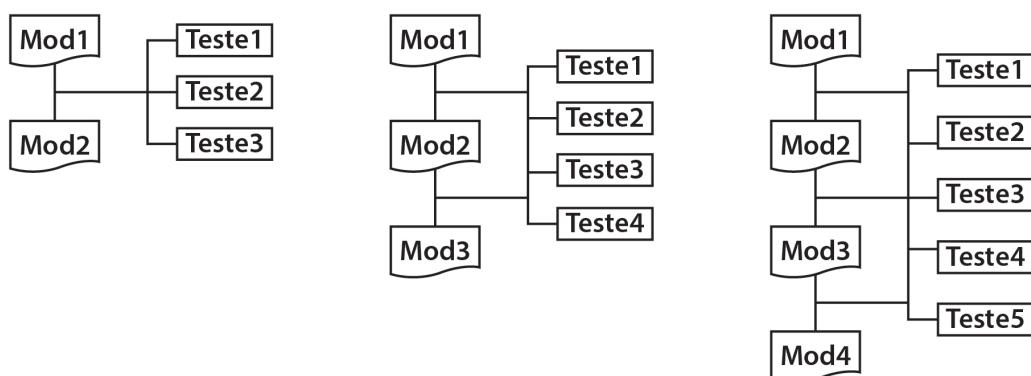


Figura 6 - Teste de Integração incremental.

Fonte: (SOMMERVILLE, 2007, p. 358).

Integração não incremental: abordagem *big-bang*. O programa é testado como um todo, mas prepare-se, o resultado é o caos. Quando erros são encontrados, a correção é difícil porque o isolamento das causas é complicado devido à amplitude por estar sendo testado o programa completo.

Integração top-down: os módulos são integrados movimentando-se de cima para baixo, através da hierarquia de controle. Inicia-se por meio de um módulo de controle principal e a partir dele os outros são incorporados à estrutura de uma maneira *depth-first* (primeiro pela profundidade) ou *breadth-first* (primeiro pela largura).

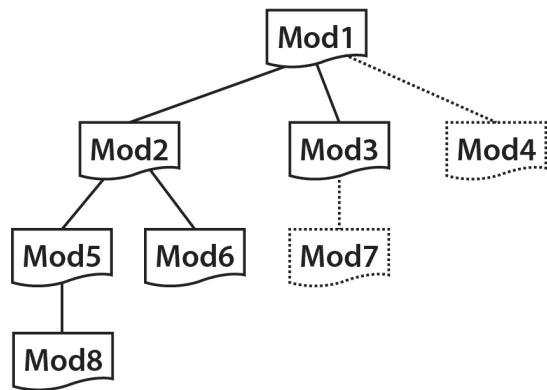


Figura 7 - Integração Top-Down. Depth-first: MOD2, MOD5, MOD8.
Breadth-first: MOD2, MOD3, MOD4.

C. Teste de Validação

Ao término da atividade de teste de integração, o *software* está completamente montado como um pacote, erros de interface foram descobertos e corrigidos e uma série final de testes de *software* – os testes de validação – podem ser inicializados.

Esse teste tem por objetivo **validar** o produto, ou seja, verificar os requisitos estabelecidos como parte da análise de requisitos de software.

É considerado um teste de alto nível, pois verifica se todos os elementos combinam-se adequadamente e se a função/desempenho global do sistema é obtida.

- A validação é bem-sucedida quando o software funciona de uma maneira razoavelmente esperada pelo cliente.
- A especificação de requisitos do software contém os critérios de validação que formam a base para uma abordagem ao Teste de Validação.
- A validação é realizada por meio de uma série de Testes de Caixa Preta que demonstram a conformidade com os requisitos.
- O plano de Teste de Validação visa garantir:
 - Que todos os requisitos **funcionais** sejam satisfeitos.
 - Que todos os requisitos de desempenho sejam conseguidos.
 - Que a documentação (projeto, programador e usuário) esteja correta.
- Requisitos não funcionais (portabilidade, compatibilidade remoção de erros, manutenibilidade etc.) sejam cumpridos.
- Revisão da configuração.
 - Verifica se os elementos de configuração de software foram adequadamente catalogados e desenvolvidos a ponto de apoiar a fase de manutenção.

Quando **um projeto de software é personalizado para vários clientes**, podemos realizar os testes de aceitação por meio das técnicas denominadas Alfa e Beta.

- **Teste Alfa:** o cliente testa o produto **nas instalações do desenvolvedor**. O software é usado num ambiente controlado com o desenvolvedor “olhando sobre os ombros” do usuário e registrando erros e problemas.
- **Teste Beta:** é realizado o teste nas **instalações do cliente** pelo usuário final do software. O desenvolvedor não está presente. O cliente registra os problemas que são encontrados e relata-os ao desenvolvedor em intervalos regulares.

Observação: quando um produto de software é desenvolvido de forma personalizada (para um cliente), não se utiliza o Teste Alfa e nem o Beta. Os testes de aceitação são realizados pelo usuário final para capacitá-lo e para validar todos os requisitos e variam entre um teste informal e uma série de testes planejados que podem ser executados nas dependências do usuário ou do desenvolvedor de acordo com o estipulado entre as partes.

D. Teste de Sistema

Nesta categoria, o *software* e outros elementos do sistema são testados como um todo. Envolve uma série de diferentes testes, cujo propósito primordial é provar completamente o sistema baseado em computador.

Teste de recuperação: um sistema deve ser capaz de tratar determinados erros e voltar à função global sem grandes problemas. Esse teste força o sistema a provocar esses erros para verificar como se comportaria.

Teste de regressão: realiza novos testes previamente executados para assegurar que alterações realizadas em partes do produto não afetem as partes já testadas. Alterações realizadas, especialmente na fase de manutenção, podem introduzir erros nas partes previamente testadas.

Teste de segurança: tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de fato, de acessos indevidos.

Teste de estresse: também chamado de “teste de tortura”, é utilizado para testar a estabilidade de um sistema. Esse teste direciona a capacidade operacional do sistema a ponto de verificar quando e se acontece o que é denominado ponto de ruptura, ou seja, em qual situação o sistema para de funcionar ou deixa de funcionar de modo seguro.

Teste de desempenho: após o sistema ter sido integrado completamente, é possível testá-lo em relação à sua confiabilidade e desempenho. Testes de desempenho são testes de aplicações de software para garantir que o sistema funcionará como esperado, independentemente da carga de trabalho.

Teste de release: também chamado de teste funcional, garantirá a entrega do sistema ao cliente com funcionalidade, confiabilidade e desempenho especificados e esperados e, principalmente, a inexistência de falhas.

As duas grandes dúvidas quando começamos a conhecer e a perceber a importância dos testes são:

- 1) Quando realizamos testes?**
- 2) Como saber se já testamos o suficiente?**

Uma resposta definitiva é difícil de oferecemos, mas, segundo Sommerville (2007), existe uma resposta pragmática que pode ser dada: jamais será completada uma atividade de teste; na verdade, o que acontece é que são transferidas essas responsabilidades do projetista para a equipe de teste e desta para o cliente. Outra resposta aceitável pode ser que você deverá interromper as suas atividades de teste quando seu tempo provisionado para isso se esgotar ou acabar o dinheiro previsto para essa etapa.

Até agora, apenas apresentamos algumas atividades de teste e reconhecemos sua importância no projeto de software. Entretanto, devido ao nosso tempo, algumas técnicas não puderam ser abordadas. Sugerimos que você pesquise um pouco mais sobre:

- Particionamento de Equivalência ou Partição de Equivalência.
- Teste de caminho (básico).
- Análise do valor limite.
- Testes top-down e bottom-up.
- Testes orientados a objetos.
- Teste de classe de objetos:
 - Teste de integração de objetos.
 - Procure fazer a relação dessas técnicas com as estratégias abordadas.

Manutenção de software

A manutenção de software é o processo geral de modificações de um sistema depois que ele começa a ser utilizado.

A manutenção pode ser simples, corrigir erros de código, ou complexa, com a finalidade de acomodar novos requisitos.

Geralmente, a manutenção não está relacionada a grandes alterações do *software*, estando mais associada à acomodação de novos componentes de sistemas.

A manutenção pode ser considerada a fase mais problemática do ciclo de vida de um *software*, pois os sistemas devem continuar rodando e as alterações são inevitáveis.

Embora na prática não haja muita diferença entre tipos de manutenção, podemos dividi-las nas seguintes categorias:

- **Corretivas:** manutenção para corrigir defeitos no software.
- **Adaptativas:** adaptar o software a um ambiente operacional diferente.
- **Perfectivas:** mudanças para fazer acréscimos a funcionalidades do sistema ou modificá-lo. Geralmente, pode incluir atender aos pedidos do usuário para:
 - Modificar funções existentes.
 - Incluir novas funções.
 - Efetuar melhorias em geral.
- **Melhorar** a manutenibilidade ou confiabilidade futura e fornecer uma base melhor para posterior aprimoramento.

O custo de manutenção deve sempre ser provisionado no valor total do projeto.

Segundo Sommerville (2005, p. 519):

Os custos de manutenção, com uma proporção dos custos de desenvolvimento variam de um domínio de aplicação para outro. Para sistemas de aplicação de negócio [...] os custos de manutenção eram amplamente comparáveis aos custos de desenvolvimento de sistemas. Para os sistemas embutidos de tempo real, os custos de manutenção podem ser até quatro vezes maiores do que os custos de desenvolvimento. Os elevados requisitos de confiabilidade e desempenho desses sistemas podem exigir que os módulos sejam estreitamente vinculados e, como consequência, difíceis de serem modificados.

Sempre a melhor opção para diminuir os custos de manutenção é investir em esforço ao projetar e implementar um sistema, pois é muito mais caro inserir novas mudanças no sistema (adicionar funcionalidades) depois da entrega, do que durante o desenvolvimento. “Boas técnicas de Engenharia de Software,, como a especificação precisa, o uso do desenvolvimento orientado a objetos e do gerenciamento de configurações, contribuem para a redução do custo de manutenção”. (SOMMERVILLE, 2005, p. 519)

Existem alguns problemas clássicos que podem interferir numa boa manutenção após a entrega do sistema para o cliente:

- É impossível ou difícil traçar a evolução do software devido às várias versões existentes.
- As alterações feitas não são adequadamente documentadas.
- É difícil rastrear o processo pelo qual o sistema foi criado.
- É muito difícil entender programas de outras pessoas, principalmente quando elas não estão por perto para explicar.
- Documentação inexistente, incompreensível ou desatualizada.
- A maioria dos softwares não foi projetada para apoiar alterações.
- A manutenção não é vista como trabalho valorizado e sim como um retrabalho.

A manutenção varia de acordo com o tipo do software, dos processos de desenvolvimento e do pessoal envolvido no processo. Algumas vezes, ela pode surgir de conversas informais, o que pode levar a um processo de **manutenção informal**. Entretanto, o mais adequado e utilizado em grandes projetos é a **manutenção formal**, com documentos específicos descrevendo etapas e características de cada fase ou etapa.

- a) Basicamente, podemos dividir o processo de manutenção em etapas:
- b) O usuário, a gerência ou clientes solicitam a mudança.
- c) Se as mudanças forem aceitas, um novo *release* do sistema será planejado. Durante esse planejamento, todas as solicitações e implicações são analisadas e o procedimento de alteração idealizado. Nessa etapa, os novos requisitos do sistema que foram propostos são analisados e validados.
- d) Mudanças são implementadas e validadas em uma nova versão do sistema.
- e) Nova versão do sistema é liberada.

Tais etapas podem melhor ser compreendidas na Figura 9.

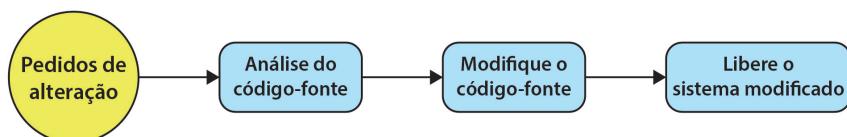


Figura 8 - Processo de reparos de emergência.

Fonte: Sommerville (2005, p. 522)

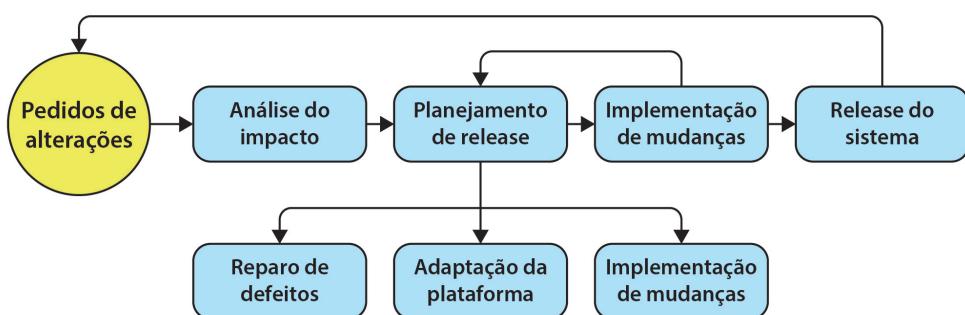


Figura 9 - Visão geral do processo de mudança.

Fonte: Sommerville (2005, p. 522).

Claro que há mudanças. Elas implicam o bom funcionamento imediato do sistema. Embora seja feito com um pouco menos de burocracia, o processo de análise e validação das alterações é sempre fundamental para que uma versão adaptada do sistema seja liberada para o cliente e não cause maiores danos. Entretanto, para assegurar que os requisitos, o projeto de software e o código se tornem inconsistentes, essas mudanças, assim como as categorizadas informais, devem sempre ser documentadas e a documentação geral do projeto de software ser atualizado (Figura 10).

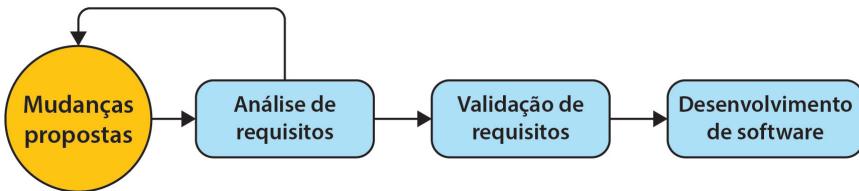


Figura 10 - Implementação de mudanças.

Fonte: Sommerville (2005, p. 522).

Gerentes odeiam surpresas. Dessa forma, uma boa equipe de desenvolvimento sempre procurará prever quais mudanças no sistema possivelmente serão solicitadas, quais partes podem causar maior dificuldade de alteração.

Segundo Sommerville (2005), considerando as previsões de mudanças, podemos dividi-las em 3 e, a cada uma delas, associá-las a determinadas perguntas:

- A) Previsão de facilidade de manutenção: que partes do sistema serão mais dispendiosas para manter?
- B) Previsão de mudanças no sistema: que partes do sistema provavelmente serão mais afetadas pelos pedidos de mudanças? Quantos pedidos de mudanças podem ser esperados?
- C) Previsão de custo de manutenção: quais serão os custos de manutenção durante o tempo de vida útil do sistema? Quais serão os custos de manutenção do sistema no próximo ano?

Prever as respostas a essas perguntas garantirá um maior controle das consequências que podem ser geradas, no sistema e na documentação, a partir da solicitação de mudanças.

Conclusão

Como podemos observar, planejar testes e manutenção faz parte dos conceitos fundamentais da idealização de um sistema.

Realizar testes está muito além de apenas compilar: está relacionado ao perfeito funcionamento, num prazo longo, de um sistema.

Entretanto, embora projetemos sistemas para ter uma vida, relativamente, longa, manutenções sempre ocorrerão. Muitas serão solicitadas pelos usuários e temos que avaliar a pertinência desses pedidos, outras serão oferecidas pelo próprio fabricante do produto com o objetivo de atualizar e melhorar a eficiência do produto de software.



Importante

Não importa como essas mudanças, ou casos de testes, são solicitados, o importante é que tanto um quanto o outro façam parte de atividades bem planejadas e principalmente, quando executadas, sejam adequadamente documentadas.

Material Complementar

Como nesta unidade abordamos os conceitos gerais da Engenharia de Software, nossa sugestão de material complementar é:

Teste:

- DIAS NETO, A. C. **Introdução aos testes de software**. Disponível em: <http://www.comp.ita.br/~mluisa/TesteSw.pdf>

Manutenção:

- ESPINDOLA, R. S.; MAJDENBAUM, A.; AUDY, J. L. N. **Uma análise crítica dos desafios para Engenharia de Requisitos em Manutenção de Software**. Disponível em: http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER04/Rodrigo_Espindola.pdf

Referências

Bibliografia fundamental

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

Bibliografia básica

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação**. 4. ed. Rio de Janeiro: LTC, 1999.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação Gerenciais**. Administrando a empresa digital. 5. ed. São Paulo: Pearson Education do Brasil, 2006.

PFLEEGER, S. L. **Engenharia de Software**: teoria e prática. São Paulo: Prentice Hall, 2004.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 2006.

SOMMERVILLE, I. **Engenharia de Software**. 6. ed. São Paulo: Pearson Addison Wesley, 2005.

Bibliografia complementar

ALCADE, E.; GARCIA, M.; PENUELAS, S. **Informática Básica**. São Paulo: Makron Books, 1991.

FAIRLEY, R. E. **Software engineering concepts**. New York: McGraw-Hill, 1987.

IEEE. **Software Engineering Standards**. 2013. Disponível em: <http://www.ieee.org/portal/innovate/products/standard/ieee_soft_eng.html>. Acesso em: 10 dez. 2013.

LUKOSEVICIUS, A. P.; CAMPOS FILHO, A. N.; COSTA, H. G. **Maturidade em gerenciamento de projetos e desempenho dos projetos**. Disponível em: <http://www.producao.uff.br/conteudo/rpep/.../RelPesq_V7_2007_07.doc>. Acesso em: 12 nov. 2013.

MAFFEO, B. **Engenharia de Software e especialização de sistemas**. Rio de Janeiro: Campus, 1992.

MICHAELIS. **Moderno dicionário da língua portuguesa**. São Paulo: Cia. Melhoramentos, 1998.

PARREIRA JÚNIOR, W.M. **Apostila de Engenharia de Software**. Disponível em: <http://www.waltenomartins.com.br/ap_es_v1.pdf>. Acesso em: 13 nov. 2013.

PAULA FILHO, W. P. **Engenharia de Software**: fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: LTC, 2001.

Revista Engenharia de Software. Disponível em: <<http://www.devmedia.com.br/revista-engenharia-de-software-magazine>>. Acesso em: 12 nov. 2013.

VONSTA, A. **Engenharia de Programas.** Rio de Janeiro: LTC, 1983.

WIENNER, R.; SINCOVEC, R. **Software engineering with Modula 2 and ADA.** New York: Wiley, 1984.

WIKIPEDIA (2007a). **ISO 9000.** Disponível em: <http://pt.wikipedia.org/wiki/ISO_9000>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007b). **Melhoria de processo de Software brasileiro.** Disponível em: <<http://pt.wikipedia.org/wiki/MPS.BR>>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007c). **CMMI.** Disponível em: <<http://pt.wikipedia.org/wiki/Cmmi>>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007d). **Engenharia de Software.** Disponível em: <http://pt.wikipedia.org/wiki/Engenharia_de_software>. Acesso em: 1 fev. 2007.

Anotações





Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br

Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade
Cruzeiro do Sul



UNICID
Universidade
Cidade de S. Paulo



UNIFRAN
Universidade
de Franca



UDF
Centro
Universitário



Módulo
Centro
Universitário