DEDMD

LIVRO

UNIDADE 4

Engenharia de Software

Os testes de software

Roque Maitino Neto

2016
Editora e Distribuidora Educacional S. A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041 -100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

OS TESTES DE SOFTWARE

Convite ao estudo

Olá! Seja bem-vindo à quarta unidade de Engenharia de Software.

Antes de chegar à última etapa do curso, você estudou conceitos fundamentais da disciplina e o jeito tradicional de se conduzir um processo de desenvolvimento de software. Depois, modelos de desenvolvimento mais atualizados foram abordados e você foi convidado a planejar e executar a passagem da X-Soft para uma forma ágil de se criar programas de computador. Em nossa última etapa antes desta, foram abordados os conceitos de qualidade e como atingi-la por meio de padrões consagrados de qualidade. Você foi chamado a colocar como objetivo central da X-Soft o atingimento da excelência em processos e produtos e saiu-se muito bem nessa missão.

E por falar em qualidade, eis que ela volta a receber toda a atenção. Nas próximas quatro aulas serão tratados com mais profundidade temas relacionados ao teste e à evolução de um software.

O objetivo geral desta unidade é que você domine conceitos e práticas associadas à atividade de teste. Cada seção, em específico, tem como objetivo possibilitar ao aluno:

- Conhecer os fundamentos da atividade de teste de software, teste funcional e estrutural.
- Conhecer os conceitos e de que forma se dá a execução do *Test-Driven Development* (TDD).
- Conhecer o que são e como implementar o teste de release e teste de usuário.

- Conhecer e saber fazer a manutenção e compreender como isso se aplica em meio à evolução do software.

Como consequência do alcance desses objetivos, você deverá desenvolver a competência técnica de conhecer e conduzir processos de teste de software. Você atingirá os objetivos e desenvolverá as competências desta unidade por meio da resolução dos desafios que serão colocados em cada seção da unidade, descritas aqui:

1. Selecionar e registrar os casos de teste que serão utilizados na aplicação de teste funcional num dos programas da X-Soft.
2. Aplicar teste funcional em um dos programas da X-Soft, com relato das atividades desempenhadas.
3. Relatar a avaliação dos resultados obtidos com a aplicação do teste de software.
4. Aplicar modelo de estimação de esforço de manutenção de um produto da X-Soft.

Nas próximas páginas será detalhada a primeira parte de nossa missão, proposto conteúdo teórico sobre teste de software e novas abordagens da situação-problema. Bom trabalho!

Seção 4.1

Testes de software: fundamentos, casos de teste, depuração, teste estrutural, teste funcional

Diálogo aberto

É notável a evolução pela qual a X-Soft tem passado. Desde seu início, como uma empresa que sequer tinha um procedimento definido para desenvolvimento de seus produtos, até sua consolidação como uma organização reconhecidamente apta a entregar software de qualidade, a X-Soft experimentou a ascensão que só uma empresa liderada por gente competente é capaz de ter. Todos os modelos e procedimentos adotados já fazem parte da rotina da empresa e a equipe sente-se à vontade com eles.

Afinal, o que mais a X-Soft deveria incluir em seu dia a dia para manter sua evolução? Qual aspecto do seu modelo de desenvolvimento ainda pode ser melhorado? Ao tratar das questões introdutórias de teste de software, esta seção propõe-se a responder essas questões e esclarecer como a atividade de teste pode contribuir de forma decisiva para a qualidade do produto final e para a consequente redução do investimento de tempo em atividades de retrabalho.

Usando como apoio os conceitos que serão tratados nesta seção, você deverá superar o desafio que se apresenta. É sua missão planejar e registrar os casos de teste que serão utilizados na aplicação de teste funcional num dos programas da X-Soft. Resumidamente, um caso de teste é um dado de entrada que leva o programa a executar partes de seu código e a respectiva saída esperada para essa entrada. A evolução da aula dará a você melhores condições para entender e aplicar esse conceito.

Bons estudos!

Não pode faltar

Na primeira unidade de nosso curso, o teste foi colocado como uma das fases finais do processo de desenvolvimento de um produto no modelo tradicional. A atividade foi situada como etapa seguinte à codificação e conceituada como a execução de um programa visando à detecção de defeitos em seu código.

Nas aulas seguintes, ao tratarmos de modelos ágeis, os testes foram abordados como atividade vinculada à codificação, com indicações de que o próprio desenvolvimento deveria ser guiado pelos testes. Avançando mais um pouco no conteúdo, as atividades que visam garantir a qualidade de um produto – das quais o teste faz parte – foram diluídas por todo o processo e, de certa forma, desconstruíram a ideia de que o teste deve ser encarado como uma fase estanque, sem comunicação e sem abrangência nas demais fases do processo.

Pois bem, qualquer que seja a forma pela qual encaremos o teste e onde quer que o situemos no processo, ele sempre será atividade indispensável e de alta criticidade na produção de programas de qualidade. Tratá-lo sem a devida importância seria arriscar toda a produção supondo que programadores não cometem erros, o que, de longe, é um equívoco.

Nas linhas seguintes você terá contato com conceitos ligados ao tema e à apresentação de duas técnicas bastante usadas para a realização de testes, o que servirá de base para a superação do nosso desafio. Sigamos.

Fundamentos

Um teste – ou um **processo de teste** – consiste em uma sequência de ações executadas com o objetivo de encontrar problemas no software, o que aumenta a percepção de qualidade geral do software e garante que o usuário final tenha um produto que atenda às suas necessidades (PINHEIRO, 2015).

É sempre bom destacar que o objetivo do teste é encontrar problemas no software, e não garantir que o programa é livre de defeitos. Se o processo de teste não revelar defeitos, há que se aprimorar os casos de teste e o processo empregado. Não se pode acreditar que o sistema não possui problemas se o teste não os revelar.

O processo de teste é normalmente separado em quatro grandes etapas:

- Planejamento: nesta etapa deve ser definido quem executa os testes, em que período, com quais recursos (ferramentas de teste e computadores, por exemplo) e qual será a técnica utilizada (técnica estrutural ou técnica funcional, por exemplo).

- Projeto de casos de teste: aqui são definidos os casos de teste que serão utilizados no processo. No próximo item, este conceito será detalhado.
- Execução do programa com os casos de teste: nesta etapa, o teste é efetivamente realizado.
- Análise dos resultados: aqui verifica-se se os testes retornaram resultados satisfatórios.

Na sequência, um item muito importante para a resolução do problema proposto será abordado.

Casos de teste

Um caso de teste é o par formado por uma entrada no programa e a correspondente saída esperada, de acordo com os requisitos do sistema. Entenda o conceito de entrada como o conjunto de dados necessários para a execução do programa. A saída esperada é o resultado de uma execução do programa ou função específica. Imagine que estejamos colocando sob teste um programa que valida datas inseridas pelo usuário. Um caso de teste possível seria (25/12/2016; válida). Ao receber a entrada 25/12/2016, o programa de validação de data deveria retornar "data válida".

É certo que a boa escolha dos casos de teste é fundamental para o sucesso da atividade. Um conjunto de casos de teste de baixa qualidade pode não exercitar partes críticas do programa e acabar não revelando defeitos no código. Se o responsável pelos testes usasse apenas datas válidas como entradas, a parte do programa que trata das datas inválidas não seria executada, o que prejudicaria a confiabilidade do processo de teste e do produto testado. Observe os casos de teste que seguem:

t1= {(13/2/2016;válida), (30/2/2004;inválida); (25/13/2000;inválida); (29/2/2016;válida), (29/2/2015;inválida), (##/1/1985;inválida)}. Com esse cenário, certamente a maior parte do código será coberta e a chance de detecção de defeitos aumentará.

Parece claro que o procedimento de testes está diretamente relacionado à boa escolha e ao bom uso dos casos de teste. Idealmente, cada conjunto de casos de teste deverá estar associado a um grande requisito diferente a ser testado. Para que não se corra o risco de defini-los incorretamente, é necessário planejamento e o bom conhecimento da aplicação. Uma boa forma de se abordar o problema é a que segue (PINHEIRO, 2015):

- Definir o ambiente no qual o teste será realizado.
- Definir a entrada deste caso de teste.

- Definir a saída esperada para cada entrada.
- Definir os passos a serem realizados para executar os testes.

Quando um caso de teste é executado, o seu resultado deve ser coletado. Podemos assumir diferentes abordagens para definir o resultado da aplicação de um caso de teste específico. A mais comum define as seguintes opções (PINHEIRO, 2015):

- Passou: todos os passos do caso de teste foram executados com sucesso para todas as entradas.
- Falhou: nem todos os passos foram executados com sucesso para uma ou mais entradas.
- Bloqueado: o teste não pôde ser executado, pois o seu ambiente não pôde ser configurado.

Pois bem, dessa forma definimos casos de teste. Há, no entanto, três conceitos especialmente importantes no contexto de teste e que são frequentemente confundidos entre si. Vamos a eles.

Defeito, Falha e Erro

Que expressão que você usa quando um programa simplesmente trava ou não produz o resultado que se espera dele? Tudo o que acontece de incomum em um programa pode ser chamado de erro? Observe os conceitos:

Defeito: trata-se de deficiência algorítmica que, se ativada, pode levar a uma falha. Vamos a um exemplo. Observe o trecho que segue, escrito em pseudocódigo:

a = -1;

b = 0;

enquanto a < 0 faça

b = *b* + 1;

imprima (*b*);

imprima (*a*);

fim_enquanto;

Em algum momento o valor da variável a deixará de ser menor que zero? Estamos diante de um defeito, mais precisamente um laço infinito. Se o caso de teste escolhido for capaz de exercitar esse trecho do código, o defeito se manifestará e então teremos uma falha observável. O programa provavelmente será interrompido.

Falha: é tida como um não funcionamento do programa, provavelmente provocada por um defeito. No entanto, uma falha também pode ser atribuída a uma queda na comunicação ou a um erro na leitura do disco, por exemplo.

Erro: ocorre quando o resultado obtido em um processamento e o que se esperava dele não são coincidentes. Um erro também está associado a uma violação nas próprias especificações do programa. Por exemplo, um usuário não autorizado consegue acessar determinado módulo do programa (esse é o resultado obtido), sendo que seu nível de privilégios não deveria permitir que o fizesse (esse era o resultado esperado).

Pesquise mais

Os conceitos de defeito, falha e erro não são uniformes nas referências. Outros conceitos básicos também podem variar entre autores. Uma boa fonte introdutória e conceitual pode ser obtida em <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em: 14 fev. 2016.

Embora estejamos diante de conceitos com diferenças sutis e que frequentemente são confundidos, a devida separação em três certamente facilitará o entendimento de outros conceitos e procedimentos mais adiante.

Depuração

Enquanto testar significa executar o software para encontrar defeitos desconhecidos, a depuração é a atividade que consiste em buscar no código a localização desses erros. O fato de saber que o programa não funciona não implica, necessariamente, em já se saber também em qual ou quais linhas o problema está. Os ambientes de programação atuais oferecem recursos para depuração do programa. Durante esse processo, o valor assumido pelas variáveis sob inspeção em cada passo do algoritmo pode ser observado. Além disso, alguns pontos de parada da execução do programa podem ser inseridos no código. Tudo para possibilitar que o testador identifique e isole o defeito no código.

Assimile

O término bem-sucedido do processo de compilação significa que o código apresenta correção sintática e semântica e, portanto, pode ser executado pelo computador. No entanto, isso não garante, definitivamente, que o programa esteja livre de erros de lógica ou de cálculo.

Como todo processo desenvolvido com base científica, a atividade de teste também inclui maneiras estruturadas e consagradas para a sua realização. Dependendo da disponibilidade do código-fonte, da ferramenta de teste escolhida e das características do programa a ser testado, a técnica funcional ou a técnica estrutural podem ser aplicadas.

Imagine o programa como uma caixa. Quando o testador não tem acesso ao código fonte, ele está lidando com uma caixa preta, cujo interior não se consegue ver. Daí o teste funcional também ser conhecido como caixa preta. A técnica estrutural, por sua vez, é também conhecida como caixa branca, cujo interior se pode ver. Essa comparação é feita justamente pela disponibilidade do código fonte e das estruturas internas do programa.

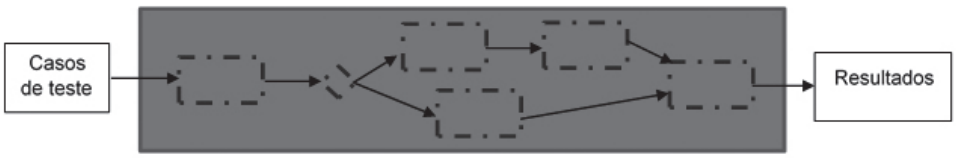
Segue uma descrição mais aprofundada dessas técnicas:

Técnica de Teste Funcional

Esta técnica baseia-se nas especificações do software para derivar os requisitos de teste. O teste é realizado nas funções do programa, daí o nome funcional. Não é seu objetivo verificar como ocorrem internamente os processamentos, mas se o algoritmo inserido produz os resultados esperados (BARTIÉ, 2002).

Uma das vantagens dessa estratégia de teste é o fato de ela não requerer conhecimento de detalhes da implementação do programa. Sequer o código-fonte é necessário. Observe uma representação dessa técnica na Figura 4.1:

Figura 4.1 | Visão de teste de caixa preta



Fonte: Bartié, 2002, p. 105.

O planejamento do teste funcional envolve dois passos principais: identificação das funções que o software deve realizar (por meio da especificação dos requisitos) e a criação de casos de teste capazes de checar se essas funções estão sendo executadas corretamente.

Apesar da simplicidade da técnica e apesar de sua aplicação ser possível em todos os programas cujas funções são conhecidas, não podemos deixar de considerar uma dificuldade inerente: não se pode garantir que partes essenciais ou críticas do software serão executadas, mesmo com um bom conjunto de casos de teste.

Exemplificando

Imagine uma função que valida nomes de identificadores em uma linguagem de programação criada por você. As condições para validação são: tamanho do identificador entre 1 e 6 caracteres, primeiro caractere necessariamente deve ser letra e caracteres especiais não são permitidos. O Tabela 4.1 resume as condições de validade do identificador e a aplicação do teste.

Tabela 4.1 | Resultado da aplicação da técnica funcional com os casos de teste dados

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho *t* do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caractere *c* é uma letra	Sim (3)	Não (4)
Só contém caracteres válidos	Sim (5)	Não (6)

Fonte: O autor (2016)

Exemplo de Conjunto de Casos de Teste

T0 = {(a1,Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)}

(1, 3, 5) (4) (6) (2)

Conheça agora uma técnica de teste bastante eficiente na descoberta de defeitos.

Técnica de Teste Estrutural

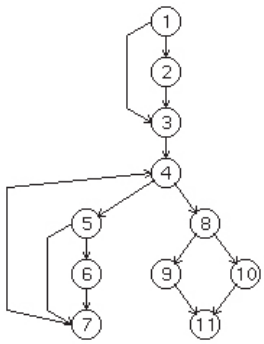
Os testes estruturais (ou de caixa branca) são assim conhecidos por serem baseados na arquitetura interna do programa. De posse do código-fonte (ou do código-objeto) e, se for o caso, das estruturas de banco de dados, o profissional designado para a atividade submete o programa a uma ferramenta automatizada de teste.

A ferramenta constrói uma representação de programa conhecida como grafo de programa. No grafo, os nós equivalem a blocos indivisíveis, ou seja, não existe desvio de fluxo do programa para o meio do bloco e, uma vez que o primeiro comando do bloco é executado, os demais comandos são executados sequencialmente. As arestas ou arcos representam o fluxo de controle entre os nós. Observe o trecho de código a seguir, escrito em linguagem C. Ele valida identificadores de acordo com os critérios dados no quadro *Exemplificando* (DELAMARO, 2004). Os números à frente de cada linha representam os nós do grafo que vem logo a seguir.

```
/* 01 */ {
/* 01 */ char achar;
/* 01 */ int length, valid_id;
/* 01 */ length = 0;
/* 01 */ printf ("Identificador: ");
/* 01 */ achar = fgetc (stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */          length = 1;
/* 03 */ achar = fgetc (stdin);
/* 04 */ while (achar != '\n')
/* 05 */ {
/* 05 */          if (!(valid_f(achar)))
/* 06 */                  valid_id = 0;
/* 07 */          length++;
/* 07 */          achar = fgetc (stdin);
/* 07 */ }
/* 08 */ if (valid_id && (length >= 1) && (length < 6) )
/* 09 */          printf ("Valido\n");
/* 10 */ else
/* 10 */          printf ("Invalido\n");
/* 11 */ }
```

A submissão do código a uma dada ferramenta de teste criará a representação vista na Figura 4.2. Note que cada trecho identificado com um número no código é reproduzido em um nó do grafo.

Figura 4.2 | Representação em grafo de código sob teste



Fonte: Delamaro (2004, p. 11).

A função do testador, então, é encontrar casos de teste que, durante uma execução do programa, sejam capazes de exercitar os caminhos, nós e arcos do grafo. Um caminho simples, por exemplo, seria é dado por (2,3,4,5,6,7). O caminho completo é representado por (1,2,3,4,5,7,4,8,9,11).

Aí está, resumidamente, colocada a técnica funcional. Apesar de ser mais eficiente em encontrar defeitos no código, sua aplicação é mais dispendiosa e complexa.

Reflita

Se é certo que um teste, por melhor que seja executado, não conseguirá assegurar que o programa é 100% livre de defeitos, qual a indicação de que é chegado o momento de interromper os testes? Pois é justamente durante o planejamento do teste que os critérios de parada da atividade são definidos.

Pois bem, a devida introdução ao teste de software foi dada. Existem outras tantas técnicas e métodos de aplicação de teste cuja abordagem extrapolaria os objetivos desta seção. Embora de difícil execução e de custo relativamente alto, a atividade de teste é fundamental no processo de criação de programas. É certo que desenvolvedores têm investido em ferramentas e em material humano para conferir graus elevados de qualidade aos seus produtos. E é importante que seja assim.

Faça você mesmo

Pesquise sobre a ferramenta de teste chamada JaBUTi e prepare um resumo sobre seu funcionamento. Comece sua pesquisa por <http://ccsl.icmc.usp.br/pt-br/projects/jabuti>. Acesso em: 14 fev. 2016.

SEM MEDO DE ERRAR!

A atividade de teste, quando tratada de modo correto e profissional, deve ser bem planejada e estruturada antes de ser executada. Não se pode conceber que uma empresa que busque a excelência em seus produtos negligencie esta etapa tão importante do desenvolvimento.

A preparação da atividade não pode deixar de prever os recursos que serão utilizados nos testes, o prazo para sua execução, a técnica a ser utilizada e, em estágio mais avançado de planejamento, os casos de teste a serem utilizados.

A X-Soft, na condição de organização que busca constante aprimoramento em

seus processos e produtos, não poderia deixar de investir boa energia no teste dos seus produtos. Como desafio proposto, é sua missão planejar e registrar os casos de teste que serão utilizados na aplicação de teste funcional num dos programas da X-Soft. Com a finalidade de dar suporte a você, as duas funções do programa a serem testadas são descritas na sequência:

Função 1: validação de CPF do cliente, segundo seu estado de origem. O terceiro dígito da direita para a esquerda identifica a unidade federativa na qual a pessoa foi registrada, de acordo com o Quadro 4.1. Exemplo: o CPF 000.000.008-00 é de alguém cujo estado de origem é São Paulo.

Quadro 4.1 | Unidade Federativa de registro do CPF

0 - Rio Grande do Sul	3 - Ceará, Maranhão e Piauí	6 - Minas Gerais	9 - Paraná e Santa Catarina
1 - Distrito Federal, Goiás, Mato Grosso do Sul e Tocantins	4 - Paraíba, Pernambuco, Alagoas e Rio Grande do Norte	7 - Rio de Janeiro e Espírito Santo	
2 - Amazonas, Pará, Roraima, Amapá, Acre e Rondônia	5 - Bahia e Sergipe	8 - São Paulo	

Fonte: <http://www.geradordecpf.org/>. Acesso em: 28 mar. 2016.

Função 2: verifica atraso no pagamento e aplica acréscimo de 1 ponto percentual sobre cada dia de atraso verificado no pagamento. Uma solução possível para este desafio é a que segue:

Exemplo de conjunto de casos de teste da Função 1:

Formato geral: (número_do_cpf, estado_de_origem; saída_esperada)

t_1={(937.599.133-42,Ceará; válido), (831.469.521-14, Tocantins; válido), (858.178.888-23, São Paulo; válido), (300.168.443-78, Rio Grande do Sul; inválido)}

Exemplo de conjunto de casos de teste da Função 2:

Formato geral: (data_do_vencimento, data_do_pagamento, valor_do_debito; valor_a_ser_pago)

t_2={(10/2/2016, 14/2/2016, 500; 520), (5/2/2016, 2/2/2016,125;125), (15/2/2016, 5/2/2016, 68; 68)}

Atenção!

A conferência da saída obtida pela aplicação do caso de teste é que indicará a existência de problema no código ou não.

Lembre-se

Quanto maior a qualidade do conjunto de casos de teste, maiores serão as chances de o teste detectar defeitos.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.

Aumentando a qualidade dos casos de teste	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software.
2. Objetivos de aprendizagem	Conhecer os fundamentos da atividade de teste de software, teste funcional e estrutural.
3. Conteúdos relacionados	Testes de software: fundamentos, casos de teste, depuração, teste estrutural, teste funcional.
4. Descrição da SP	A RH-Soft, empresa desenvolvedora de sistemas feitos para departamentos de recursos humanos, adotou recentemente procedimento de teste em seus produtos. Depois de codificado o sistema, a empresa passa a executar alguns testes antes da entrega. No entanto, o procedimento não tem evitado que o código permaneça com defeitos que deveriam ter sido descobertos por meio dos testes. Como consequência, a frequência com que erros se manifestam em ambiente de produção tem sido alta. Seu desafio aqui é indicar, no âmbito das atividades do teste funcional, ações para tornar mais eficiente a descoberta de problemas no código.
5. Resolução da SP	A solução do caso requer as possíveis providências: • Planejamento da atividade: além da definição de prazos e recursos a serem usados no teste, durante o planejamento há também que se buscar na especificação de requisitos a totalidade das funções que deverão ser testadas.

	• Seleção dos casos de teste: com base nas funções do sistema, a equipe deverá selecionar casos de teste abrangentes o suficiente para exercitar maior número possível de trechos do código, principalmente aqueles que produzem resultado numérico provenientes de cálculos. Encontrado o problema, a depuração no código deverá ser usada para saná-lo. • Análise dos resultados: aplicados os testes, a equipe deverá avaliar se os casos de teste foram eficientes na detecção dos defeitos. Baixa detecção não significa, necessariamente, programa de alta qualidade.

Lembre-se

O conjunto de casos de teste deve abranger tanto entradas corretas como as incorretas. Para ambas, a saída obtida deverá ser igual a saída esperada, o que é indício da corretude da função testada.

Faça você mesmo

Qual o perfil de um profissional que realiza os testes? Quais as habilidades requeridas para esta função? Faça a leitura dos artigos publicados em: <http://www.devmedia.com.br/artigo-engenharia-de-software-18-profissional-da-area-de-testes/14801> e <http://crowdtest.me/caracteristicas-testador-infografico/> (acesso em: 15 fev. 2016) e fique sabendo.

Faça valer a pena

1. Em relação aos fundamentos da atividade de teste, assinale a alternativa que contém expressões que completam corretamente as lacunas na frase a seguir.

"O processo de teste consiste em executar um programa com o objetivo de revelar a presença de _____; ou, falhando nesse objetivo, aumentar a _____ sobre o programa. Após revelada a presença do defeito, o processo de _____ auxilia em sua busca e correção."

a) funções recursivas; *performance*; teste.

b) defeitos; exatidão; teste exaustivo.

c) casos de teste; confiança; apuração.

d) defeitos; confiança; depuração.

e) defeitos; exatidão; casos de teste.

2. Em relação aos casos de teste, analise as afirmações a seguir:

I. Um caso de teste é o par formado por uma entrada no programa e a correspondente saída esperada.

II. Um caso de teste equivale a uma seção em que os testes são realizados.

III. A escolha correta dos casos de teste tem importância relativa no processo, já que são selecionados pelo cliente.

IV. Os casos de teste são específicos para cada programa submetido a teste.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) IV.

d) II.

e) II e IV.

3. Em relação aos conceitos de defeito, falha e erro, analise as afirmações a seguir:

I. Um laço de repetição construído de forma a deixar o processamento preso em laço infinito configura um defeito.

II. Falha é um não funcionamento do programa.

III. Uma falha no programa pode ser provocada por um problema no hardware.

É verdadeiro o que se afirma em:

a) III, apenas.

b) I, II e III.

c) I e III, apenas.

d) II e III, apenas.

e) I, apenas.

Seção 4.2

O *Test-Driven Development* (TDD)

Diálogo aberto

Além de introduzir conceitos fundamentais de teste, nossa última aula serviu para desconstruir a impressão de que verificação em um programa pode ser feita apenas ao se vasculhar o código ou executar o programa algumas vezes, sem critério ou formalidade. Duas das formas mais comuns de executar verificações num software – o teste funcional e o teste estrutural – foram introduzidas e a importância da correta escolha de casos de teste foi destacada.

Pois bem, a percepção geral na X-Soft é que a introdução de procedimento formal de teste começará em breve a produzir bons resultados. A fase inicial do novo procedimento incluiu a seleção de casos de teste e aquele desafio tornou você e a equipe de desenvolvimento aptos para dar continuidade ao processo de teste. O desafio que se coloca nesta seção é a efetiva aplicação do teste funcional em um dos produtos da X-Soft, incluindo o relato das atividades desempenhadas e as ações empreendidas pelos envolvidos no procedimento.

Além de fornecer nova abordagem do *Test-Driven Development* (TDD) e apresentar esta técnica como própria do modelo ágil de desenvolvimento, é também objetivo desta seção apresentar o tratamento que um erro encontrado por um testador recebe após sua descoberta. Esse tratamento desenhará o que chamaremos de ciclo de vida de um erro e você, na condição de participante ativo do processo de teste, será convidado a sugerir aprimoramentos no desenho deste ciclo, que nasce na descoberta do erro e termina em sua completa correção.

Com este conteúdo, abordado principalmente no item *Não pode faltar*, você seguirá aprimorando sua competência para conhecer e conduzir processos de testes de software, sempre mirando a excelência nos procedimentos e na qualidade dos produtos da X-Soft.

Adiante e bom trabalho!

Não pode faltar

Na Unidade 2 deste material, uma abordagem preliminar de Desenvolvimento Guiado pelos Testes (ou TDD – *Test-Driven Development*) foi feita. O assunto se encaixava perfeitamente no contexto das metodologias ágeis, que na ocasião eram tratadas em detalhes. Depois de termos tratado da maioria das práticas do XP (*Extreme Programming*) e, já na Unidade 3, passado por assuntos relacionados à qualidade, eis que novamente o TDD assume lugar de importância em nosso estudo.

É natural que, depois de conhecer aspectos de qualidade e do teste de software, você encare o assunto de outra maneira e esteja mais apto a expandir seus conhecimentos sobre ele.

Esta seção tratará do caminho que um erro percorre após sua identificação e os responsáveis por cada ação empreendida para corrigi-lo. Cada metodologia de desenvolvimento guarda particularidade na execução do processo de teste e o TDD será abordado como procedimento de teste do modelo ágil, já estudado na Unidade 2.

Na sequência, o conteúdo teórico desses assuntos será abordado.

Engenharia de Testes

Em nossa última seção, quando tratamos de temas iniciais de teste de software, foi dada ênfase às definições de defeito, erro e falha. Entre um exemplo e outro, o texto procurou deixar clara a sutileza entre os conceitos, sem que as diferenças entre os três se perdessem.

No entanto, fora dos limites da esfera acadêmica, qualquer problema que se manifeste no código tende a ser chamado simplesmente de *bug*, ou como referenciaremos aqui, de erro. A intenção do que chamamos de Engenharia de Testes é simples: oferecer entendimento do que é um erro e oferecer formas de encontrá-lo.

Em seções anteriores, foi discutida a impossibilidade de assegurar a completa e irrestrita ausência de erros em um programa, justamente pela condição falível dos programadores. Mas será que apenas os programadores falham?

Um erro ocorre quando uma ou mais das opções a seguir for verdadeira (PINHEIRO, 2015):

- O software **não faz** algo que a especificação estabelece que ele deveria fazer.
- O software **faz** algo que a especificação estabelece que ele não deveria fazer.
- O software **faz** algo que a especificação não menciona.

- O software **não faz** algo que a especificação não menciona, mas deveria mencionar.
- O software é difícil de usar, entender ou, na visão do testador, pode ser visto pelo usuário final como não estando correto.

Fica claro que a especificação incorreta é uma das grandes causas de erros em programas. Mas não só ela. Em sistemas pequenos, erros na codificação respondem por aproximadamente 75% das ocorrências. A tendência é simples de ser entendida: quanto maior o projeto, menor a quantidade de erros na codificação e maior na especificação.

Pelo seu alto potencial em causar problema para equipe e clientes, muito se investe na busca por um perfil ou um padrão na ocorrência de erros. Sabe-se que 85% dos erros podem ser corrigidos em uma hora. É sabido que 80% do esforço é concentrado em apenas 20% do código, o que nos leva ao raciocínio de que os erros estão concentrados em partes específicas do código (PINHEIRO, 2015).

O ciclo de vida de um erro

Compreende o período entre a identificação do erro e sua efetiva correção, sempre durante o processo de testes.

Em cada fase de seu ciclo, o erro recebe uma denominação diferente, providência importante para que os envolvidos identifiquem seu estado com rapidez e correção. O ciclo inclui os seguintes estados, descritos no Quadro 4.2:

Quadro 4.2 | Estados assumidos por um erro em seu ciclo de vida

Estado	Descrição	Definido por
Novo	Novo erro encontrado pelo testador	Equipe de teste
Aberto	Erro que foi revisado e confirmado como um defeito real	Líder do teste
Rejeitado	Erro que não foi confirmado como tal	Líder do desenvolvimento
Atribuído	Erro confirmado e já designado (ou atribuído) a um desenvolvedor para correção	Líder do desenvolvimento
Corrigido	Erro já corrigido e pronto para passar por novo teste	Desenvolvedor
Reaberto	Erro que se manifestou novamente durante o novo teste	Equipe de teste
Fechado	Erro que passou com sucesso pelo novo teste	Equipe de teste
Postergado	O erro será tratado em versões futuras do programa, seja por baixa prioridade ou tempo escasso	Gerente de teste do cliente

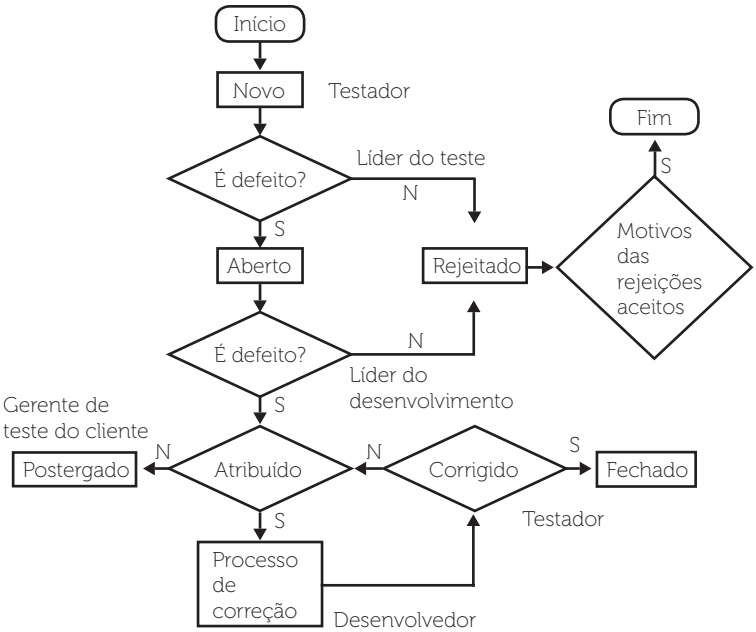
Fonte: <http://learndatamodeling.com/blog/what-is-bug-life-cycle-in-software-testing/>. Acesso em: 20 fev. 2016.

Exemplificando

Vamos ao exemplo de como o erro percorre seu caminho da descoberta até sua correção. Imagine um aplicativo de envio de *e-mail* sendo testado. Assim que o programa é terminado, o responsável confere a página de *login* e descobre que o campo de nome de usuário permite nomes duplicados. A partir da constatação, o responsável registra o erro e o reporta ao líder do teste. O líder então verifica o erro e, caso o valide, este receberá o *status* de "novo" e será passado à equipe de desenvolvimento. O líder da equipe de desenvolvimento faz nova verificação e, se for constatado que se trata de fato de um erro no programa, o erro é atribuído a um desenvolvedor – no caso um especialista em banco de dados – para correção. O desenvolvedor cria uma chave primária na tabela, altera o *status* do erro para "corrigido" e reporta a ação para o líder do teste que, por sua vez, muda o *status* para "novo teste" e atribui a tarefa de revisão a um testador. Se, depois disso, o erro tiver sido sanado, seu *status* passará para "Fechado".

Como o caminho entre a descoberta do erro e sua efetiva correção percorre um caminho definido, podemos representá-lo conforme ilustrado na Figura 4.3. Observe o fluxo:

Figura 4.3 | Fluxograma de ações para tratamento de um erro



Fonte: <http://learndatamodeling.com/blog/what-is-bug-life-cycle-in-software-testing/>. Acesso em: 21 fev. 2016.

Pois bem, este é um meio possível de se tratar um erro. Sejam quais forem as ações escolhidas, elas devem ser registradas, estruturadas e, naturalmente, desempenhadas por todos os envolvidos no processo de teste. Há, no entanto, algumas outras situações que podem ocorrer durante o tratamento de um erro. Vejamos:

- O testador reporta um erro ao líder do teste, que não interpreta a situação como erro e o rejeita.
- O líder do teste valida o erro encontrado pelo testador e o reporta ao líder do desenvolvimento, que o rejeita.
- Tanto o líder do teste quanto o líder do desenvolvimento validam o erro. O desenvolvedor trabalha para corrigi-lo e reporta ao líder do teste que ele está resolvido. Mais uma vez, o testador realiza o teste e verifica que ele não foi resolvido. O erro, então, reassume o status de "aberto".
- Com base em sua prioridade ou gravidade, o erro pode ser postergado. A classificação da gravidade pode ser baixa, média, alta ou crítica.

(Disponível em: <http://learndatamodeling.com/blog/what-is-bug-life-cycle-in-software-testing/>. Acesso em: 21 fev. 2016).

Assimile

A intenção do que chamamos de Engenharia de Testes é simples: oferecer entendimento do que é um erro e oferecer formas de encontrá-lo.

Por mais que os procedimentos de teste sejam tidos como universais, não há como não os vincular ao modelo de desenvolvimento adotado para a construção do programa que será testado. Os dois modelos que já estudamos – tradicional e ágil – apresentam particularidades no trato do processo de teste, tais como quais testes serão executados e quando. Vejamos em detalhes como os modelos Cascata e *Extreme Programming* (XP) conduzem os testes.

Modelo Cascata

Teoricamente, a fase de testes nesta metodologia concentra-se logo após o término da fase de codificação do programa. Na prática, contudo, o que se tem adotado é a revisão dos artefatos criados em cada uma das fases, sejam eles documentos, especificações, projetos ou um programa. Ao longo dos anos, a aplicação de verificações e validações (nas quais o teste está incluído) apenas no programa executável mostrou-se ineficiente, principalmente por deixar incorreções da

especificação dos requisitos propagarem-se pela fase de implementação (PINHEIRO, 2015).

Na sequência, abordaremos o já conhecido TDD, meio pelo qual o XP conduz seu processo de teste.

TDD - *Test-Driven Development* (Desenvolvimento Guiado pelos Testes)

Trata-se de um formato de teste muito parecido com o "codificar e testar", modelo de desenvolvimento no qual não se dá ênfase a outras etapas, senão as de codificar e testar.

Pesquise mais

O formato "codificar e testar" é também conhecido como metodologia codifica-corrige. Pesquise mais sobre ela em: <https://marcelocoelhao.wordpress.com/2010/01/06/metodologias-de-desenvolvimento-de-software/>. Acesso em: 22 fev. 2016.

Outra fonte interessante pode ser consultada em <http://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>. Acesso em: 26 fev. 2016.

Para a realização dos testes, o TDD apoia-se no fato de que a descrição dos requisitos – ou as estórias escritas pelo cliente – não constitui documento com excessiva formalidade e detalhamento de funções e que, por esse motivo, o cliente deve acompanhar o processo de codificação e teste. Em tempos passados, o teste era realizado tomando-se como base o código completo, ou quase completo, já que as linguagens de programação dificultavam a execução de apenas partes do programa. Hoje em dia, tecnologias que suportam linguagens orientadas a objeto (como o Java, por exemplo) permitem não só a automatização dos testes – ação tão importante no âmbito do TDD – como também a execução de partes autônomas de um programa, como uma classe, por exemplo.

O teste de parte do código segue-se à sua criação, ou seja, o teste e a integração são feitos logo após sua codificação.

Os passos do desenvolvimento guiado pelos testes incluem:

- A seleção de um conjunto de casos de teste.
- A execução do caso de teste. Encontrando-se o defeito, o código deverá ser ajustado. Caso não se encontre defeito, novo conjunto de casos de teste deve ser selecionado e o processo deve ser reiniciado (PINHEIRO, 2015).

Reflita

Por que a figura do testador é necessária no processo de testes? Por que os desenvolvedores, eles próprios, não assumem os testes? Reflita:

Os testes feitos por desenvolvedores tendem a verificar apenas trechos do código que, provavelmente, não conterão defeitos. Desenvolvedores só conseguem enxergar de 50% a 60% dos casos de teste. O testador, por sua vez, tem perfil diferente. São exploradores, gostam de encontrar problemas, são criativos nas execuções do software e possuem visão das diferentes situações em que o programa pode ser usado (PINHEIRO, 2015).

Este foi o conteúdo teórico preparado para esta aula. Na parte da aula reservada às questões práticas, trataremos da aplicação de um teste funcional.

Faça você mesmo

Sua tarefa é conceber e relatar uma sequência de tratamento do erro encontrado no código, seja por meio de fluxograma, seja por meio de descrição textual. O tratamento deve incluir as ações e seus responsáveis.

SEM MEDO DE ERRAR!

A afirmação de que a atividade de teste deve ser bem planejada e estruturada antes de ser executada, feita na aula anterior, ainda soa atual aqui. A seleção dos casos de teste foi o primeiro passo tomado pela X-Soft para disciplinar o processo de teste. Agora chegou o momento de efetivamente aplicar o teste funcional, com o cuidado de estabelecer critérios para o tratamento dos erros encontrados e responsáveis por cada etapa do processo.

Como desafio proposto, sua missão então é a de aplicar o teste funcional num dos produtos da empresa, obedecendo os critérios previamente definidos. Tome por base o cenário descrito na sequência para a resolução do caso. Como as circunstâncias não são as mesmas especificadas na seção anterior, assuma que os casos de teste apropriados para este teste já foram selecionados.

- A equipe designada para a tarefa conta com um testador, dois desenvolvedores e um líder de desenvolvimento.
- Existem três funções a serem testadas no programa: manutenção de dados do

contato, geração de relatório dos clientes mais rentáveis por período e envio automático de mensagem ao cliente aniversariante.

Uma solução possível para este desafio é a que segue:

Com base nos casos de teste criados, o testador deverá executar o programa e conferir as saídas que cada uma das funções fornece quando acionadas.

Caso um erro seja encontrado pelo testador, ele deverá ser verificado pelo líder do desenvolvimento. Se for constatado que, de fato, se trata de um erro, sua descrição será passada a um dos desenvolvedores, com indicação de em qual função ele foi encontrado.

O desenvolvedor procede a correção e repassa a informação de ajuste feito ao líder do desenvolvimento que, por sua vez, realiza nova verificação. Se, na visão do desenvolvedor e de seu líder, o erro não existir mais, nova informação de ajuste feito é emitida, desta vez ao testador, que faz a validação ou não da correção.

Durante o processo, os envolvidos deverão anotar o tempo utilizado em cada correção, em qual função o erro foi encontrado, as ocorrências de não validação do erro e as ocorrências de não validação do ajuste feito.

Caso o código tenha sido escrito na linguagem Java, este procedimento poderá ser desenvolvido na *JUnit*, ferramenta gratuita desenvolvida por Erich Gamma e Kent Beck para criação de testes automatizados.

Atenção!

Por causa das suas especificidades, o tempo empenhado nos testes e nas correções varia de função para função.

Lembre-se

Não se deve assumir que um erro foi corrigido sem que um novo teste seja feito.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.

Ajustando o processo de teste	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software.
2. Objetivos de aprendizagem	Conhecer os conceitos e de que forma se dá a execução do *Test-Driven Development* (TDD).
3. Conteúdos relacionados	Desenvolvimento dirigido a testes – *Test-Driven Development* (TDD): conceitos, processo e benefícios.
4. Descrição da SP	A RH-Soft, empresa desenvolvedora de sistemas feitos para departamentos de recursos humanos, adotou recentemente procedimento de teste em seus produtos. O processo de descoberta e correção do erro pode ser resumido assim: uma vez descoberto pelo desenvolvedor, o erro é informado a outro membro da equipe, que o corrige e relata a correção ao primeiro desenvolvedor. Mesmo com esse tratamento, os erros continuam se manifestando após a entrega feita ao cliente. Seu desafio aqui é indicar o procedimento adequado de teste, de modo a minimizar a continuidade do erro após a entrega.
5. Resolução da SP	A solução do caso requer as possíveis providências: um especialista em teste (ou testador) deve ser incorporado à equipe. As verificações feitas pelo desenvolvedor tendem a buscar funções e partes do sistema que apresentam menor chance de retornar um erro. Após a informação de que o erro foi corrigido, um outro profissional – possivelmente um segundo testador ou um líder do desenvolvimento – deve fazer nova verificação no programa ajustado. Com essas pequenas providências, o processo de teste pode evitar retrabalho.

Lembre-se

A figura do testador é indispensável no processo de teste. Programas conferidos por quem o desenvolveu tendem a continuar mantendo os erros após os testes.

Faça você mesmo

Faça a leitura e resuma o artigo encontrado em: <http://revistapensar.com.br/tecnologia/pasta_upload/artigos/a80.pdf>. Acesso em: 8 mar. 2016. Ele trata da importância de uma equipe de testes de software presente em uma organização, além de abordar o processo de teste.

Faça valer a pena

1. Em relação à Engenharia de Testes, assinale a alternativa que contém a afirmação verdadeira.

a) Ramo da Engenharia de Software que cuida da busca e formação de engenheiros de testes.

b) Área que cuida da criação de fluxogramas para processos de teste.

c) Área que cuida da busca e entendimento do erro em um software e das formas de eliminá-lo.

d) Trata-se de expressão sinônima de Engenharia de Software.

e) Área que tem como objetivo eliminar a necessidade de testes em um processo de desenvolvimento.

2. . Em relação às ocasiões em que um erro ocorre, analise as afirmações a seguir:

I. Um erro ocorre quando o usuário falha na operação do sistema.

II. Um erro ocorre quando o sistema executa uma operação não prevista nos requisitos.

III. Um erro ocorre quando o sistema não executa uma função prevista nos requisitos.

IV. Um erro ocorre quando o sistema executa uma função vetada nos requisitos.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) IV.

d) II.

e) II, III e IV.

3. Em relação ao ciclo de vida de um erro, analise as afirmações a seguir:

I. O ciclo de vida de um erro acompanha o ciclo de vida do sistema, de acordo com o modelo de desenvolvimento adotado.

II. Um erro rejeitado é aquele comportamento supostamente anômalo cujo teste mostra não ser um erro.

III. Um erro atribuído é aquele já corrigido.

IV. Um erro fechado é aquele sobre o qual não se pode aplicar métodos de correção, por não estar acessível ao testador.

É verdadeiro o que se afirma em:

a) II, apenas.

b) I, II, III e IV.

c) II e IV, apenas.

d) II e III, apenas.

e) I, apenas.

Seção 4.3

Os testes de *release* e de usuário

Diálogo aberto

Nesta etapa do nosso estudo, já concordamos que os testes são fundamentais no processo de desenvolvimento de um produto. Sem que as devidas verificações sejam feitas, não há a mínima segurança de que o sistema se comportará conforme o esperado.

A expressão "devidas verificações" demanda atenção especial de nossa parte: ela indica, entre outras coisas, a necessidade de aplicação do teste certo, pela pessoa certa e no tempo devido.

Antes de iniciarmos a abordagem teórica, é bom que nos lembremos que temos adquirido, ao longo do curso, conhecimento das principais metodologias de desenvolvimento de software, das normas de qualidade e, nesta unidade, dos processos de teste de software. Temos como objetivo específico, nesta aula, conhecer o que são e como implementar o teste de release e o teste de usuário.

Conforme constataremos na sequência, não há apenas um tipo de teste a ser feito no produto. As verificações devem focar desde o desempenho do programa até sua adequação à máquina em que será executado. Não se pode conceber também que apenas uma pessoa ou grupo realize os testes. Diferentes pontos de vista e interesses devem ser acomodados pelo processo, fato que também deu motivo para a criação de várias modalidades de teste.

Ainda resta uma questão a ser abordada: é válido e útil o registro dos resultados obtidos no teste? Podemos considerar esses registros base para criação de histórico do processo? Pois será nesta seção que abordaremos estes assuntos que circunstanciam o processo de teste e que serão argumento para o desafio desta aula.

Novamente se faz necessário resgatarmos o desafio que dá sentido ao nosso curso: fazer com que a X-Soft atinja bom nível de excelência em seus processos e produtos, tendo como ponto de partida uma organização sem processo definido e sem qualquer padronização de qualidade. Em específico, nossa missão nesta seção

é fazer o registro dos resultados obtidos com a aplicação do teste de software como meio eficiente de formar uma base para manutenção de histórico dos processos. Pelos próximos parágrafos você terá contato com a base teórica sobre modalidades de teste e sobre registros de atividades no processo.

Bons estudos!

Não pode faltar

É justamente por causa de sua elevada importância e criticidade que a atividade de teste conquistou grande abrangência no processo de desenvolvimento de um produto. Os testes há muito não estão mais restritos apenas às ações da equipe de desenvolvimento e as ocasiões em que são executados não se limitam simplesmente à pós-codificação.

Diversas modalidades de testes vêm sendo aplicadas por diferentes atores do processo, o que visa garantir que pessoas com diferentes visões do produto e em tempos distintos possam avalizá-lo.

Os testes escritos para o usuário final (ou para o cliente) serão abordados nos próximos parágrafos e seu conteúdo teórico dividirá espaço com outros conceitos relacionados à atividade e com boas práticas de análise dos resultados obtidos pela aplicação de um teste funcional, de modo a capacitá-lo para superar o desafio proposto nesta seção. Ao trabalho!

Testes de unidade

Para entender um teste de unidade, imagine o software como um conjunto de partes que, juntas, formam o todo. Essa modalidade de teste é direcionada a uma rotina, classe ou pequena parte de um produto e é normalmente executada pelo próprio desenvolvedor, de modo a assegurar que determinada unidade poderá ser integrada ao resto do software.

No contexto dos testes de unidade (ou testes unitários) insere-se um elemento conhecido como *stub*. Ele simula resultados que são retornados por um determinado componente do qual o software depende (PINHEIRO, 2015). Em outras palavras, um *stub* é um trecho de código que substituirá as entradas, dependências e comunicações que a unidade deveria receber em uma execução do programa. Quando um componente A que vai ser testado chama operações de outro componente *B* que ainda não foi implementado, pode-se criar uma implementação simplificada de *B*, chamada *stub*. Neste caso, devemos entender que o componente A é a unidade a ser testada.

Exemplificando

Veja um bom exemplo de aplicação de um *stub*: Suponha que o componente *A* é uma classe que deve usar um gerador de números primos *B*. A implementação de *B* ainda não foi feita. No entanto, para testar a unidade *A* não será necessária a geração de alguns poucos números primos. Assim, *B* pode ser substituído por uma função *stub* que gera apenas os 5 primeiros números primos (WAZLAWICK, 2013).

Há situações, no entanto, em que o módulo *B* já está implementado, mas o módulo *A* (que aqui representa nossa unidade) que chama as funções de *B* ainda não foi implementado. Deverá ser implementada, então, uma simulação do módulo *A*, que recebe o nome de driver. A diferença entre *stubs* e *drivers* concentra-se na relação de dependência entre os componentes.

As técnicas de teste funcional e estrutural, ambas abordadas anteriormente nesta seção, podem ser utilizadas para a execução de um teste de unidade. O teste funcional serve para validar a função específica que está sob teste. Quando utilizada, a técnica de teste estrutural (ou caixa-branca) viabiliza a validação dos fluxos de execução da unidade. Ambas as técnicas podem ser utilizadas de forma combinada ou isolada num teste de unidade.

Reflita

Os *stubs*, como sabemos, geram valores de entrada para unidades que serão testadas. Você entende como legítimo que um stub seja construído também para gerar valores errados para a classe, por exemplo?

Testes de integração

Trata-se de teste executado pelo testador para garantir que vários componentes do sistema funcionem corretamente juntos (PINHEIRO, 2015). Eles são feitos quando as unidades (as classes, por exemplo) já foram testadas de forma isolada e precisam ser integradas para gerar uma nova versão do *software*.

No caso de as classes a serem testadas precisarem de comunicação com outros componentes ou classes que ainda não foram testadas – ou mesmo implementadas –, os *stubs* mais uma vez serão necessários. O problema com um *stub* é que se investe tempo para desenvolver uma função que não será efetivamente entregue. Além disso, nem sempre é possível saber se a simulação produzida pelo *stub* será suficientemente adequada para os testes (WAZLAWICK, 2013).

Com o sistema integrado e testado, chega o momento de entregá-lo ao usuário

para que ele o teste também.

Teste de usuário (ou teste de aceitação)

Quando já se dispõe da interface final do sistema, o teste de aceitação já pode ser aplicado. Como o próprio nome nos faz supor, esse teste é executado pelo usuário e não pela equipe de testadores ou desenvolvedores. Para entendermos melhor essa modalidade de teste, é interessante que a coloquemos em oposição ao teste de sistema.

O **teste de sistema** é feito quando todas as unidades e as integrações entre elas já foram testadas. Pode ser que a equipe já se sinta confiante o bastante nesse ponto para assumir que seu produto é de boa qualidade. No entanto, é necessário que ele passe por esse novo teste depois de integrado. O objetivo da sua aplicação é o de verificar se o programa é capaz de executar processos completos, sempre adotando o ponto de vista do usuário.

Pois bem, o teste de usuário pode ser planejado tal qual o teste de sistema. A diferença é que ele será executado pelo usuário. Em outras palavras, enquanto o teste de sistema faz a *verificação* do sistema, o teste de aceitação faz sua *validação*.

Exemplificando

Um plano viável para a realização de teste de aceitação em um programa de vendas pela internet é o que segue no Quadro 4.3.

Quadro 4.3 | Exemplo de roteiro para teste de aceitação

Como testar	Resultado
Um cliente cadastrado informa livros válidos, indica um endereço e cartão de crédito válidos e a operadora (um *stub*, possivelmente) autoriza a compra.	Compra efetuada.
Um cliente cadastrado informa livros válidos e guarda o carrinho.	Carrinho guardado.
Um cliente não cadastrado informa livros válidos e guarda o carrinho.	O cliente é cadastrado e o carrinho é guardado.
Um cliente cadastrado informa livros válidos, indica um endereço inválido e depois um endereço válido; indica um cartão válido e a operadora autoriza a compra.	O endereço inválido é atualizado e a compra é efetuada.
Um cliente cadastrado informa livros válidos, indica um endereço e cartão válidos e a operadora não autoriza a compra. O cliente informa outro cartão válido e a operadora autoriza a compra.	Compra efetuada com o segundo cartão informado.

Fonte: adaptado de Wazlawick (2013, p. 296).

O teste de aceitação é chamado de **Teste Alfa** quando feito pelo cliente sem planejamento rígido ou formalidades. Se o teste é ainda menos controlado pelo time de desenvolvimento e as versões do programa são testadas por vários usuários, sem acompanhamento direto ou controle da desenvolvedora, então o procedimento é chamado de **Teste Beta**. Nesta modalidade, as versões disponibilizadas costumam expirar depois de certo tempo de uso.

Assimile

O teste de aceitação visa à validação dos requisitos implementados no software, não mais a verificação de defeitos (WAZLAWICK, 2013).

Um termo bastante comum no contexto dos testes de aceitação é o ***release***. Literalmente, o termo refere-se a uma liberação ou lançamento de uma nova versão de um produto de software. Os testes aplicados sobre um *release* seguem o padrão dos testes de usuário. Usualmente, um lançamento obedece às seguintes fases:

- **Alfa**: nesta fase, o lançamento ainda está em processo de testes, que são realizados por pessoas situadas normalmente fora do processo de desenvolvimento. Os produtos em fase de teste alfa podem ser instáveis e sujeitos a travamento.
- **Beta**: trata-se da classificação posterior a Alfa. Um lançamento em beta teste é avaliado por testadores beta, normalmente clientes em potencial que aceitam a tarefa de teste sem cobrar por isso. Uma versão Beta é utilizada para demonstrações dentro da organização e para clientes externos.
- ***Release Candidate***: trata-se de versão do sistema com potencial para ser a versão final. Neste caso, todas as funcionalidades já foram devidamente testadas por meio das fases Alfa e Beta.

Um *release* pode ser interno ou externo. O primeiro é usado somente pela equipe interna de desenvolvimento para fins de controle ou para demonstração a clientes e usuários. Um *release* externo é uma versão do produto distribuída para os usuários finais (BARTIÉ, 2002).

De uma forma ou de outra, as modalidades de teste apresentadas até aqui relacionam-se diretamente às funcionalidades do sistema. Em outras palavras, elas fazem a verificação dos processos que efetivamente devem ser realizados pelo programa. No entanto, a abrangência dos testes é maior e demanda a aplicação de outros tipos de verificações no produto. A estes tipos damos o nome de testes suplementares. Vejamos dois exemplos:

a) **Teste de Desempenho**: tipo que tem por objetivo determinar se, nas situações de

pico máximo de acesso e concorrência, o desempenho ainda permanece consistente com o que foi definido para essa característica do sistema. Assim, o critério de sucesso aqui é a obtenção de tempo de resposta compatível com o que se espera do produto, quando o sistema trabalha em seu limite. Um plano possível para esse teste está descrito a seguir (BARTIÉ, 2002):

- Validar todos os requisitos de desempenho identificados.
- Simular *n* usuários acessando a mesma informação, de forma simultânea.
- Simular *n* usuários processando a mesma transação, de forma simultânea.
- Simular *n%* de tráfego na rede.
- Combinar todos esses elementos.

Quando o procedimento eleva ao máximo a quantidade de dados e transações às quais o sistema é submetido, o teste realizado é chamado de **Teste de Estresse**. Ele é realizado para que se possa verificar se o sistema é suficientemente robusto em situações extremas de carga de trabalho.

b) **Teste de recuperação**: seu objetivo é avaliar o software após a ocorrência de uma falha ou de uma situação anormal de funcionamento. Algumas aplicações demandam alta disponibilidade do programa e, no caso de falha, o software deve ter a capacidade de se manter em execução até que a condição adversa desapareça.

Os testes de recuperação devem prever também os procedimentos de recuperação do estado inicial da transação interrompida, impedindo que determinados processamentos sejam realizados pela metade (BARTIÉ, 2002).

Normalmente, o teste de recuperação trata de situações referentes a (WAZLAWICK, 2013):

- Queda de energia na organização em que o sistema está em funcionamento.
- Discos corrompidos.
- Problemas de queda de comunicação.
- Quaisquer outras condições que possam provocar a terminação anormal do programa ou a interrupção temporária em seu funcionamento.

Pesquise mais

É possível que, ao se aplicar uma manutenção num programa, outros defeitos sejam gerados no código? Acertou se respondeu que sim.

> Pesquise sobre Teste de Regressão para saber mais. Comece por <http://gtsw.blogspot.com.br/2009/03/importancia-dos-testes-de-regressao.html> e <http://www.testavo.com.br/2010/05/desmistificando-testes-de-regressao.html>. Acesso em: 1 mar. 2016.

Pois bem, eis então algumas das outras modalidades de teste e outros conceitos relacionados à atividade. Chega a hora, então, de tratarmos dos registros das atividades de teste, visando capacitá-lo ainda mais para o desafio desta seção. Vamos a eles?

Os relatórios da qualidade do software

O registro das atividades relacionadas ao processo de qualidade, sobretudo os testes, são feitos em relatórios específicos. Além do efetivo registro, eles servem também como instrumentos de medição e análise. A execução do teste deve gerar o que chamamos de *log*, que nada mais é do que o registro das atividades desempenhadas. Vamos a alguns deles:

***Log* de execução**: este documento é criado para registrar todos os procedimentos realizados durante a execução de um ciclo de testes, bem como apontar as eventuais interrupções ocorridas. O *log* de execução certifica que, de fato, o teste foi realizado.

Ocorrências da validação: neste documento registram-se todas as ocorrências geradas durante um teste. Uma ocorrência pode ser, por exemplo, a identificação de um defeito. Neste caso, alguns dos dados a serem relatados serão: um nome ou número de identificação do defeito, a data em que foi encontrado e a sequência de ações capazes de reproduzi-lo (BARTIÉ, 2002).

Um dado também bastante importante a ser registrado é a classificação do defeito. Apesar de haver muitas classificações, alguns deles são mais relevantes:

a) **Falha na descrição funcional**: esta classificação refere-se à discrepância entre a descrição da funcionalidade e sua efetiva implementação. Por exemplo: a descrição da funcionalidade estabelece que o programa deveria promover acréscimo trimestral automático do salário dos gerentes de seção e o sistema aplica o aumento a cada quatro meses.

b) **Falha no controle, lógica ou sequenciamento do algoritmo**: um laço de repetição infinito é um exemplo desse tipo de falha.

c) **Falha nas estruturas de dados**: trata-se da definição incorreta, por exemplo, de matrizes e tabelas de banco de dados.

O relatório de teste, em resumo, serve para registrar a maior quantidade possível de dados acerca do processo.

Faça você mesmo

A norma IEEE 829 é o padrão criado pela IEEE para documentação do processo de teste. Um dos relatórios recomendados pelo padrão é o de incidente de teste, que prevê a descrição dos seguintes itens, entre outros, em caso de ocorrência de um incidente: entradas, resultados esperados e resultados encontrados. Vale registrar que um incidente pode ser entendido como discrepância entre o resultado esperado e o encontrado na execução dos casos de teste. Crie um exemplo de um processo de teste que contemple uma entrada de caso de teste, faça previsão de um resultado esperado para aquela entrada e forneça a descrição do resultado obtido.

SEM MEDO DE ERRAR!

A caminhada da X-Soft rumo à excelência não para! Depois de fazer a seleção dos casos de teste e de efetivamente aplicar teste funcional em um dos seus produtos, a X-Soft vê-se na necessidade de fazer um registro criterioso das atividades desempenhadas no processo de teste, com a finalidade de criar base de comparação para testes futuros. Afinal, como saber se a aplicação de um método foi bem-sucedida se não pela comparação entre dois ou mais resultados?

Vale a pena dizer que, na prática, os registros dos resultados obtidos no teste são feitos no ato da sua aplicação ou em situação inserida em seu contexto. Por exemplo, as anotações referentes à ocorrência de uma falha no algoritmo devem ser feitas assim que ela acontece, no momento do teste. Para efeito de organização didática, esta seção coloca o registro das ocorrências e resultados como etapa posterior à aplicação do teste, embora não o seja de fato.

Pois bem, o desafio lançado aqui é justamente o de registrar o processo e os resultados obtidos com a aplicação do teste de software em um dos seus produtos. O que deve ser registrado? O que não deve ser registrado? Há formato definido?

Uma solução possível para este desafio é a seguinte:

- A equipe deve criar, oficializar e publicar um *log* de execuções, ou seja, um formulário apropriado para que todos os registros de ocorrências sejam feitos.
- O formulário de registro deve conter, no mínimo, os seguintes campos:

Responsáveis pelo teste

Data e horário de início

Data e horário de término

Função, unidade ou sistema testados

Fase do teste: teste de unidade, teste de integração ou teste de sistema.

Falha na interpretação da função: nesta seção devem ser relatadas as divergências entre a função especificada nos requisitos e a função de fato implementada.

Falha na construção de estrutura de dados: erros na criação de tabelas e outras estruturas devem ser registrados aqui.

Defeito algorítmico: devem ser identificados erros de lógicas, laços infinitos, por exemplo.

Travamento de origem não identificada

Outras ocorrências relevantes

Este formulário pode – e deve – passar por constante aprimoramento em seus campos e em sua utilização.

Atenção!

O formato do *log* de execuções é livre e deve ser definido pela equipe, segundo critérios próprios.

Lembre-se

A norma IEEE 829 é o padrão criado pela IEEE para documentação do processo de teste.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.

Incluindo nova etapa no processo de teste	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software.
2. Objetivos de aprendizagem	Conhecer o que são e como implementar o teste de *release* e o teste de usuário.
3. Conteúdos relacionados	Testes de *release* e de usuário: conceitos, características e tipos.
4. Descrição da SP	A M-Soft, empresa desenvolvedora de sistemas para consultórios médicos, adota procedimento de teste que prevê a verificação isolada de cada função do sistema. Na ocorrência de erro, ele é registrado, atribuído e corrigido. Após sua integração, o programa é submetido ao cliente para teste de aceitação. Mesmo com os procedimentos seguidos como se deve, muitas ocorrências de erros continuam se manifestando no teste de aceitação, causando frustração no cliente e na equipe. Seu desafio aqui é indicar procedimento adequado de teste, com as etapas corretamente estipuladas, de modo a minimizar a continuidade dos erros nos testes de aceitação.
5. Resolução da SP	A solução para o caso requer que seja incluída etapa de **teste de integração** no processo, com o objetivo de verificar as comunicações e interfaces entre os módulos ou unidades do sistema. Sem a execução dessa etapa, não haverá confiança suficiente de que os dados de saída de uma função, por exemplo, estejam no formato e quantidade adequados para servirem como dados de entrada de outra função, subordinada àquela.

Lembre-se

No teste de integração, os módulos são combinados e testados conjuntamente. Trata-se de etapa indispensável no processo e que antecede o teste de sistema.

Faça você mesmo

Imagine que você crie um sistema, submeta-o a teste, certifique-se de que ele funciona corretamente e, tempos depois que o instala no servidor situado na organização do cliente, descobre que seu funcionamento não está correto por causa de questões relacionadas ao *hardware*. Para evitar essa situação, é costume proceder o **teste de instalação**. Pesquise em *sites* e livros especializados mais sobre o conceito e sobre o procedimento correto para esse tipo de teste.

Faça valer a pena

1. Em relação ao Teste de Unidade, assinale a alternativa que contém a afirmação verdadeira.

a) Modalidade de teste que toma o sistema todo como uma unidade e aplica sobre ele um teste funcional.

b) Atividade de teste que consiste em verificar se um componente individual do software (ou unidade) foi corretamente implementado.

c) Atividade de teste que consiste em verificar se várias unidades interligadas foram corretamente implementadas.

d) Tipo de teste em que o cliente valida uma única funcionalidade do sistema.

e) Tipo de teste em que o cliente valida várias unidades interligadas do sistema.

2. Em relação a um *stub*, analise as afirmações a seguir:

I. Trata-se de um tipo de defeito que ainda não foi revelado pelos testes.

II. Um *stub* é um trecho de código que substituirá as entradas, dependências e comunicações que a unidade deveria receber em uma execução do programa.

III. Um *stub* não será aproveitado como parte integrante do sistema que será entregue ao cliente.

IV. Um testador reconhece um stub pelo tipo de manifestação do defeito durante a execução do programa.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) IV.

d) II.

e) II, III e IV.

3. Em relação ao teste de aceitação, analise as afirmações a seguir:

I. O teste de aceitação é executado pela equipe de desenvolvimento e constitui a última tentativa da equipe de descobrir defeitos no código antes da entrega.

II. O teste de aceitação é executado pelo usuário final e visa à validação dos requisitos previamente estipulados.

III. Executado o teste de aceitação, o cliente não poderá ainda utilizar o produto, já que ele deverá passar pelo teste de sistema antes da entrega.

IV. O teste de aceitação é executado pelo usuário final e visa à verificação de falhas ainda não manifestadas no programa.

É verdadeiro o que se afirma em:

a) II, apenas.

b) I, II, III e IV.

c) II e IV, apenas.

d) II e III, apenas.

e) I, apenas.

Seção 4.4

A manutenção e evolução de Software

Diálogo aberto

Aqui estamos, em nossa última aula.

No trajeto que se iniciou pelo desenvolvimento de produtos com base apenas no talento individual dos seus desenvolvedores até o atingimento da maturidade processual, a X-Soft percorreu um longo caminho.

Assim que constatou a necessidade de organizar seus processos, a empresa adotou e consolidou metodologia consagrada de desenvolvimento de software e, no momento seguinte, enxergou a necessidade de evoluir para um modelo ágil e moderno de criação de programas. Ato contínuo, passou a adotar medidas e padrões de qualidade com fins de atingir a excelência em seus produtos. Por fim, como parte do esforço para aprimoramento da qualidade, implantou procedimento formal de testes. Você, claro, foi figura fundamental nessa transformação.

Este é o retrato da X-Soft atual: uma empresa sólida, reconhecida em seu meio, amadurecida em seus processos e preocupada com sua constante evolução. E é justamente de evolução que trata esta última seção. Como meio de completar seu ciclo virtuoso de desenvolvimento, a organização pretende adotar meios eficientes de manutenção dos seus produtos e implantar modelo para estimar o esforço que esta atividade acarretará à equipe.

Em resumo, o objetivo desta seção é abordar a manutenção e a evolução do software e o desafio que se propõe é justamente implantar um modelo de estimação de esforço de manutenção num dos produtos da X-Soft. Com essa medida, a manutenção deixará de ser encarada como um retrabalho e passará a ser tida como meio de proporcionar evolução ao produto.

Por fim, nesta última aula, vale a pena resgatarmos as competências que motivaram nosso estudo até aqui. Você está lembrado? Pois bem, esperamos que o conhecimento das principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software tenha sido adquirido.

Parabéns pelo bom trabalho feito até aqui e vamos em frente!

Não pode faltar

A expressão manutenção de software, quando entendida da forma costumeira, remete apenas a correções no programa provocadas pela ocorrência de erros durante a operação feita pelo usuário final. No entanto, essa atividade pode e deve assumir caráter mais significativo do que o meramente corretivo. Ao mudar sua concepção sobre a atividade, a equipe poderá colocar-se diante da oportunidade de fazer o produto evoluir sem fazê-lo perder suas características principais, o que certamente terá boa consequência na utilidade do produto e na satisfação do cliente.

Quando estudamos o modelo tradicional de desenvolvimento, o tema foi abordado principalmente de forma conceitual e circunstanciado como parte final do ciclo de criação de um programa. Nesta seção, contudo, você terá embasamento para perceber a manutenção de forma diferente e poderá ter contato com um meio de dimensionar o esforço que deverá ser exigido para a conclusão de uma atividade de manutenção. Ao trabalho!

A manutenção pós-entrega

Assim que o produto passa pelo teste de aceitação, ele está apto a ser entregue ao cliente. Embora longe de ser considerada o encerramento definitivo do ciclo do software, a entrega marca o término do desenvolvimento de uma versão apta a ser utilizada em meio produtivo e adequada ao propósito de resolver as situações colocadas pelo cliente como requisitos do produto.

Este marco no processo caracteriza-se também por ser o ponto de início de outra etapa igualmente desafiadora: a manutenção pós-entrega.

Hoje em dia, a expressão "manutenção de software" vem sendo substituída ou usada em conjunto com "evolução de software". Evolução talvez seja o termo mais adequado, já que as atividades de modificação de um produto que já está em operação não visam mantê-lo em seu estágio atual, mas fazê-lo evoluir de forma a adaptar-se a novos requisitos ou ainda corrigir defeitos (WAZLAWICK, 2013). Usaremos aqui as duas expressões como sinônimas.

Reflita

É correto imaginar que, após seu desenvolvimento, um software terá seu valor diminuído com o passar do tempo? Se imaginou que sim, está correto. Conforme falhas são descobertas no programa, conforme seus requisitos originais mudam e conforme produtos menos complexos, mais eficientes e mais avançados são lançados, o valor (não necessariamente

financeiro) do software original vai se perdendo. Daí a necessidade de se aplicar melhorias que façam o produto evoluir.

Conforme estudamos na primeira unidade, existem basicamente três razões que justificam o investimento na aplicação de modificações em um produto. Uma falha de compreensão de um requisito, de projeto, codificação, de documentação ou de qualquer outro tipo deve ser corrigida. Tal ação configura uma **manutenção corretiva**. Quando o código ou outro artefato é modificado para que a eficiência do produto aumente, temos uma **manutenção de aperfeiçoamento**, ou perfectiva. Se o programa deve se adaptar a uma mudança no ambiente em que ele opera (uma mudança na quantidade de dígitos das linhas telefônicas, por exemplo), então ocorre uma **manutenção adaptativa** (SCHACH, 2008).

É saudável que toda iniciativa de manutenção passe pela avaliação de viabilidade em se aplicar modificações em um programa ou se construir um novo produto. Fatores como dificuldades técnicas na manutenção, inadequação do produto original às necessidades do cliente e custo devem ser levados em conta antes da decisão.

Exemplificando

A diferença entre a necessidade de novo desenvolvimento e a necessidade de aplicação da manutenção em produto já existente pode parecer clara, mas vale a pena contrastarmos as duas situações, considerando outros fatores que não o aspecto técnico apenas. Leia com atenção o que segue.

Imagine que uma mulher teve seu retrato pintado aos 18 anos. Anos mais tarde, já casada, ela decide que o marido deve ser incluído no quadro, ao seu lado. Grandes dificuldades acompanham essa decisão:

- A tela não é grande o suficiente para acomodar o marido ao lado dela.
- Com o tempo, as cores do quadro ficaram menos vivas. Haveria possibilidade de retoque, mas a tinta usada já não é mais fabricada.
- O artista original se aposentou e foi morar em outro país.
- O rosto da mulher envelheceu e não há garantia de que a figura do quadro possa ficar parecida com a feição atual da mulher.

Considere agora a manutenção de um programa de computador que custa R$ 2 milhões para ser desenvolvido. Quatro dificuldades se apresentam:

- O disco rígido no qual a base de dados está armazenada está quase cheio e novos dados não podem mais ser incluídos.

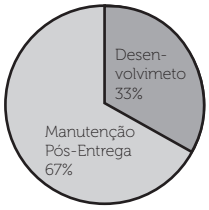
- A empresa que fabricava o disco faliu e outro disco de maior capacidade, de outro fabricante, já foi comprado. No entanto, há incompatibilidade entre o novo disco e o software e as adequações custarão R$ 100 mil.
- Os desenvolvedores originais já se aposentaram e a manutenção deverá ser feita por profissionais que jamais cuidaram do programa antes.
- O produto original foi construído por meio do uso do modelo clássico de desenvolvimento. Não entanto, há real necessidade de que o paradigma de orientação a objeto seja usado no produto que surgirá depois da manutenção.

Percebe a correspondência entre os itens do quadro e os itens do programa? A conclusão para o primeiro caso é que, partindo da estaca zero, um outro artista deverá pintar o retrato do casal para atender ao pedido da mulher. Isso também significa que, ao invés da manutenção de R$ 100 mil, um novo produto de R$ 2 milhões deve ser criado? Independentemente de estarmos tratando de quadros ou de programas de computador, muitas vezes parecerá mais simples e viável a criação de um novo produto. No entanto, considerações financeiras podem tornar a manutenção muito mais imediata do que um novo desenvolvimento (SCHACH, 2008).

Por mais improvável que possa parecer, a manutenção pós-entrega consome mais tempo e recurso do que qualquer outra atividade do ciclo do produto. Atualmente, aproximadamente dois terços do custo total de um programa estão relacionados à manutenção (SCHACH, 2008). As Figuras 4.4a e 4.4b mostram a relação de custos entre atividades de desenvolvimento e manutenção em dois momentos passados.

Figura 4.4a | Relação entre custo e manutenção entre 1976 e 1981

Dados obtidos entre 1976 e 1981

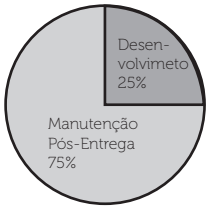


Desenvolvimento
Manutenção Pós-Entrega

Fonte: Schach, 2008.

Figura 4.4b | Relação entre custo e manutenção entre 1992 e 1998

Dados obtidos entre 1992 e 1998



Desenvolvimento
Manutenção Pós-Entrega

Fonte: Schach, 2008.

Fica claro que o investimento em técnicas, ferramentas e boas práticas que levem à redução desse custo se justifica plenamente. Uma dessas boas práticas é a incorporação da facilidade de manutenção do produto desde o início do seu desenvolvimento, por meio da correta modularização do produto e das atividades de verificação e validação desempenhadas durante todo o processo.

A correção de um erro apontado pelo usuário final demanda conhecimento não só do código-fonte, mas de todos os demais aspectos relacionados à sua produção, tais como as especificações dos requisitos, o projeto e a documentação relacionada.

Assimile

Já que o produto de software é formado por outros elementos além do código-fonte, qualquer alteração nos manuais feita após a entrega pode ser considerada manutenção.

Na Seção 4.2 estudamos tratamentos possíveis para erros no programa descobertos durante seu processo de desenvolvimento. Embora estejamos tratando agora de um programa já entregue, não haveria motivo relevante o suficiente para nos fazer mudar o fluxo para correção do erro neste atual contexto.

Como qualquer atividade desenvolvida durante a criação de um produto de *software*, a manutenção também requer processo formal e gerenciamento. Um elemento bastante importante neste gerenciamento é o **relatório de defeitos**. Via de regra, ele é preenchido pelo usuário final e contém informações suficientes para permitir que o programador de manutenção recrie o problema descoberto pelo cliente. Confirmada a existência do erro, o programador poderá classificá-lo como crítico, importante, normal, secundário e trivial, conforme sua relevância (SCHACH, 2008).

Exemplificando

Se um programa usado para folha de pagamento apresenta erro um dia antes da data do cálculo dos salários dos funcionários, então estaremos diante de um erro crítico e a intervenção da equipe deverá ser imediata.

Há, no entanto, a possibilidade de o usuário final ter interpretado mal um item do manual do produto e, por isso, entendido que determinada situação relacionada a ele se tratava de um erro. Neste caso, ao invés de correção, cabe apenas esclarecimento.

Suponhamos que o programador de manutenção tenha constatado a anomalia no programa. Ele deverá iniciar, então, a correção, sempre tomando precaução para que o processo não acabe introduzindo novos erros.

Assimile

A inclusão ou alteração de uma função ou módulo no sistema requer a execução de **testes de regressão**, que visam evitar erros de integração com os módulos originais do sistema.

Feita a correção, o programador (ou outro membro da equipe) deverá realizar novo processo de teste, objetivando assegurar que o problema original foi sanado e nenhum outro foi criado em decorrência do ajuste.

São claras, portanto, as semelhanças entre o processo de correção de um erro antes da entrega do programa e o processo de correção desenvolvido após a entrega. No primeiro caso, entretanto, o testador é o responsável principal pela descoberta do erro. No segundo, o próprio usuário final o encontrará.

Lembre-se

Já que o produto de software é formado por outros elementos além do código-fonte, qualquer alteração nos manuais feita após a entrega pode ser considerada manutenção.

Pois bem, parece-nos claro agora que a manutenção é atividade fundamental no ciclo de vida do software e que, dada a sua contribuição para o aprimoramento constante do produto, ela justifica plenamente ser chamada de evolução do software. Em que pese sua importância no processo, há que se pensar no esforço necessário para empreender tal atividade. Afinal, como você pode imaginar, investimento em recursos humanos e financeiros sempre serão necessários.

Na sequência, será abordado um modelo de estimação de esforço de manutenção. Através da aplicação dessa técnica, você poderá quantificar qual o trabalho previsto para a atividade, com base na porcentagem de linhas de código que sofrerão alteração.

ACT - Modelo de Estimação de Esforço de Manutenção

O modelo ACT (*Annual Change Traffic* ou Tráfego Anual de Mudança) foi proposto por Boehm (1981) e se baseia em uma estimativa de porcentagem de linhas de código que passarão por manutenção. Para efeito de contagem, são consideradas como *linhas em manutenção* tanto as linhas a serem alteradas quanto as novas linhas criadas. O valor da variável ACT reflete o número de linhas que sofrem manutenção dividido pelo número total de linhas do código em um ano típico (WAZLAWICK, 2013). A fórmula criada é $E = ACT * SDT$, em que E representa o esforço, medido em desenvolvedor/mês, a ser aplicado no período de um ano nas atividades de manutenção, *ACT*

representa a porcentagem esperada de linhas modificadas ou adicionadas durante um ano em relação ao tamanho do software e *SDT* é o tempo de desenvolvimento do software, ou *Software Development Time*.

Pesquise mais

Entre os anos de 1974 e 1996, o pesquisador alemão Meir M. Lehman publicou oito leis que colocam a evolução do software como necessária e inevitável. Você pode pesquisar sobre o tema em:

<http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/evolucao-leis-lehman_v01.pdf>. Acesso em: 10 mar. 2016.

<http://www.rafaeldiasribeiro.com.br/downloads/testesw5.pdf>. Acesso em: 24 mar. 2016.

WAZLAWICK, Raul Sidnei. **Engenharia de Software**: Conceitos e Práticas. Rio de Janeiro: Elsiever, 2013, páginas 318 e seguintes.

Como exemplo, se tomarmos um programa que foi desenvolvido com o esforço de 80 horas de desenvolvimento/mês, seu *SDT* será igual a 80. Se a taxa anual esperada de linhas em manutenção (*ACT*) for de 2%, então o esforço anual esperado de manutenção para este programa será dado por $E = 0{,}02 * 80$; $E = 1{,}6$.

De acordo com essa técnica, espera-se esforço anual de 1,6 horas de desenvolvimento/mês destacadas para atividade de manutenção (WAZLAWICK, 2013).

É claro que você deve considerar a criticidade da manutenção, a experiência dos programadores, a complexidade do programa e outros fatores que poderão influenciar na necessidade de mão de obra. Contudo, não se pode negar que essa estimativa irá te ajudar a ter boa noção da quantidade de programadores que deverá destacar para a atividade.

Pois bem, chegamos assim ao fim do nosso conteúdo teórico. Torcemos vivamente para que a compreensão dos assuntos abordados em nossos 16 encontros tenha sido satisfatória e que o conhecimento adquirido seja decisivo na realização do seu projeto de vida. Fica a seguinte sugestão: não pare por aqui! A Engenharia de Software é um campo vastíssimo e tópicos mais avançados desta disciplina devem merecer sua atenção.

Faça você mesmo

O modelo de estimativa de esforço de manutenção chamado CII ou Cocomo II toma como entrada o número de linhas de código a serem

desenvolvidas ou Pontos por Função convertidos milhares de linha de código-fonte (KSLOC). Descubra a fórmula utilizada neste modelo e descreva cada um dos seus itens.

Comece por <http://fattocs.com/pt/cursos/nossos-cursos/cocomoii.html>. Acesso em: 24 mar. 2016.

SEM MEDO DE ERRAR!

Não nos enganemos: a manutenção é uma daquelas atividades que apresentam grandes desafios em sua execução e que não conferem grande reconhecimento a quem as pratica. Não é incomum que programadores sem muita experiência ou sem grande capacidade técnica sejam destacados para cuidar dos ajustes em um programa. Isso, contudo, está longe de diminuir sua importância.

É por meio da manutenção que o programa se torna mais adequado à sua função, mais confiável, melhora seu desempenho e ganha inovações interessantes. Claro que tudo isso tem seu custo, tanto material, quanto relacionado à mão de obra. Sob essa perspectiva, é natural que uma organização queira fazer estimativas de quanto esforço deverá ser empreendido na atividade e, com elas, dimensionar seu investimento.

Depois de adotar as medidas de qualidade necessárias para o atingimento de nível satisfatório de excelência, a atenção da X-Soft volta-se agora justamente para um meio de estimar o esforço que empreenderá em suas manutenções. A métrica a ser adotada é de aplicação bem simples e a apuração de seu resultado servirá para parametrizar as decisões relacionadas à atividade.

A maneira que se propõe para a resolução do desafio é igualmente simples. Imaginemos um programa relativamente complexo da X-Soft que tenha sido desenvolvido com o esforço de 100 horas de desenvolvimento no mês. Por causa da complexidade das suas funções, a taxa esperada de linhas que passarão por manutenção é de 6%.

Aplicando esses valores na fórmula $E = ACT * SDT$, teremos que:

$E = 0{,}06 * 100$

Espera-se, então, esforço anual de 6 horas de desenvolvimento por mês, no intervalo de 12 meses.

Atenção!

Existem outras técnicas importantes de estimativa de esforço de manutenção. Uma delas leva o nome de CII.

Lembre-se

Conforme abordado na Unidade 3, a métrica de Pontos por Função não se aplica apenas a programas em fase de planejamento. Ela poderá ser aplicada também a processos de manutenção, permitindo a estimativa do esforço necessário para se fazer certa alteração no programa.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.

Estimando o esforço de manutenção por Pontos por Função	
1. Competência geral	Competência para conhecer e conduzir processos de qualidade de software.
2. Objetivos de aprendizagem	Conhecer e saber fazer a manutenção e compreender como isso se aplica em meio à evolução do software.
3. Conteúdos relacionados	Necessidade de manutenção e Evolução de Software. Classificação das atividades de manutenção. Processo, ferramentas e tipos de atividades para manutenção de software.
4. Descrição da SP	Mais uma vez colocamos a M-Soft, desenvolvedora de programas para consultório médico, à frente da situação a ser tratada. A empresa vem adotando com relativo sucesso a técnica ACT para estimar os esforços de manutenção. No entanto, esse modelo de estimativa tem falhado em sistemas que demandam adição de novas funcionalidades, o que levou a estimativas incorretas e consequente necessidade de deslocamento de outros programadores de suas funções originais para a manutenção. Sua missão é propor a adoção da métrica de Pontos por Função para que a equipe possa ter a correta noção do investimento das atividades de manutenção.
5. Resolução da SP	A métrica de Pontos por Função pode ser eficiente para estimativa de inclusão de novas funcionalidades em sistemas já existentes. Vale lembrarmos que a criação de novas funções também é classificada como manutenção. Aqui, o processo de cálculo passa pelas mesmas etapas da aplicação da métrica em sistemas ainda não construídos. Assim, a estimativa de esforço de manutenção pode ser obtida pela sequência: Identificação da fronteira do aplicativo -> contagem das funções do tipo dados e do tipo transações -> obtenção dos pontos não ajustados -> determinação do fator de ajuste - > obtenção dos pontos ajustados.

	A fórmula para o cálculo dos pontos por função ficará como segue: $$VAF = 0{,}65 + \left(0{,}01 * \sum_{i=1}^{14} GSCi\right)$$

Lembre-se

A técnica de Pontos por Função pode ser aplicada em processos de manutenção, pois permite estimar o esforço para implantação de nova funcionalidade no sistema.

Faça você mesmo

Há situações em que ajustar partes do código simplesmente não basta para que a manutenção em um programa fique completa. Sistemas antigos, sem a devida documentação, poderão requerer o que chamamos de reengenharia. Pesquise sobre o tema e resuma-o em uma página. Você pode começar sua pesquisa por <http://www.inf.ufpr.br/silvia/ES/reengenharia/reengenharia.pdf>. Acesso em: 11 mar. 2016.

Faça valer a pena

1. Assinale a alternativa que descreve o conceito de manutenção pós-entrega.

a) Processo de revisão pelo qual passam as especificações de requisitos após terem sido aprovadas pela equipe de desenvolvimento.

b) Atividade que inclui melhorias, adequações e correções em um programa feitas após ele ter sido entregue ao testador.

c) Processo de inclusão de funções logo após a entrega do programa ao líder do desenvolvimento ter sido feita.

d) Atividade que inclui melhorias, adequações e correções em um programa feitas após ele ter sido entregue ao usuário final.

e) Processo de troca do local de instalação do programa.

2. Assinale a alternativa que contém apenas situações determinantes para que um gestor tome a decisão de desenvolver um novo sistema ao invés de aplicar manutenção no já existente:

a) Alto custo de desenvolvimento integral do programa, alta disponibilidade de desenvolvedores.

b) Programa atual obsoleto, necessidade de inúmeras alterações e acréscimos de funções.

c) Programa atual construído a partir de plataforma moderna, alta disponibilidade de desenvolvedores.

d) Falta de espaço no disco rígido para a inclusão de novos dados, programa atual construído com base no paradigma de Orientação a Objeto.

e) Baixa disponibilidade de testadores, falta de espaço no disco rígido para a inclusão de novos dados.

3. Em relação à origem dos problemas que levam à necessidade de manutenção, analise as afirmações a seguir:

I. Apenas travamentos do programa durante a execução justificam a aplicação de manutenção.

II. Não só o código-fonte, mas incorreções nas especificações dos requisitos, no projeto e na documentação podem ensejar manutenção.

III. Aplica-se manutenção, inclusive, quando um requisito não foi corretamente traduzido para uma função executável.

IV. A necessidade de manutenção nasce de uma manifestação do testador.

É verdadeiro o que se afirma apenas em:

a) IV.

b) II e IV.

c) III e IV.

d) II e III.

e) I.

Referências

BARTIÉ, Alexandre. **Garantia da qualidade de software**: as melhores práticas de engenharia de software aplicadas à sua empresa. 5. ed. São Paulo: Elsiever, 2002.

DELAMARO, M. E. **Introdução ao teste de software**. Rio de Janeiro: Elsevier, 2004.

MECENAS, Ivan; OLIVEIRA, Viviane. **Qualidade em *software***: uma metodologia para homologação de sistemas. [s. l.]: Alta Books, 2005.

MOORTHY, Vishnuvarthanan. ***Jumpstart to software quality Assurance***. [s.l : s.n.], 2013.

PINHEIRO, V. Um comparativo na execução de testes manuais e testes de aceitação automatizados em uma aplicação web. Simpósio Brasileiro de Qualidade de Software – SBQS 2015. **Anais**... Manaus: Uninorte, 2015.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Pearson Prentice Hall, 2009.

ROCHA, Ana Regina; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de software**: teoria e prática. São Paulo: Pearson, 2001.

SCHACH, Stephen. **Engenharia de software**: os paradigmas clássico e orientado a objetos. 7. ed. São Paulo: McGraw-Hill, 2008.

SHAFFER, Steven C. ***A brief introduction to software development and quality assurance management***. [s.l : s.n.], 2013.

WAZLAWICK, Raul Sidnei. **Engenharia de software**: conceitos e práticas. Rio de Janeiro: Elsiever, 2013.