6 Testes de ordem superior

Quando você termina de testar o módulo de um programa, você tem apenas iniciou o processo de teste. Isso é especialmente verdadeiro para programas grandes ou complexos. Considere este importante conceito:

> Um erro de software ocorre quando o programa não faz o que o usuário final espera razoavelmente que ele faça.

Aplicando esta definição, mesmo se você pudesse realizar um teste de módulo absolutamente perfeito, você ainda não poderia garantir que encontrou todos os erros de software. Para completar o teste, então, alguma forma de teste adicional é necessária.
Chamamos essa nova forma de teste de ordem superior.

O desenvolvimento de software é, em grande parte, um processo de comunicação de informações sobre o programa eventual e de tradução dessas informações de uma forma para outra. Em essência, está se movendo do conceitual para o concreto. Por esse motivo, a grande maioria dos erros de software pode ser atribuída a falhas, erros e “ruídos” durante a comunicação e tradução de informações.

Essa visão do desenvolvimento de software é ilustrada na Figura 6.1, um modelo do ciclo de desenvolvimento de um produto de software. O fluxo do processo pode ser resumido em sete etapas:

1. Traduzir as necessidades do usuário do programa em um conjunto de requisitos escritos. Esses são os objetivos do produto.

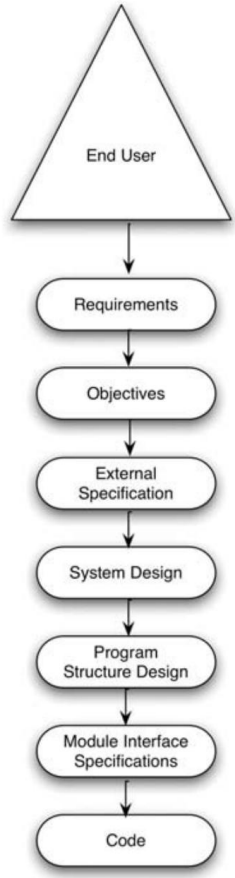

FIGURA 6.1 O Processo de Desenvolvimento de Software.

2. Traduzir os requisitos em objetivos específicos avaliando viabilidade, tempo e custo, resolvendo requisitos conflitantes e estabelecendo prioridades e compensações.

3. Traduzir os objetivos em uma especificação precisa do produto, visualizando o produto como uma caixa preta e considerando apenas suas interfaces e interações com o usuário final. Essa descrição é chamada de externa especificação.
4. Se o produto for um sistema como um sistema operacional, controle de voo sistema, sistema de banco de dados ou sistema de gerenciamento de pessoal de funcionários, em vez de um aplicativo (por exemplo, compilador, programa de folha de pagamento, processador de texto), o próximo processo é o projeto do sistema. Este passo

particiona o sistema em programas, componentes ou subsistemas individuais e define suas interfaces.

5. Projete a estrutura do programa ou programas especificando a função de cada módulo, a estrutura hierárquica dos módulos e as interfaces entre os módulos.

6. Desenvolva uma especificação precisa que defina a interface e a função de cada módulo.
7. Traduzir, por meio de uma ou mais subetapas, a especificação da interface do módulo para o algoritmo do código-fonte de cada módulo.

Aqui está outra maneira de ver essas formas de documentação:

Os requisitos especificam por que o programa é necessário.
Os objetivos especificam o que o programa deve fazer e quão bem o programa deve fazê-lo.
Especificações externas definem a representação exata do programa aos usuários.

A documentação associada aos processos subsequentes especifica, em níveis crescentes de detalhes, como o programa é construído.

Dada a premissa de que as sete etapas do ciclo de desenvolvimento envolvem comunicação, compreensão e tradução de informações, e a premissa de que a maioria dos erros de software decorre de falhas no tratamento da informação, existem três abordagens complementares para prevenir e/ou detectar esses erros.

Primeiro, podemos introduzir mais precisão no processo de desenvolvimento para evitar muitos dos erros. Em segundo lugar, podemos introduzir, no final de cada processo, uma etapa de verificação separada para localizar o maior número possível de erros antes de prosseguir para o próximo processo. Essa abordagem é ilustrada na Figura 6.2. Por exemplo, a especificação externa é verificada comparando-a com a saída do estágio anterior (a declaração de objetivos) e retornando quaisquer erros descobertos ao processo de especificação externa. (Use os métodos de inspeção de código e passo a passo discutidos no Capítulo 3 na etapa de verificação no final do sétimo processo.)

A terceira abordagem é orientar processos de teste distintos para processos de desenvolvimento distintos. Ou seja, concentre cada processo de teste em uma etapa de tradução específica - portanto, em uma classe específica de erros. Essa abordagem é ilustrada na Figura 6.3.

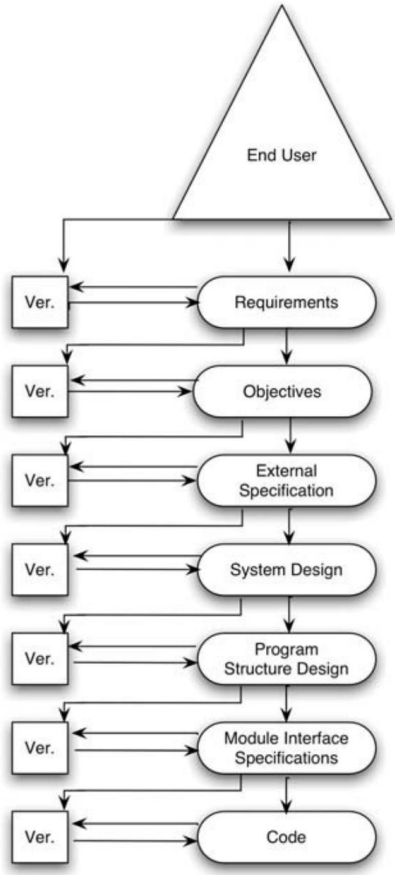

FIGURA 6.2 O Processo de Desenvolvimento com Verificação Intermediária Passos.

O ciclo de teste é estruturado para modelar o ciclo de desenvolvimento. Em outros Em outras palavras, você deve ser capaz de estabelecer uma correspondência direta entre os processos de desenvolvimento e teste. Por exemplo:

A finalidade de um teste de módulo é encontrar discrepâncias entre os módulos do programa e suas especificações de interface.

O propósito de um teste de função é mostrar que um programa não corresponder às suas especificações externas.

O objetivo de um teste de sistema é mostrar que o produto está de acordo com seus objetivos originais.

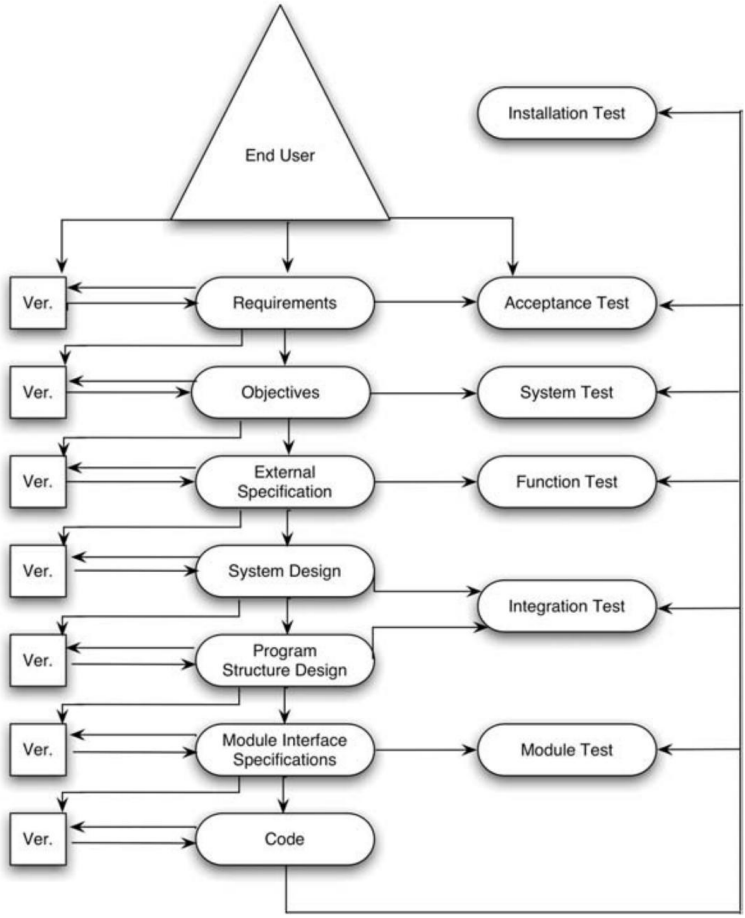

FIGURA 6.3 A correspondência entre desenvolvimento e teste Processos.

Observe como estruturamos estas declarações: "encontra discrepâncias", "não corresponde", "é inconsistente". Lembre-se de que o objetivo do teste de software é encontrar problemas (porque sabemos que haverá problemas!). Se você se propõe a provar que algum tipo de entrada funciona corretamente, ou assume que o programa é fiel às suas especificações e objetivos, seu teste ficará incompleto. Somente ao tentar provar que algum tipo de entrada funciona de forma inadequada, e assumir que o programa não é fiel às suas especificações e objetivos, seu teste estará completo. Este é um conceito importante que repetimos ao longo deste livro.

As vantagens dessa estrutura são que ela evita testes redundantes improdutivos e evita que você ignore turmas grandes de erros. Por exemplo, em vez de simplesmente rotular os testes do sistema como "o teste de todo o sistema" e possivelmente repetir testes anteriores, o teste do sistema é orientado para uma classe distinta de erros (aqueles cometidos durante a tradução dos objetivos para a especificação externa) e medido em relação a um tipo distinto de documentação no processo de desenvolvimento.

Os métodos de teste de ordem superior mostrados na Figura 6.3 são mais aplicáveis a produtos de software (programas escritos como resultado de um contrato ou destinados a uso amplo, em oposição a programas experimentais ou escrito para uso apenas pelo autor do programa). Programas não escritos como os produtos muitas vezes não têm requisitos e objetivos formais; por tal programas, o teste de função pode ser o único teste de ordem superior. Também o a necessidade de testes de ordem superior aumenta junto com o tamanho do programa. A razão é que a proporção de erros de projeto (erros cometidos no processos de desenvolvimento) para erros de codificação é consideravelmente maior em grandes programas do que em pequenos programas.

Observe que a sequência de processos de teste na Figura 6.3 não implica necessariamente uma sequência de tempo. Por exemplo, como o teste do sistema não é definido como "o tipo de teste que você faz após o teste de função", mas sim como um tipo distinto de teste focado em uma classe distinta de erros, pode muito bem ser parcialmente sobreposto no tempo com outros processos de teste.

Neste capítulo, discutimos os processos de teste de função, sistema, aceitação e instalação. Omitimos o teste de integração porque muitas vezes ele não é considerado uma etapa de teste separada; e, quando o módulo incremental teste é usado, é uma parte implícita do teste do módulo.

Manteremos as discussões desses processos de teste breves, gerais, e, na maioria das vezes, sem exemplos porque técnicas específicas utilizadas nesses testes de ordem superior são altamente dependentes do programa específico sendo testado. Por exemplo, as características de um teste de sistema (os tipos de casos de teste, a maneira como os casos de teste são projetados, as ferramentas de teste usadas) para um sistema operacional diferirá consideravelmente de um teste de sistema de um compilador, um programa que controla um reator nuclear ou um programa de aplicativo de banco de dados.

Nas últimas seções deste capítulo, abordamos questões organizacionais e de planejamento, juntamente com a importante questão de determinar quando parar de testar.

Teste de função

Conforme indicado na Figura 6.3, o teste de função é um processo de tentativa de encontrar discrepâncias entre o programa e a especificação externa. Uma especificação externa é uma descrição precisa do comportamento do programa do ponto de vista do usuário final.

Exceto quando usado em programas pequenos, o teste de função normalmente é uma atividade de caixa preta. Ou seja, você confia no processo de teste de módulo anterior para atingir os critérios de cobertura lógica de caixa branca desejados.

Para executar um teste de função, você analisa a especificação para derivar um conjunto de casos de teste. Os métodos de particionamento de equivalência, análise de valor de contorno, representação gráfica de causa e efeito e métodos de adivinhação de erros descritos no Capítulo 4 são especialmente pertinentes ao teste de função. Na verdade, os exemplos do Capítulo 4 são exemplos de testes de função. As descrições da instrução Fortran DIMENSION , o programa de pontuação do exame e o comando DISPLAY são exemplos de especificações externas. Não são, contudo, exemplos completamente realistas; por exemplo, uma especificação externa real para o programa de pontuação incluiria uma descrição precisa do formato dos relatórios. (Nota: Como discutimos testes de função no Capítulo 4, não apresentamos exemplos de testes de função nesta seção.)

Muitas das diretrizes que fornecemos no Capítulo 2 também são particularmente pertinentes ao teste de função. Em particular, acompanhe quais funções exibiram o maior número de erros; essa informação é valiosa porque informa que essas funções provavelmente também contêm a preponderância de erros ainda não detectados. Além disso, lembre-se de focar uma quantidade suficiente de atenção em condições de entrada inválidas e inesperadas. (Lembre-se de que a definição do resultado esperado é uma parte vital de um caso de teste.)

Finalmente, como sempre, tenha em mente que o objetivo do teste de função é expor erros e discrepâncias com a especificação, não demonstrar que o programa corresponde à sua especificação externa.

Teste do sistema

O teste do sistema é o processo de teste mais incompreendido e mais difícil. O teste de sistema não é um processo de teste das funções do sistema ou programa completo, porque isso seria redundante com o processo de teste de função. Em vez disso, como mostrado na Figura 6.3, o teste do sistema tem um

propósito particular: comparar o sistema ou programa com seus objetivos originais. Dado este propósito, considere estas duas implicações:

1. O teste do sistema não se limita aos sistemas. Se o produto for um programa, teste de sistema é o processo de tentar demonstrar como o programa, como um todo, não atinge seus objetivos.
2. O teste do sistema, por definição, é impossível se não houver um conjunto de objetivos escritos e mensuráveis para o produto.

Ao procurar discrepâncias entre o programa e seus objetivos, focar nos erros de tradução cometidos durante o processo de concepção do especificação externa. Isso torna o teste do sistema um processo de teste vital, pois em termos do produto, do número de erros cometidos e da gravidade desses erros, esta etapa do ciclo de desenvolvimento geralmente é a mais propenso a erros.

Também implica que, ao contrário do teste de função, a especificação externa não pode ser usado como base para derivar os casos de teste do sistema, pois isso subverteria o propósito do teste do sistema. Por outro lado, o documento de objetivos não pode ser usado por si só para formular casos de teste, uma vez que não contém, por definição, descrições precisas das funções do programa interfaces externas. Resolvemos este dilema usando o user do programa documentação ou publicações - projete o teste do sistema analisando o Objetivos; formular casos de teste analisando a documentação do usuário. este tem o efeito colateral útil de comparar o programa com seus objetivos e a documentação do usuário, bem como comparar a documentação do usuário com os objetivos, conforme mostrado na Figura 6.4.

A Figura 6.4 ilustra por que o teste do sistema é o processo de teste mais difícil. A seta mais à esquerda na figura, comparando o programa com seus objetivos, é o objetivo central do teste do sistema, mas não há metodologias de projeto de casos de teste conhecidas. A razão para isso é que os objetivos declaram o que um programa deve fazer e quão bem o programa deve fazê-lo, mas eles fazem não informar a representação das funções do programa. Por exemplo, o objetivos para o comando DISPLAY especificados no Capítulo 4 podem ter leia a seguir:

> Será fornecido um comando para visualizar, a partir de um terminal, o conteúdo de principais locais de armazenamento. Sua sintaxe deve ser consistente com a taxa de sintaxe de todos os outros comandos do sistema. O usuário deve ser capaz de especificar

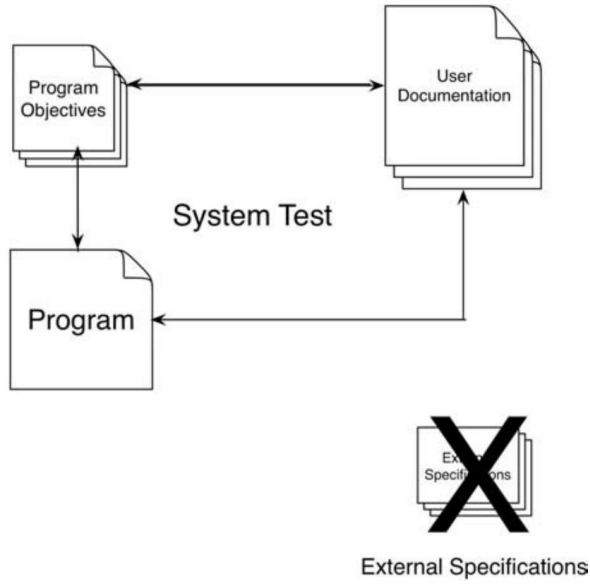

FIGURA 6.4 O Teste do Sistema.

um intervalo de locais, por meio de um intervalo de endereços ou um endereço e uma contagem.
Padrões sensatos devem ser fornecidos para operandos de comando.

A saída deve ser exibida como várias linhas de várias palavras (em hexadecimal), com espaçamento entre as palavras. Cada linha deve conter o endereço da primeira palavra dessa linha. O comando é um comando "trivial", o que significa que sob cargas razoáveis do sistema, ele deve começar a exibir a saída dentro de dois segundos, e deve haver não haja tempo de atraso observável entre as linhas de saída. Uma programação erro no processador de comando deve, na pior das hipóteses, causar a falha do comando; o sistema e a sessão do usuário não devem ser afetados.
O processador de comandos não deve ter mais de um erro detectado pelo usuário após o sistema ser colocado em produção.

Dada a declaração de objetivos, não há metodologia identificável que produziria um conjunto de casos de teste, além da linha de orientação vaga, mas útil, de escrever casos de teste para tentar mostrar que o programa é consistente com cada sentença da declaração de objetivos. Daí, um abordagem diferente para o projeto de casos de teste é tomada aqui: Em vez de descrever uma metodologia, são discutidas categorias distintas de casos de teste do sistema. Por causa da ausência de uma metodologia, o teste do sistema requer um

quantidade de criatividade; de fato, o projeto de bons casos de teste de sistema requer mais criatividade, inteligência e experiência do que o necessário para projetar o próprio sistema ou programa.

A Tabela 6.1 lista 15 categorias de casos de teste, juntamente com uma breve descrição. Discutimos as categorias aqui. Não afirmamos que todas as 15 categorias se aplicam a todos os programas, mas para evitar negligenciar algo, recomendamos que você explore todas elas ao projetar casos de teste.

TABELA 6.1 15 Categorias de Casos de Teste

Categoria	Descrição
Instalação	Garantir que a funcionalidade nos objetivos seja implementada.
Volume	Sujeite o programa a volumes anormalmente grandes de dados para processo.
Estresse	Sujeitar o programa a cargas anormalmente grandes, geralmente processamento simultâneo.
Usabilidade	Determine quão bem o usuário final pode interagir com o programa.
Segurança	Tente subverter as medidas de segurança do programa.
atuação	Determinar se o programa atende a resposta e requisitos de rendimento.
Armazenar	Certifique-se de que o programa gerencie corretamente suas necessidades de armazenamento, tanto sistema e físico.
Configuração	Verifique se o programa funciona adequadamente no configurações recomendadas.
Compatibilidade/ Conversão	Determinar se novas versões do programa são compatível com versões anteriores.
Instalação	Certifique-se de que os métodos de instalação funcionem em todos os plataformas.
Confiabilidade	Determinar se o programa atende às especificações de confiabilidade como tempo de atividade e MTBF.
Recuperação	Teste se as instalações de recuperação do sistema funcionam conforme projetado.
Facilidade de manutenção/ Manutenção	Determine se o aplicativo fornece corretamente mecanismos para gerar dados sobre eventos que requerem suporte técnico.
Documentação	Valide a precisão de toda a documentação do usuário.
Procedimento	Determinar a precisão dos procedimentos especiais necessários para usar ou manter o programa.

Teste de instalações

O tipo mais óbvio de teste de sistema é determinar se cada instalação (ou função; mas a palavra "função" não é usada aqui para evitar confundir isso com teste de função) mencionada nos objetivos foi realmente
implementado. O procedimento é escanear os objetivos frase por frase, e quando uma frase especifica um quê (por exemplo, "a sintaxe deve ser consistente..." "o usuário deve ser capaz de especificar um intervalo de localizações..."),
determinar que o programa satisfaz o "o quê". Esse tipo de teste geralmente
pode ser executado sem um computador; uma comparação mental dos objetivos com a documentação do usuário às vezes é suficiente. Mesmo assim, um
lista de verificação é útil para garantir que você verifique mentalmente os mesmos objetivos na próxima vez que você realizar o teste.

Teste de volume

Um segundo tipo de teste de sistema é submeter o programa a grandes volumes
De dados. Por exemplo, um compilador pode receber um programa fonte absurdamente grande para compilar. Um editor de ligação pode ser alimentado com um programa contendo milhares de módulos. Um simulador de circuito eletrônico pode receber um circuito contendo milhões de componentes. A fila de trabalhos de um sistema operacional pode ser preenchido até a capacidade. Se um programa deve manipular arquivos que abrangem vários volumes, são criados dados suficientes para fazer com que o programa mude de um volume para outro. Em outras palavras, o objetivo do teste de volume é
mostrar que o programa não pode lidar com o volume de dados especificado em seu Objetivos.

Obviamente, o teste de volume pode exigir recursos significativos, portanto, em termos de tempo de máquina e pessoas, você não deve exagerar. Ainda, cada programa deve ser exposto a pelo menos alguns testes de volume.

Teste de estresse

O teste de estresse submete o programa a cargas pesadas ou estresses. Isto deveria não confunda com teste de volume; um estresse pesado é um volume de pico de dados, ou atividade, encontrados em um curto espaço de tempo. Uma analogia seria avaliando um datilógrafo: Um teste de volume determinaria se o datilógrafo
poderia lidar com um rascunho de um grande relatório; um teste de estresse determinaria se o datilógrafo poderia digitar a uma taxa de 50 palavras por minuto.

Como o teste de estresse envolve um elemento de tempo, não é aplicável a muitos programas - por exemplo, um compilador ou uma folha de pagamento de processamento em lote programa. É aplicável, no entanto, a programas que operam sob diferentes cargas ou programas interativos, em tempo real e de controle de processos. Se um sistema de controle de tráfego aéreo deve rastrear até 200 aviões em seu setor, você pode fazer um teste de estresse simulando a presença de 200 aviões. Desde não há nada que impeça fisicamente um avião 201 de entrar no setor, um teste de estresse adicional exploraria a reação do sistema a esse inesperado avião. Um teste de estresse adicional pode simular a entrada simultânea de um grande número de aviões no setor.

Se um sistema operacional deve suportar um máximo de 15 trabalhos simultâneos, o sistema pode ficar sobrecarregado ao tentar executar 15 trabalhos simultaneamente. Você pode enfatizar um simulador de aeronave de treinamento de pilotos determinar a reação do sistema a um trainee que força o leme para a esquerda, puxa o acelerador, abaixa os flaps, levanta o nariz, abaixa o trem de pouso, acende as luzes de pouso e inclina para a esquerda, tudo ao mesmo tempo. (Tais casos de teste podem exigir um piloto de quatro mãos ou, realisticamente, dois especialistas no cockpit.) Você pode testar um sistema de controle de processo fazendo com que todos os processos monitorados gerem sinais simultaneamente, ou um sistema de comutação telefônica roteando para ele um grande número de telefonemas simultâneos.

Aplicativos baseados na Web são assuntos comuns de testes de estresse. Aqui, você deseja garantir que seu aplicativo e hardware possam lidar com um volume de destino de usuários simultâneos. Você pode argumentar que pode ter milhões de pessoas acessando o site ao mesmo tempo, mas isso não é realista. Você precisa definir seu público e, em seguida, criar um teste de estresse para representar o número máximo de usuários que você acha que usarão seu site. (O Capítulo 10 fornece mais informações sobre como testar aplicativos baseados na Web.)

Da mesma forma, você pode enfatizar um aplicativo de dispositivo móvel - um telefone celular sistema operacional, por exemplo - lançando vários aplicativos que correr e permanecer residente, fazendo ou recebendo um ou mais telefones chamadas. Você pode lançar um programa de navegação GPS, um aplicativo que usa recursos de CPU e radiofrequência (RF) quase continuamente, então tentar usar outros aplicativos ou fazer chamadas telefônicas. (Capítulo 11 discute o teste de aplicativos móveis com mais detalhes.)

Embora muitos testes de estresse representem condições que o programa provavelmente experimentará durante sua operação, outros podem representar verdadeiramente situações "nunca ocorrerão"; mas isso não significa que esses testes sejam

nao é útil. Se essas condições impossíveis detectarem erros, o teste é valioso porque é provável que os mesmos erros também possam ocorrer em situações realistas e menos estressantes.

Testando usabilidade

Outra área importante do caso de teste é a usabilidade, ou teste de usuário. Embora essa técnica de teste tenha quase 30 anos, ela se tornou mais importante com o advento de mais softwares baseados em GUI e a profunda penetração de hardware e software de computador em todos os aspectos de nossa sociedade. Ao incumbir o usuário final final de um aplicativo de testar o software em um ambiente do mundo real, problemas potenciais podem ser descobertos que até mesmo o roteamento de teste automatizado mais agressivo provavelmente não encontraria. Essa área de teste de software é tão importante que a abordaremos com mais detalhes no próximo capítulo.

Teste de segurança

Em resposta à crescente preocupação da sociedade com a privacidade, muitos programas agora têm objetivos específicos de segurança. O teste de segurança é o processo de tentar criar casos de teste que subvertam as verificações de segurança do programa.
Por exemplo, você pode tentar formular casos de teste que contornam o mecanismo de proteção de memória de um sistema operacional. Da mesma forma, você pode tentar subverter os mecanismos de segurança de dados de um sistema de banco de dados. Uma maneira de conceber tais casos de teste é estudar problemas de segurança conhecidos em sistemas semelhantes e gerar casos de teste que tentem demonstrar problemas comparáveis no sistema que você está testando. Por exemplo, fontes publicadas em revistas, salas de bate-papo ou grupos de notícias frequentemente cobrem bugs conhecidos em sistemas operacionais ou outros sistemas de software. Ao pesquisar falhas de segurança em programas existentes que fornecem serviços semelhantes ao que você está testando, você pode criar casos de teste para determinar se seu programa sofre do mesmo tipo de problema.

Os aplicativos baseados na Web geralmente precisam de um nível mais alto de teste de segurança do que a maioria dos aplicativos. Isso é especialmente verdadeiro para sites de comércio eletrônico. Embora exista tecnologia suficiente, ou seja, criptografia, para permitir que os clientes concluam transações com segurança pela Internet, você não deve confiar na mera aplicação de tecnologia para garantir a segurança. Além disso, você precisará convencer sua base de clientes de que sua aplicação é segura,

ou corre o risco de perder clientes. Novamente, o Capítulo 10 fornece mais informações sobre testes de segurança em aplicativos baseados na Internet.

Teste de performance

Muitos programas têm objetivos específicos de desempenho ou eficiência, declarando propriedades como tempos de resposta e taxas de rendimento sob determinadas condições de carga de trabalho e configuração. Novamente, como o objetivo de um teste de sistema é demonstrar que o programa não atende aos seus objetivos, os casos de teste devem ser projetados para mostrar que o programa não atende aos seus objetivos de desempenho.

Teste de armazenamento

Da mesma forma, os programas ocasionalmente têm objetivos de armazenamento que informam, por exemplo, a quantidade de memória do sistema que o programa usa e o tamanho dos arquivos temporários ou de log. Você precisa verificar se seu programa pode controlar o uso da memória do sistema para que não afete negativamente outros processos em execução no host. O mesmo vale para arquivos físicos no sistema de arquivos. O preenchimento de uma unidade de disco pode causar um tempo de inatividade significativo. Você deve projetar casos de teste para mostrar que esses objetivos de armazenamento não foram atendidos.

Teste de configuração

Programas como sistemas operacionais, sistemas de gerenciamento de banco de dados e programas de mensagens suportam uma variedade de configurações de hardware, incluindo vários tipos e números de dispositivos de E/S e linhas de comunicação ou diferentes tamanhos de memória. Muitas vezes, o número de configurações possíveis é muito grande para testar cada uma, mas no mínimo, você deve testar o programa com cada tipo de dispositivo de hardware e com a configuração mínima e máxima. Se o próprio programa puder ser configurado para omitir componentes do programa, ou se o programa puder ser executado em computadores diferentes, cada configuração possível do programa deverá ser testada.

Hoje, muitos programas são projetados para vários sistemas operacionais. Assim, ao testar tal programa, você deve fazê-lo em todos os sistemas operacionais para os quais foi projetado. Programas projetados para serem executados em um navegador da Web requerem atenção especial, pois existem vários navegadores da Web disponíveis e nem todos funcionam da mesma maneira. Dentro

Além disso, o mesmo navegador da Web funcionará de maneira diferente em sistemas operacionais diferentes.

Teste de compatibilidade/conversão

A maioria dos programas desenvolvidos não são completamente novos; muitas vezes são substitutos de algum sistema deficiente. Como tal, os programas geralmente têm objetivos específicos em relação à sua compatibilidade e procedimentos de conversão do sistema existente. Novamente, ao testar o programa em relação a esses objetivos, a orientação dos casos de teste é demonstrar que os objetivos de compatibilidade não foram atendidos e que os procedimentos de conversão não funcionam. Aqui você tenta gerar erros ao mover dados de um sistema para outro. Um exemplo seria atualizar um sistema de banco de dados.

Você deseja garantir que a nova versão seja compatível com seus dados existentes, assim como precisa validar que uma nova versão de um aplicativo de processamento de texto seja compatível com seus formatos de documento anteriores. Existem vários métodos para testar esse processo; no entanto, eles são altamente dependentes do sistema de banco de dados que você emprega.

Teste de instalação

Alguns tipos de sistemas de software têm procedimentos de instalação complicados. Testar o procedimento de instalação é uma parte importante do processo de teste do sistema. Isso é particularmente verdadeiro para um sistema de instalação automatizado que faz parte do pacote do programa. Um programa de instalação com defeito pode impedir que o usuário tenha uma experiência bem-sucedida com o sistema principal que você está testando. A primeira experiência de um usuário é quando ele instala o aplicativo. Se essa fase tiver um desempenho ruim, o usuário/cliente pode encontrar outro produto ou ter pouca confiança na validade do aplicativo.

Teste de confiabilidade

É claro que o objetivo de todos os tipos de teste é a melhoria da confiabilidade do programa, mas se os objetivos do programa contiverem declarações específicas sobre confiabilidade, testes de confiabilidade específicos podem ser planejados. Testar objetivos de confiabilidade pode ser difícil. Por exemplo, um sistema on-line moderno, como uma rede corporativa de longa distância (WAN) ou um provedor de serviços de Internet (ISP), geralmente tem um tempo de atividade desejado de 99,97% ao longo da vida útil do

sistema. Não há nenhuma maneira conhecida de testar esse objetivo dentro de um período de teste de meses ou mesmo anos. Os sistemas de software críticos de hoje têm padrões de confiabilidade ainda mais altos, e o hardware de hoje deve suportar esses objetivos. Você potencialmente pode testar programas ou sistemas com objetivos de tempo médio entre falhas (MTBF) mais modestos ou objetivos de erro operacional razoáveis (em termos de teste).

Um MTBF de não mais de 20 horas, ou um objetivo de que um programa não apresente mais de 12 erros únicos após ser colocado em produção, por exemplo, apresenta possibilidades de teste, particularmente para estatística, comprovação de programa ou baseado em modelo. metodologias de teste. Esses métodos estão além do escopo deste livro, mas a literatura técnica (online e não) oferece ampla orientação nessa área. Por exemplo, se essa área de teste de programa for de seu interesse, pesquise o conceito de asserções indutivas. O objetivo deste método é o desenvolvimento de um conjunto de teoremas sobre o programa em questão, cuja demonstração garante a ausência de erros no programa. O método começa escrevendo asserções sobre as condições de entrada do programa e resultados corretos. As asserções são expressas simbolicamente em um sistema lógico formal, geralmente o cálculo de predicados de primeira ordem. Você então localiza cada loop no programa e, para cada loop, escreve uma asserção declarando as condições invariantes (sempre verdadeiras) em um ponto arbitrário do loop. O programa agora foi particionado em um número fixo de caminhos de comprimento fixo (todos os caminhos possíveis entre um par de asserções). Para cada caminho, você pega a semântica das instruções do programa intervenientes para modificar a asserção e, eventualmente, chega ao final do caminho. Neste ponto, existem duas asserções no final do caminho: a original e a derivada da asserção na extremidade oposta.

Você então escreve um teorema afirmando que a afirmação original implica a afirmação derivada e tenta provar o teorema. Se os teoremas puderem ser provados, você pode assumir que o programa não contém erros - contanto que o programa termine. Uma prova separada é necessária para mostrar que o programa sempre terminará eventualmente.

Por mais complexo que pareça esse tipo de prova ou previsão de software, testes de confiabilidade e, de fato, o conceito de engenharia de confiabilidade de software (SRE) estão conosco hoje e são cada vez mais importantes para sistemas que devem manter tempos de atividade muito altos. Para ilustrar esse ponto, examine a Tabela 6.2 para ver o número de horas por ano que um sistema deve funcionar para dar suporte a vários requisitos de tempo de atividade. Esses valores devem indicar a necessidade de SRE.

TABELA 6.2 Horas por ano para vários requisitos de tempo de atividade

Requisitos de porcentagem de tempo de atividade	Horas de funcionamento por ano
100	8760,0
99,9	8751.2
98	8584,8
97	8497.2
96	8409.6
95	8322,0

Teste de recuperação

Programas como sistemas operacionais, sistemas de gerenciamento de banco de dados e programas de teleprocessamento geralmente têm objetivos de recuperação que indicam como o sistema é se recuperar de erros de programação, falhas de hardware e dados erros. Um objetivo do teste do sistema é mostrar que essas funções de recuperação não funcionam corretamente. Erros de programação podem ser injetados propositalmente em um sistema para determinar se ele pode se recuperar deles. Hardware falhas como erros de paridade de memória ou erros de dispositivo de E/S podem ser simuladas. Erros de dados, como ruído em uma linha de comunicação ou um inválido ponteiro em um banco de dados pode ser criado propositalmente ou simulado para analisar o reação do sistema.

Um objetivo de projeto de tais sistemas é minimizar o tempo médio de recuperação (MTTR). O tempo de inatividade geralmente faz com que uma empresa perca receita porque o sistema está inoperante. Um objetivo do teste é mostrar que o sistema não cumpre o acordo de nível de serviço para MTTR. Muitas vezes, o MTTR têm um limite superior e inferior, portanto, seus casos de teste devem refletir esses limites.

Teste de manutenção/manutenção

O programa também pode ter objetivos para sua operacionalidade ou manter características de habilidade. Todos os objetivos desse tipo devem ser testados. Tais objetivos podem definir os auxílios de serviço a serem fornecidos com o sistema, incluindo programas de despejo de armazenamento ou diagnósticos, o tempo médio para depurar um problema aparente, os procedimentos de manutenção e a qualidade da documentação da lógica interna.

Teste de Documentação

Como ilustramos na Figura 6.4, o teste do sistema também se preocupa com a precisão da documentação do usuário. A principal forma de realizar este teste é usar a documentação para determinar a representação do casos de teste de sistema anteriores. Isto é, uma vez que um caso particular de estresse é planejado, você usaria a documentação como um guia para escrever o caso de teste real. Além disso, a própria documentação do usuário deve ser objeto de uma inspeção (semelhante ao conceito de inspeção de código no Capítulo 3), para verificar precisão e clareza. Quaisquer exemplos ilustrados na documentação devem ser codificados em casos de teste e inseridos no programa.

Teste de procedimento

Finalmente, muitos programas são partes de sistemas maiores, não completamente automatizados, envolvendo procedimentos que as pessoas executam. Quaisquer procedimentos humanos prescritos, como aqueles para o operador do sistema, administrador de banco de dados usuário final, deve ser testado durante o teste do sistema.

Por exemplo, um administrador de banco de dados deve documentar procedimentos para backup e recuperação do sistema de banco de dados. Se possível, uma pessoa não associados à administração do banco de dados devem testar os procedimentos. No entanto, uma empresa deve criar os recursos necessários para testar adequadamente os procedimentos. Esses recursos geralmente incluem hardware e licenciamento de software adicional.

Executando o teste do sistema

Uma das considerações mais importantes na implementação do teste do sistema é determinar quem deve fazê-lo. Para responder a isso de forma negativa, (1) os programadores não devem realizar um teste de sistema; e (2) de todos os testes fases, esta é a que a organização responsável por desenvolver o programas definitivamente não devem funcionar.

O primeiro ponto decorre do fato de que uma pessoa executando um sistema teste deve ser capaz de pensar como um usuário final, o que implica uma compreensão completa das atitudes e do ambiente do usuário final e de como o programa será usado. Obviamente, então, se viável, um bom candidato a teste é um ou mais usuários finais. No entanto, como o fim típico usuário não terá a habilidade ou experiência para executar muitas das

categorias de testes descritas anteriormente, uma equipe de teste de sistema ideal pode ser composta por alguns especialistas profissionais em teste de sistema (pessoas que passam a vida realizando testes de sistema), um usuário final representativo ou dois, um engenheiro de fatores humanos e os principais analistas originais. ou designers do programa. Incluir os projetistas originais não viola o princípio 2 da Tabela 2.1, "Diretrizes de Teste de Programas Vitais", recomendando não testar seu próprio programa, já que o programa provavelmente passou por muitas mãos desde que foi concebido. Portanto, os designers originais não têm os incômodos vínculos psicológicos com o programa que motivou esse princípio.

O segundo ponto decorre do fato de que um teste de sistema é uma atividade do tipo “vale tudo, não há restrições”. Novamente, a organização de desenvolvimento tem vínculos psicológicos com o programa que são contrários a esse tipo de atividade. Além disso, a maioria das organizações de desenvolvimento está mais interessada em que o teste do sistema prossiga da maneira mais suave possível e dentro do cronograma, portanto, não está realmente motivada a demonstrar que o programa não atende aos seus objetivos. No mínimo, o teste do sistema deve ser realizado por um grupo independente de pessoas com poucos ou nenhum vínculo com a organização de desenvolvimento.

Talvez a maneira mais econômica de realizar um teste de sistema (econômico em termos de encontrar o maior número de erros com uma determinada quantia de dinheiro, ou gastar menos dinheiro para encontrar o mesmo número de erros), seja subcontratar o teste a uma empresa separada. Falamos mais sobre isso na última seção deste capítulo.

Teste de aceitação

Voltando ao modelo geral do processo de desenvolvimento mostrado na Figura 6.3, você pode ver que o teste de aceitação é o processo de comparar o programa com seus requisitos iniciais e as necessidades atuais de seus usuários finais. É um tipo incomum de teste, pois geralmente é realizado pelo cliente ou usuário final do programa e normalmente não é considerado responsabilidade da organização de desenvolvimento. No caso de um programa contratado, a organização contratante (usuária) realiza o teste de aceitação comparando a operação do programa com o contrato original.

Como é o caso de outros tipos de teste, a melhor maneira de fazer isso é criar casos de teste que tentem mostrar que o programa não atende ao contrato; se esses casos de teste não forem bem-sucedidos, o programa será aceito. No

No caso de um produto de programa, como o sistema operacional de um fabricante de computador ou o sistema de banco de dados de uma empresa de software, o cliente sensato primeiro realiza um teste de aceitação para determinar se o produto satisfaz suas necessidades.

Embora o teste de aceitação final seja, de fato, responsabilidade do cliente ou usuário final, o desenvolvedor experiente conduzirá os testes do usuário durante o ciclo de desenvolvimento e antes de entregar o produto acabado ao usuário final ou cliente contratado. Consulte o Capítulo 7 para obter mais informações sobre teste de usuário ou usabilidade.

Teste de instalação

O processo de teste restante na Figura 6.3 é o teste de instalação. Sua posição na figura é um pouco incomum, pois não está relacionada, como todos os outros processos de teste, a fases específicas do processo de projeto. É um tipo incomum de teste porque seu objetivo não é encontrar erros de software, mas encontrar erros que ocorrem durante o processo de instalação.

Muitos eventos ocorrem durante a instalação de sistemas de software. Uma pequena lista de exemplos inclui o seguinte:

O usuário deve selecionar uma variedade de opções.
Arquivos e bibliotecas devem ser alocados e carregados.
As configurações de hardware válidas devem estar presentes.
Os programas podem precisar de conectividade de rede para se conectar a outros programas.

A organização que produziu o sistema deve desenvolver os testes de instalação, que devem ser entregues como parte do sistema e executados após a instalação do sistema. Entre outras coisas, os casos de teste podem verificar se um conjunto compatível de opções foi selecionado, se todas as partes do sistema existem, se todos os arquivos foram criados e têm o conteúdo necessário e se a configuração de hardware é apropriada.

Planejamento e Controle de Testes

Se você considerar que o teste de um sistema grande pode envolver escrever, executar e verificar dezenas de milhares de casos de teste, lidar com milhares de

de módulos, reparando milhares de erros e empregando centenas de pessoas em um período de um ano ou mais, é evidente que você enfrenta um imenso desafio de gerenciamento de projeto no planejamento, monitoramento e controle do processo de teste. Na verdade, o problema é tão grande que poderíamos dedicar um livro inteiro apenas ao gerenciamento de testes de software. A intenção desta seção é resumir algumas dessas considerações.

Conforme mencionado no Capítulo 2, o principal erro cometido com mais frequência no planejamento de um processo de teste é a suposição tácita de que nenhum erro será encontrado. O resultado óbvio desse erro é que os recursos planejados (pessoas, tempo de calendário e tempo de computador) serão grosseiramente subestimados, um problema notório na indústria de computação. Para agravar o problema, está o fato de que o processo de teste cai no final do ciclo de desenvolvimento, o que significa que as alterações de recursos são difíceis. Um segundo problema, talvez mais insidioso, é que a definição errada de teste está sendo usada, pois é difícil ver como alguém usando a definição correta de teste (o objetivo é encontrar erros) planejaria um teste usando a suposição de que não erros serão encontrados.

Como é o caso da maioria dos empreendimentos, o plano é a parte crucial da gestão do processo de teste. Os componentes de um bom plano de teste são os seguintes:

1. Objetivos. Os objetivos de cada fase de teste devem ser definidos.
2. Critérios de conclusão. Os critérios devem ser elaborados para especificar quando cada fase de teste será considerada completa. Este assunto é discutido na próxima seção.

3. Horários. Os horários do calendário são necessários para cada fase. Eles devem indicar quando os casos de teste serão projetados, escritos e executados. Algumas metodologias de software como Extreme Programming (discutidas no Capítulo 9) exigem que você projete os casos de teste e os testes de unidade antes do início da codificação do aplicativo.
4. Responsabilidades. Para cada fase, as pessoas que projetarão, escreverão, executarão e verificarão os casos de teste e as pessoas que repararão os erros descobertos devem ser identificadas. E, como em grandes projetos surgem inevitavelmente disputas sobre se determinados resultados de testes representam erros, um árbitro deve ser identificado.

5. Bibliotecas e padrões de casos de teste. Em um grande projeto, são necessários métodos sistemáticos de identificação, escrita e armazenamento de casos de teste.

6. Ferramentas. As ferramentas de teste necessárias devem ser identificadas, incluindo um plano para quem irá desenvolvê-los ou adquiri-los, como eles serão usados e quando eles serão necessários.
7. Hora do computador. Este é um plano para a quantidade de tempo de computador necessários para cada fase de teste. Incluiria servidores usados para compilar aplicativos, se necessário; máquinas desktop necessárias para testes de instalação; Servidores Web para aplicações baseadas na Web; em rede dispositivos, se necessário; e assim por diante.
8. Configuração de hardware. Se configurações ou dispositivos especiais de hardware são necessários, é necessário um plano que descreva os requisitos, como eles serão atendidos e quando serão necessários.
9. Integração. Parte do plano de teste é uma definição de como o programa serão reunidos (por exemplo, teste incremental de cima para baixo). Um sistema contendo os principais subsistemas ou programas pode ser montado de forma incremental, usando a abordagem de cima para baixo ou de baixo para cima, para instância, mas onde os blocos de construção são programas ou subsistemas, em vez de módulos. Se este for o caso, um plano de integração do sistema é necessário. O plano de integração do sistema define a ordem de integração, a capacidade funcional de cada versão do sistema e as responsabilidades de produzir "scaffolding", código que simula a função de componentes inexistentes.
10. Procedimentos de rastreamento. Você deve identificar meios para rastrear vários aspectos do progresso do teste, incluindo a localização de módulos propensos a erros e estimativa do progresso em relação ao cronograma, recursos, e critérios de conclusão.
11. Procedimentos de depuração. Você deve definir mecanismos para relatar erros detectados, acompanhar o andamento das correções e adicionar o correções no sistema. Cronogramas, responsabilidades, ferramentas e tempo/recursos do computador também devem fazer parte do plano de depuração.
12. Teste de regressão. O teste de regressão é realizado após fazer uma melhoria funcional ou reparo no programa. Seu propósito é determinar se a mudança regrediu outros aspectos do programa. Geralmente é executado executando novamente algum subconjunto de casos de teste do programa. O teste de regressão é importante porque as mudanças e correções de erros tendem a ser muito mais propensas a erros do que as código do programa original (da mesma forma que a maioria dos erros tipográficos em jornais são o resultado de editoriais de última hora

alterações, em vez de alterações na cópia original). Um plano para testes de regressão – quem, como, quando – também é necessário.

Critérios de conclusão do teste

Uma das questões mais difíceis de responder ao testar um programa é determinar quando parar, pois não há como saber se o erro detectado é o último erro remanescente. Na verdade, em qualquer coisa que não seja um programa pequeno, não é razoável esperar que todos os erros sejam eventualmente detectados. Dado esse dilema, e dado o fato de que a economia dita que os testes devem terminar, você pode se perguntar se a pergunta deve ser respondida de maneira puramente arbitrária ou se existem alguns critérios úteis de interrupção.

Os critérios de conclusão normalmente usados na prática são sem sentido e contraproducente. Os dois critérios mais comuns são estes:

1. Pare quando o tempo programado para o teste expirar.
2. Pare quando todos os casos de teste forem executados sem detectar erros - ou seja, parar quando os casos de teste não forem bem-sucedidos.

O primeiro critério é inútil porque você pode satisfazê-lo não fazendo absolutamente nada. Não mede a qualidade do teste. O segundo critério é igualmente inútil porque também é independente da qualidade dos casos de teste. Além disso, é contraproducente porque subconscientemente o encoraja a escrever casos de teste com baixa probabilidade de detectar erros.

Conforme discutido no Capítulo 2, os humanos são altamente orientados para objetivos. Se lhe disserem que terminou uma tarefa quando os casos de teste não tiveram sucesso, você inconscientemente escreverá casos de teste que levam a esse objetivo, evitando os casos de teste úteis, de alto rendimento e destrutivos.

Existem três categorias de critérios mais úteis. A primeira categoria, mas não a melhor, é basear a conclusão no uso de metodologias específicas de projeto de casos de teste. Por exemplo, você pode definir a conclusão do teste do módulo da seguinte forma:

Os casos de teste são derivados de (1) satisfazer o critério de cobertura multicondicional e (2) uma análise de valor limite do módulo

especificação de interface, e todos os casos de teste resultantes são eventualmente malsucedidos.

Você pode definir o teste de função como concluído quando o seguinte condições são satisfeitas:

Os casos de teste são derivados de (1) gráficos de causa-efeito, (2) análise de valor de limite e (3) suposição de erro, e todos os casos de teste resultantes são eventualmente malsucedidos.

Embora esse tipo de critério seja superior aos dois mencionados anteriormente, ele apresenta três problemas. Em primeiro lugar, não é útil em uma fase de teste em que metodologias específicas não estejam disponíveis, como a fase de teste do sistema. Em segundo lugar, trata-se de uma medida subjetiva, pois não há como garantir que uma pessoa tenha utilizado uma determinada metodologia, como a análise de valor limite, de forma adequada e rigorosa. Terceiro, em vez de definir uma meta e deixar o testador escolher a melhor maneira de alcançá-la, ele faz o oposto; metodologias de projeto de caso de teste são ditadas, mas nenhum objetivo é dado. Portanto, esse tipo de critério é útil às vezes para algumas fases de teste, mas deve ser aplicado somente quando o testador provou suas habilidades no passado ao aplicar as metodologias de projeto de casos de teste com sucesso.

A segunda categoria de critérios – talvez a mais valiosa – é declarar os requisitos de conclusão em termos positivos. Como o objetivo do teste é encontrar erros, por que não fazer do critério de conclusão a detecção de um número predefinido de erros? Por exemplo, você pode declarar que um teste de módulo de um módulo específico não está completo até que três erros sejam descobertos. Talvez o critério de conclusão de um teste de sistema deva ser definido como a detecção e reparo de 70 erros, ou um tempo decorrido de três meses, o que ocorrer depois.

Observe que, embora esse tipo de critério reforce a definição de teste, ele apresenta dois problemas, ambos superáveis. Um problema é determinar como obter o número de erros a serem detectados.
A obtenção desse número requer as três estimativas a seguir:

1. Uma estimativa do número total de erros no programa.
2. Uma estimativa de qual porcentagem desses erros pode ser encontrada por meio de testes.

3. Uma estimativa de qual fração dos erros originou-se em processos de projeto específicos e durante quais fases de teste esses erros provavelmente serão detectados.

Você pode obter uma estimativa aproximada do número total de erros de várias maneiras. Um método é obtê-los através da experiência com programas anteriores. Além disso, existe uma variedade de módulos preditivos. Alguns deles exigem que você teste o programa por algum período de tempo, registre os tempos decorridos entre a detecção de erros sucessivos e insira esses tempos em parâmetros em uma fórmula. Outros módulos envolvem a propagação de erros conhecidos, mas não divulgados, no programa, testando o programa por um tempo e, em seguida, examinando a proporção de erros propagados detectados para erros não propagados detectados. Outro modelo emprega duas equipes de teste independentes cujos membros testam por um tempo, examinam os erros encontrados por cada uma e os erros detectados em comum por ambas as equipes e usam esses parâmetros para estimar o número total de erros. Outro método bruto para obter essa estimativa é usar médias de toda a indústria. Por exemplo, o número de erros que existem em programas típicos no momento em que a codificação é concluída (antes que um passo a passo ou inspeção de código seja empregado) é de aproximadamente 4 a 8 erros por 100 instruções de programa.

A segunda estimativa da lista anterior (a porcentagem de erros que podem ser encontrados de forma viável por meio de testes) envolve uma estimativa um tanto arbitrária, levando em consideração a natureza do programa e as consequências de erros não detectados.

Dada a atual escassez de informações sobre como e quando os erros são cometidos, a terceira estimativa é a mais difícil. Os dados existentes indicam que, em grandes programas, aproximadamente 40% dos erros são erros de codificação e de projeto lógico, e que o restante é gerado nos processos de projeto anteriores.

Para usar este critério, você deve desenvolver suas próprias estimativas que sejam pertinentes ao programa em questão. Um exemplo simples é apresentado aqui. Suponha que estamos prestes a começar a testar um programa de 10.000 instruções, que o número de erros remanescentes após a execução das inspeções de código é estimado em 5 por 100 instruções, e estabelecemos como objetivo a detecção de 98% da codificação e do projeto lógico erros e 95 por cento dos erros de projeto. O número total de erros é, portanto, estimado em 500. Dos 500 erros, assumimos que 200 são erros de codificação e design lógico e

TABELA 6.3 Estimativa hipotética de quando os erros podem ser encontrados

	Erros de codificação e design lógico	Erros de projeto
Teste do módulo	65%	0%
Teste de funcionamento	30%	60%
Teste do sistema	3%	35%
Total	98%	95%

300 são falhas de projeto. Portanto, o objetivo é encontrar 196 codificação e design lógico erros e 285 erros de projeto. Uma estimativa plausível de quando os erros são provável de ser detectado é mostrado na Tabela 6.3.

Se agendarmos quatro meses para testes de função e três meses para teste do sistema, os três critérios de conclusão a seguir podem ser estabelecido:

1. O teste do módulo é concluído quando 130 erros são encontrados e corrigidos (65 por cento dos 200 erros estimados de codificação e design lógico).
2. O teste de função é concluído quando 240 erros (30 por cento de 200 mais 60 por cento de 300) são encontrados e corrigidos, ou quando quatro meses de teste de função foram concluídos, o que ocorrer mais tarde. A razão para a segunda cláusula é que se encontrarmos 240 erros rapidamente, é provavelmente uma indicação de que subestimamos o número total de erros e, portanto, não deve interromper o teste de função cedo.
3. O teste do sistema é concluído quando 111 erros são encontrados e corrigidos, ou quando três meses de teste do sistema forem concluídos, o que ocorrer mais tarde.

O outro problema óbvio com esse tipo de critério é o da superestimação. E se, no exemplo anterior, houver menos de 240 erros restante quando o teste de função começa? Com base no critério, podemos nunca complete a fase de teste de função.

Este é um problema estranho se você pensar bem: não temos o suficiente erros; o programa é bom demais. Você poderia rotulá-lo como não um problema porque é o tipo de problema que muitas pessoas gostariam de ter. Se isso acontecer ocorrer, um pouco de bom senso pode resolvê-lo. Se não encontrarmos 240 erros em quatro meses, o gerente de projeto pode empregar uma pessoa de fora para analisar a

casos de teste para julgar se o problema é (1) casos de teste inadequados ou (2) casos de teste excelentes, mas falta de erros para detectar.

O terceiro tipo de critério de conclusão é fácil na superfície, mas envolve muito julgamento e intuição. Requer que você trace o número de erros encontrados por unidade de tempo durante a fase de teste. Ao examinar a forma da curva, muitas vezes você pode determinar se deve continuar a fase de teste ou encerrá-la e iniciar a próxima fase de teste.

Suponha que um programa esteja sendo testado em função e o número de erros encontrados por semana esteja sendo plotado. Se, na sétima semana, a curva for a de cima da Figura 6.5, seria imprudente interromper o teste de função, mesmo que tivéssemos atingido nosso critério de número de erros a serem encontrados.

Como na sétima semana ainda parecemos estar em alta velocidade (encontrando muitos erros), a decisão mais sábia (lembrando que nosso objetivo é encontrar erros) é continuar o teste de função, projetando casos de teste adicionais, se necessário.

Por outro lado, suponha que a curva seja a inferior da Figura 6.5.
A eficiência de detecção de erros caiu significativamente, o que implica que talvez tenhamos escolhido o teste de função de forma limpa e que talvez a melhor jogada seja encerrar o teste de função e iniciar um novo tipo de teste (um teste de sistema, talvez). É claro que também devemos considerar outros fatores, como se a queda na eficiência de detecção de erros foi devido à falta de tempo do computador ou ao esgotamento dos casos de teste disponíveis.

A Figura 6.6 é uma ilustração do que acontece quando você não consegue traçar o número de erros detectados. O gráfico representa três fases de teste de um sistema de software extremamente grande. Uma conclusão óbvia é que o projeto não deveria ter mudado para uma fase de teste diferente após o período 6. Durante o período 6, a taxa de detecção de erros foi boa (para um testador, quanto maior a taxa, melhor), mas mudar para uma segunda fase neste ponto fez com que a taxa de detecção de erros caísse significativamente.

O melhor critério de conclusão é provavelmente uma combinação dos três tipos que acabamos de discutir. Para o teste do módulo, principalmente porque a maioria dos projetos não rastreia formalmente os erros detectados durante esta fase, o melhor critério de conclusão é provavelmente o primeiro. Você deve solicitar que um conjunto específico de metodologias de projeto de caso de teste seja usado. Para as fases de teste de função e sistema, a regra de conclusão pode ser parar quando um número predefinido de erros for detectado ou quando o tempo programado tiver decorrido, o que ocorrer depois, mas desde que uma análise do gráfico de erros versus tempo indique que o teste se tornou improdutivo.

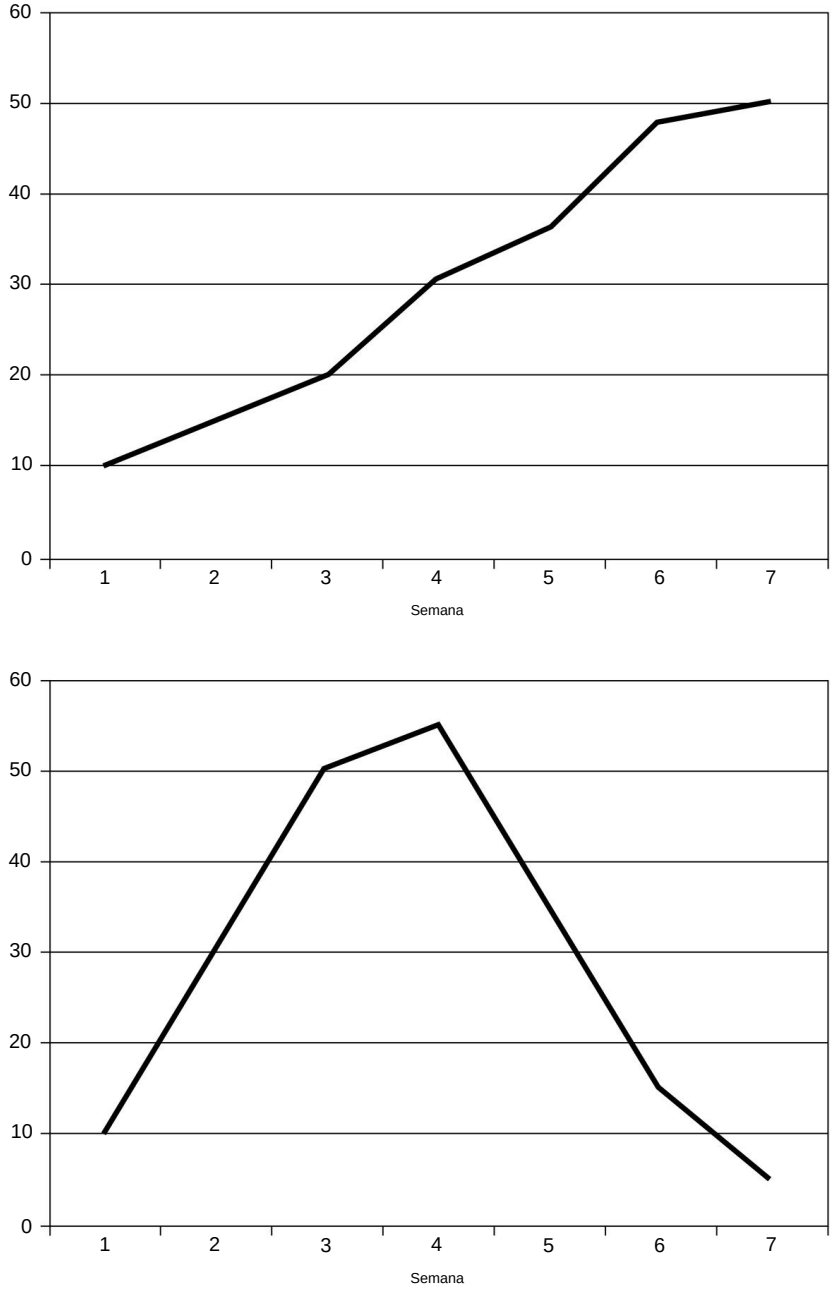

FIGURA 6.5 Estimativa de conclusão por plotagem de erros detectados por Unidade de tempo.

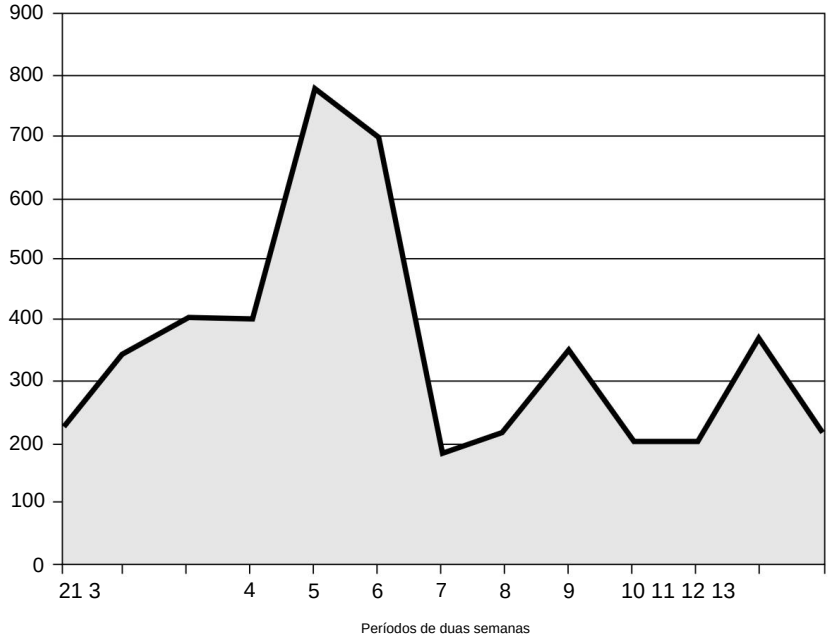

FIGURA 6.6 Estudo post mortem dos processos de teste de um grande Projeto.

A Agência de Testes Independente

Anteriormente neste capítulo e no Capítulo 2, enfatizamos que uma organização deve evitar tentar testar seus próprios programas. Nosso raciocínio é que a organização responsável pelo desenvolvimento de um programa tem dificuldade em testar objetivamente o mesmo programa. A organização do teste deve ser tão o mais distante possível, em termos da estrutura da empresa, da organização de desenvolvimento. Na verdade, é desejável que a organização do teste não fazer parte da mesma empresa, pois se for, ainda é influenciada pela mesma pressões gerenciais que influenciam a organização de desenvolvimento.

Uma maneira de evitar esse conflito é contratar uma empresa separada para o software teste. Esta é uma boa idéia, se a empresa que projetou o sistema e irá usá-lo desenvolveu o sistema, ou se um desenvolvedor de terceiros produziu o sistema. As vantagens geralmente observadas são o aumento da motivação no processo de teste, uma competição saudável com o desenvolvimento organização, remoção do processo de teste sob a gestão

controle da organização de desenvolvimento, e as vantagens de
conhecimento que a agência de teste independente traz para lidar com o problema.

Resumo

Testes de ordem superior podem ser considerados o próximo passo. Nós discutimos
e defendeu o conceito de teste de módulo - usando várias técnicas para
componentes de software de teste, os blocos de construção que se combinam para formar o
produto final. Com componentes individuais testados e depurados, é
tempo para ver o quão bem eles trabalham juntos.

Testes de ordem superior são importantes para todos os produtos de software, mas
tornam-se cada vez mais importantes à medida que o tamanho do projeto aumenta. Fica
raciocinar que quanto mais módulos e mais linhas de código um projeto contém, mais
oportunidades existem para erros de codificação ou mesmo de design.

O teste de função tenta descobrir erros de projeto, isto é, discrepâncias entre o programa
finalizado e suas especificações externas — uma descrição precisa do comportamento do
programa da perspectiva do usuário final.

O teste do sistema, por outro lado, testa a relação entre o
software e seus objetivos originais. O teste do sistema é projetado para descobrir
erros cometidos durante o processo de tradução dos objetivos do programa para o
especificação externa e, finalmente, em linhas de código. É esta tradução
etapa em que os erros têm os efeitos de maior alcance; da mesma forma, é o palco
no processo de desenvolvimento que é mais propenso a erros. Talvez a parte mais difícil do
teste do sistema seja projetar os casos de teste. Em geral você quer
para se concentrar nas principais categorias de teste e, em seguida, seja realmente criativo nos testes
essas categorias. A Tabela 6.1 resume 15 categorias que detalhamos neste
capítulo que pode orientar seus esforços de teste do sistema.

Não se engane, testes de ordem superior certamente são uma parte importante do
testes de software completos, mas também pode se tornar um processo assustador,
especialmente para sistemas muito grandes, como um sistema operacional. A chave para
o sucesso é um planejamento de teste consistente e bem planejado. Apresentamos este tópico
neste capítulo, mas se você estiver gerenciando o teste de sistemas grandes, mais
pensamento e planejamento serão necessários. Uma abordagem para lidar com isso é
contratar uma empresa externa para testes ou gerenciamento de testes.

No Capítulo 7, expandimos um aspecto importante do teste de ordem superior: teste de
usuário ou de usabilidade.