

20

TESTANDO APLICAÇÕES PARA WEB

CONCEITOS-CHAVE

dimensões da qualidade	469
estratégias	470
planejamento	470
teste de base de dados	473
teste de carga	486
teste de compatibilidade	478
teste de configuração	482
teste de conteúdo	472
teste de desempenho	485
teste de esforço (stress)	486
teste de interface	474
teste de navegação	480
teste de segurança	484
testes de usabilidade	477
teste em nível de componente	479

Há uma urgência que sempre permeia um projeto de WebApp (aplicações para Web). Os grupos de interessados – preocupados com a competição de outras WebApps, coagidos pelas exigências do cliente e temerosos com a perda de um nicho de mercado — exercem pressão para obtê-la on-line. Como resultado, as atividades técnicas que muitas vezes ocorrem mais tarde no processo, tais como teste de WebApp, dispõem algumas vezes de um prazo muito curto. Isso pode ser um erro catastrófico. Para tanto, os membros da equipe precisam assegurar que cada artefato tenha alta qualidade. Wallace e seus colegas [Wal03] observam isso quando afirmam:

O teste não deveria esperar até que o projeto esteja terminado. Comece o teste antes de escrever uma linha de código. Teste constante e efetivamente, e você desenvolverá um site muito mais duradouro.

Como os requisitos e modelos de projeto não podem ser testados no sentido clássico, toda a equipe deve realizar revisões técnicas (Capítulo 15) e também testes executáveis. A intenção é descobrir e corrigir erros antes que a WebApp seja disponibilizada aos usuários finais.

PANORAMA

O que é? O teste de WebApp é um conjunto de atividades relacionadas com um único objetivo: descobrir erros no conteúdo, na função, na usabilidade, na navegabilidade, no desempenho, na capacidade e na segurança da WebApp. Para tanto, deve ser utilizada uma estratégia de teste que abrange as revisões e o teste executável.

Quem realiza? Os engenheiros de aplicações para Web e outros interessados no projeto (gerentes, clientes e usuários finais), todos participam do teste da WebApp.

Porque é importante? Se os usuários finais encontrarem erros que abalem sua credibilidade na WebApp, buscarão em outro lugar pelo conteúdo e função que precisam, e a WebApp será um fracasso. Por essa razão, deve-se trabalhar para eliminar tantos erros quantos forem possíveis antes que a WebApp entre no ar.

Quais são as etapas envolvidas? O processo de teste começa focalizando os aspectos visíveis da WebApp ao usuário e passa para os testes de tecnologia e infraestrutura. Executam-se sete etapas de teste: de conteúdo, interface, navegação, componente, configuração, desempenho e segurança.

Qual é o artefato? Em algumas instâncias um plano de teste de uma WebApp é produzido. Em qualquer instância, é desenvolvida uma série de casos de testes para todas as etapas e é mantido um arquivo dos resultados para uso futuro.

Como garantir que o trabalho foi realizado corretamente? Embora nunca se possa ter certeza de ter executado todos os testes necessários, é possível verificar se erros foram apontados (e esses foram corrigidos). Além disso, se foi estabelecido um plano de teste, é aconselhável certificar-se de que todos os testes planejados foram realizados.

20.1 CONCEITOS DE TESTE PARA WEBAPPS

O teste é um processo pelo qual se experimenta o software com a intenção de encontrar (e por fim corrigir) erros. Essa filosofia fundamental, apresentada inicialmente no Capítulo 17, não muda para as WebApps. Na verdade, pelo fato de os sistemas e aplicações baseados na Web residirem em uma rede e interoperarem com muitos sistemas operacionais diferentes, browsers (residindo em uma variedade de dispositivos), plataformas de hardware, protocolos de comunicação e aplicações “de retaguarda” à procura dos erros representam um desafio significativo para os engenheiros.

Para entender os objetivos do teste no contexto de engenharia de Web, deve-se considerar as muitas dimensões da qualidade da WebApp.¹ Aqui, consideramos dimensões de qualidade particularmente relevantes em qualquer discussão de teste da WebApp. Consideramos também a natureza dos erros encontrados como consequência do teste e a estratégia de teste aplicada para descobrir esses erros.

20.1.1 Dimensões da qualidade

A qualidade é incorporada a uma aplicação Web como consequência de um bom projeto. Ela é avaliada aplicando-se uma série de revisões técnicas que investigam vários elementos do modelo de projeto e utilizando-se um processo de teste que é discutido no decorrer deste capítulo. Revisões e teste examinam ambos uma ou mais das seguintes dimensões da qualidade [Mil00a]:

- **Conteúdo:** é avaliado em nível sintático e semântico. No nível sintático, examina-se a ortografia, pontuação e gramática em documentos baseados em texto. No nível semântico, são analisadas: exatidão (das informações apresentadas), consistência (através de todo o objeto de conteúdo e objetos relacionados) e ausência de ambiguidade.
- **Função:** é testada para descobrir erros que indicam falta de conformidade com os requisitos do cliente. Cada função da WebApp é avaliada quanto à exatidão, instabilidade e conformidade geral com os padrões apropriados de implementação (por exemplo, padrões de linguagem Java ou AJAX).
- **Estrutura:** é avaliada para assegurar o fornecimento apropriado de conteúdo e função da WebApp, que seja extensível e que possa ser mantido na medida em que novo conteúdo ou nova funcionalidade são acrescentados.
- **Usabilidade:** é testada para garantir que cada categoria de usuário seja suportada pela interface e que possa aprender e aplicar todas as sintaxes e semânticas de navegação necessárias.
- **Navegabilidade:** é testada para assegurar que toda a sintaxe e semânticas de navegação sejam experimentadas para descobrir quaisquer erros de navegação (por exemplo, links inativos, impróprios, errados).
- **Desempenho:** é testado sob uma variedade de condições de operação, configurações e carga para assegurar que o sistema responda à interação com o usuário e suporte cargas extremas sem degradação inaceitável da operação.
- **Compatibilidade:** é testada executando-se a WebApp em uma variedade de diferentes configurações hospedeiras tanto no lado cliente quanto no lado servidor. A finalidade é encontrar erros específicos a determinada configuração de hospedeira.
- **Interoperabilidade:** é testada para garantir que a WebApp tenha uma interface adequada com outras aplicações e/ou bases de dados.
- **Segurança:** é testada para investigar vulnerabilidades potenciais e tentar explorar cada uma delas. Qualquer tentativa de invasão bem-sucedida é considerada uma falha de segurança.

Desenvolveram-se estratégias e táticas para teste de WebApp para experimentar cada uma dessas dimensões da qualidade. Posteriormente, neste capítulo, elas serão discutidas.

20.1.2 Erros em um ambiente WebApp

Os erros encontrados em consequência de um teste de WebApp bem-sucedido têm uma série de características especiais [Ngu00]:

¹ As dimensões de qualidade de software genérico, igualmente aplicável a WebApps, são discutidas no Capítulo 14.

 Como investigamos a qualidade no contexto de uma WebApp e seu ambiente?

“Inovação é o paraíso para os testadores de software. Justamente quando parece que sabemos como testar uma tecnologia em particular, uma nova tecnologia [WebApps] aparece e todas as possibilidades perdem sentido.”

James Bach

? O que torna os erros encontrados durante a execução de WebApp de certa forma diferentes daqueles identificados em software convencional?

1. Devido a muitos tipos de testes de WebApp detectarem problemas evidenciados primeiro no lado cliente (isto é, via interface implementada em um browser específico ou em um dispositivo de comunicação pessoal), muitas vezes nos deparamos com o indício de erro e não o erro em si.
2. Devido a uma WebApp ser implementada em diferentes configurações e ambientes, pode ser difícil ou impossível reproduzir um erro fora do ambiente no qual foi encontrado originalmente.
3. Embora alguns erros sejam resultado de projeto incorreto ou codificação HTML (ou outra linguagem de programação) incorreta, muitos erros podem ser atribuídos à configuração da WebApp.
4. Devido a WebApps residirem em uma arquitetura cliente-servidor, os erros podem ser difíceis de localizar através das três camadas arquitetônicas: o cliente, o servidor ou a própria rede.
5. Alguns erros são decorrentes do *ambiente operacional estático* (a configuração específica na qual o teste é executado), enquanto outros são atribuíveis ao ambiente operacional dinâmico (a carga instantânea de recursos ou erros relacionados com o tempo).

Esses cinco atributos de erros sugerem que o ambiente tem um papel importante no diagnóstico de todos os erros encontrados durante o teste da WebApp. Em algumas situações (por exemplo, teste de conteúdo), o local dos erros é óbvio, mas em muitos outros tipos de teste da WebApp (por exemplo, teste de navegação, teste de desempenho, teste de segurança) a causa subjacente do erro pode ser consideravelmente mais difícil de determinar.

20.1.3 Estratégia de teste

A estratégia para teste de WebApp adota os princípios básicos para todo o teste de software (Capítulo 17) e aplica estratégia e táticas recomendadas para sistemas orientados a objeto (Capítulo 19). Os passos a seguir resumem a abordagem:

1. O modelo de conteúdo para a WebApp é revisto para descobrir erros.
2. O modelo de interface é revisto para garantir que todos os casos de uso possam ser acomodados.
3. O modelo de projeto da WebApp é revisto para descobrir erros de navegação.
4. A interface com o usuário é testada para descobrir erros nos mecanismos de apresentação e/ou navegação.
5. Os componentes funcionais são submetidos a testes de unidade.
6. É testada a navegação por toda a arquitetura.
7. A WebApp é implementada em uma variedade de configurações ambientais diferentes e testada quanto à compatibilidade com cada configuração.
8. São executados testes de segurança na tentativa de explorar vulnerabilidades na WebApp ou em seu ambiente.
9. São realizados testes de desempenho.
10. A WebApp é testada por uma população de usuários finais controlados e monitorados; os resultados de suas interações com o sistema são avaliados quanto a erros de conteúdo e navegação, preocupações de usabilidade, preocupações de compatibilidade, segurança, confiabilidade e desempenho da WebApp.

Devido ao fato de muitas WebApps evoluírem continuamente, o processo de teste é uma atividade contínua, executada por pessoal de suporte que usa testes de regressão derivados dos desenvolvidos quando a WebApp foi inicialmente criada.

20.1.4 Planejamento de teste

O uso da palavra *planejamento* (em qualquer contexto) é anátema para alguns desenvolvedores da Web. Estes não planejam; simplesmente iniciam — na esperança de que irá surgir uma poderosa WebApp. Uma abordagem mais disciplinada reconhece que o planejamento estabelece um roteiro para todo o trabalho a ser feito. Isso compensa o esforço. Em seu livro sobre teste de WebApp, Splaine e Jaskiel [Spl01] afirmam:

PONTO-CHAVE

A estratégia geral para teste de WebApp pode ser resumida nos dez passos descritos a seguir.

WebRef

Excelentes artigos sobre teste de WebApp podem ser encontrados em www.stickyminds.com/testing.asp.

PONTO-CHAVE

O plano de teste identifica um conjunto de tarefas de teste, os artefatos a ser desenvolvidos e a maneira pela qual os resultados devem ser avaliados, registrados e reutilizados.

Exceto para o mais simples dos sites, torna-se imediatamente aparente que algum tipo de planejamento de teste seja necessário. Com muita frequência, o número inicial de erros encontrados por meio de um teste *ad hoc* é tão grande que nem todos são corrigidos quando detectados. Isso apresenta uma dificuldade adicional para os testadores de sites e aplicações. Eles devem não apenas evocar novos testes criativos, mas também lembrar como os anteriores foram executados para retestar o site/aplicação de forma confiável e assegurar que os erros conhecidos tenham sido removidos e que novos erros não tenham sido introduzidos.

As perguntas a ser feitas são: como “podemos evocar novos testes criativos”, e o que esses testes devem focalizar? As respostas a essas perguntas estão contidas em um plano de teste.

Um plano de teste da WebApp identifica (1) o conjunto de tarefas² a ser aplicado quando o teste começa, (2) os artefatos que serão produzidos na medida em que cada tarefa de teste é executada e (3) a maneira pela qual os resultados do teste são avaliados, registrados e reutilizados quando for executado o teste de regressão. Em alguns casos, o plano de teste é integrado ao plano do projeto. Em outros, o plano de teste é um documento separado.

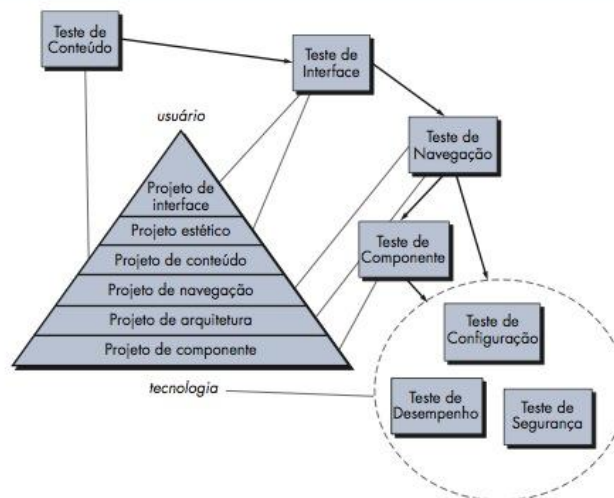
20.2 O PROCESSO DE TESTE – UMA VISÃO GERAL

O processo de teste da WebApp começa com testes que experimentam o conteúdo e a funcionalidade da interface, pontos imediatamente visíveis aos usuários finais. Na medida em que o teste é realizado, são experimentados aspectos da arquitetura de projeto e da navegação da WebApp. Por fim, o foco muda para testes que examinam os recursos tecnológicos que nem sempre são aparentes aos usuários finais – tópicos de infraestrutura e de instalação/implementação da WebApp.

A Figura 20.1 aproxima o processo de teste da WebApp da pirâmide de projeto para WebApps (Capítulo 13). Observe que enquanto ocorre o fluxo do teste da esquerda para a direita e de cima para baixo, os elementos do projeto da WebApp visíveis ao usuário (elementos do topo da pirâmide) são testados primeiro, seguidos pelos elementos de projeto da infraestrutura.

FIGURA 20.1

O processo de teste



² Conjuntos de tarefas são discutidos no Capítulo 2. Um termo relacionado – *fluxo de trabalho* – é também usado para descrever uma série de tarefas necessárias para exercer uma atividade de engenharia de software.

20.3 TESTE DE CONTEÚDO



Embora as revisões técnicas não façam parte do teste, a de conteúdo deverá ser executada para garantir que o conteúdo tenha qualidade.

Erros no conteúdo da WebApp podem ser tão triviais quanto pequenos erros tipográficos ou muito significativos como informações incorretas, organização inadequada ou violação de leis de propriedade intelectual. O teste de conteúdo tenta descobrir esses e muitos outros problemas antes que sejam encontrados pelos usuários.

Ele combina tanto revisões quanto geração de casos de testes executáveis. As revisões são aplicadas para descobrir erros semânticos em conteúdo (discutido na Seção 20.3.1). O teste executável é usado para descobrir erros de conteúdo que podem ser atribuídos a conteúdo derivado dinamicamente, controlado por dados adquiridos de um ou mais banco de dados.

20.3.1 Objetivos do teste de conteúdo

O teste de conteúdo tem três importantes objetivos: (1) descobrir erros de sintaxe (por exemplo, erros ortográficos, erros gramaticais) em documentos de texto, representações gráficas e outros meios; (2) descobrir erros de semântica (isto é, erros na exatidão ou integralidade das informações) em qualquer objeto de conteúdo apresentado quando ocorre a navegação e (3) encontrar erros na organização ou estrutura do conteúdo apresentado ao usuário final.

Para atingir o primeiro objetivo, usam-se verificadores automáticos de ortografia e gramática. Porém, muitos erros de sintaxe fogem à detecção por essas ferramentas e devem ser descobertos por um revisor humano (testador). De fato, um grande site pode contratar os serviços de um revisor profissional para descobrir erros de ortografia, erros de gramática, erros na consistência do conteúdo, erros em representações gráficas e erros de referência cruzada.

O teste de semântica concentra-se nas informações apresentadas em cada objeto de conteúdo. O revisor (testador) deve responder às seguintes questões:

- As informações são efetivamente precisas?
- As informações são concisas e direcionadas ao assunto?
- É fácil para o usuário entender o layout do objeto de conteúdo?
- As informações contidas em um objeto de conteúdo podem ser encontradas facilmente?
- Foram fornecidas referências apropriadas para todas as informações derivadas de outras fontes?
- As informações apresentadas são consistentes internamente e consistentes com as informações apresentadas em outros objetos de conteúdo?
- O conteúdo é ofensivo, confuso ou dá margem a litígio?
- O conteúdo desrespeita os direitos autorais existentes ou de marcas registradas?
- O conteúdo contém links que complementam o conteúdo existente? Os links estão corretos?
- O estilo estético do conteúdo está em conflito com o estilo estético da interface?

Obter respostas para cada uma dessas perguntas para uma grande WebApp (contendo centenas de objetos de conteúdo) pode ser uma tarefa assustadora. No entanto, se não forem descobertos os erros de semântica, será abalada a confiança do usuário na WebApp e isso pode levar ao fracasso da aplicação.

Objetos de conteúdo existem em uma arquitetura que tem um estilo específico (Capítulo 13). Durante o teste de conteúdo, a estrutura e a organização da arquitetura de conteúdo são verificadas para assegurar que o conteúdo necessário seja apresentado ao usuário final na ordem e relacionamentos apropriados. Por exemplo, a WebApp **CasaSeguraGarantida.com** apresenta uma variedade de informações sobre sensores usados como parte dos produtos de segurança e vigilância. Objetos de conteúdo fornecem informações descritivas, especificações técnicas, uma representação fotográfica e informações relacionadas. Os testes da arquitetura de conteúdo de **CasaSeguraGarantida.com** procuram descobrir erros na apresentação dessas informações (por exemplo, uma descrição do Sensor X é apresentada com uma foto do Sensor Y).

PONTO-CHAVE

Os objetivos do teste de conteúdo são: (1) descobrir erros de sintaxe no conteúdo, (2) descobrir erros de semântica e (3) encontrar erros estruturais.



Quais questões deverão ser formuladas e respondidas para descobrir erros de semântica no conteúdo?

“Em geral, as técnicas de teste de software usadas em outras aplicações são as mesmas que as utilizadas em aplicações baseadas na Web... A diferença é que a tecnologia varia no ambiente Web.”

Hung Nguyen

20.3.2 Teste de base de dados

As WebApps modernas fazem muito mais do que apresentar objetos de conteúdo estáticos. Em muitos domínios de aplicação, interfaceiam com sistemas sofisticados de gerenciamento de banco de dados e criam objetos de conteúdo dinâmico em tempo real usando os dados adquiridos de um banco de dados.

Por exemplo, uma WebApp de serviços financeiros pode produzir informações complexas baseadas em texto, na forma tabular e gráfica sobre um fundo específico (por exemplo, um fundo de ações ou fundo mútuo). O objeto de conteúdo composto que apresenta essas informações é criado dinamicamente quando o usuário solicita informações sobre um fundo específico. Para tanto, são necessários os seguintes passos: (1) é consultado um grande banco de dados de fundos, (2) são extraídos os dados relevantes do banco de dados, (3) os dados extraídos devem ser organizados como um objeto de conteúdo e (4) esse objeto de conteúdo (representando informações personalizadas requisitadas por um usuário final) é transmitido para o ambiente do cliente para ser apresentado. Erros podem ocorrer e efetivamente ocorrem, em consequência de cada uma dessas etapas. O objetivo do teste de banco de dados é descobrir os erros, mas esse tipo de teste é complicado por uma variedade de fatores:

 Quais os tópicos que complicam o teste de banco de dados para WebApps?

1. *A solicitação original de informações do lado do cliente raramente é apresentada de maneira (por exemplo, linguagem de consulta estruturada, Structured Query Language – SQL) que possa ser colocada em um sistema de gerenciamento de banco de dados (database management system – DBMS).* Portanto, deverão ser projetados testes para descobrir erros de tradução da solicitação do usuário em uma forma que possa ser processada pelo DBMS.
2. *O banco de dados pode ser remoto ao servidor que abriga a WebApp.* Portanto, devem ser desenvolvidos testes que descubram erros na comunicação entre a WebApp e o banco de dados remoto.³
3. *Dados brutos adquiridos do banco de dados devem ser transmitidos para o servidor da WebApp e formatados corretamente para subsequente transmissão ao cliente.* Portanto, devem ser criados testes que demonstrem a validade dos dados brutos recebidos pelo servidor da WebApp, e devem ser criados também testes adicionais que demonstrem a validade das transformações aplicadas a esses dados para criar objetos de conteúdo válidos.
4. *O(s) objeto(s) de conteúdo dinâmico deve(m) ser transmitido(s) ao cliente de maneira que possa(m) ser apresentado(s) ao usuário final.* Portanto, uma série de testes deverá ser projetada para (1) descobrir erros no formato do objeto de conteúdo e (2) testar compatibilidade com diferentes configurações de ambiente.

“... nós não confiamos em um site que sofre de frequentes paradas, trava no meio de uma transação ou tem pouco senso de utilidade. O teste, portanto, tem um papel crucial no processo geral de desenvolvimento.”

Wing Lam

Considerando esses quatro fatores, os métodos de projeto de casos de teste deverão ser aplicados a cada uma das “camadas de interação” [Ngu01] observadas na Figura 20.2. O teste deve assegurar que (1) sejam passadas informações válidas entre o cliente e o servidor por meio da camada de interface, (2) que a WebApp processa scripts corretamente e extrai ou formata adequadamente os dados do usuário, (3) que os dados do usuário são passados corretamente para uma função de transformação de dados no lado servidor que formata apropriadamente as consultas (por exemplo, SQL), (4) que as consultas são passadas para uma camada de gerenciamento de dados⁴ que se comunica com as rotinas de acesso a banco de dados (potencialmente localizadas em outras máquinas).

As camadas de transformação de dados, gerenciamento de dados e acesso a banco de dados da Figura 20.2 muitas vezes são construídas com componentes reutilizáveis que foram validados separadamente e como um pacote. Se for esse o caso, o teste da WebApp focaliza o projeto de casos de testes para experimentar as interações entre a camada cliente e as duas primeiras camadas servidor (WebApp e transformação de dados) mostradas na figura.

³ Esses testes podem se tornar complexos quando são encontradas bases de dados distribuídas ou quando é necessário acesso a um armazém de dados (Capítulo 1).

⁴ A camada de gerenciamento de dados tipicamente incorpora uma interface SQL em nível de chamada (SQL-CLI) como, por exemplo, o Microsoft OLE/ADO ou Java Database Connectivity (JDBC).

A camada da interface de usuário é testada para garantir que os scripts sejam construídos corretamente para cada consulta de usuário e adequadamente transmitidos para o lado servidor. A camada da WebApp no lado do servidor é testada para assegurar que os dados do usuário sejam adequadamente extraídos dos scripts e transmitidos corretamente para a camada de transformação de dados no lado do servidor. As funções de transformação de dados são testadas para assegurar que o SQL correto seja criado e passado para os componentes apropriados de gerenciamento de dados.

Uma discussão detalhada da tecnologia subjacente que deve ser entendida para projetar adequadamente esses testes de banco de dados está além do escopo deste livro. Para mais detalhes, veja [Sce02], [Ngu01] e [Bro01].

20.4 TESTE DA INTERFACE DE USUÁRIO



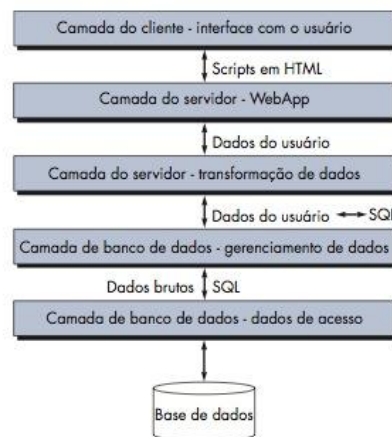
AVISO
Com exceção de aspectos específicos orientados para a WebApp, a estratégia de interface observada aqui é aplicável a todos os tipos de software cliente-servidor.

A verificação e a validação de uma interface de usuário de uma WebApp ocorre em três pontos distintos. Durante a análise, o modelo de interface é revisado para assegurar que esteja em conformidade com os requisitos dos interessados e com outros elementos do modelo de requisitos. Durante o projeto, o modelo de projeto de interface é revisado para assegurar que critérios genéricos de qualidade estabelecidos para todas as interfaces de usuário (Capítulo 11) tenham sido satisfeitos e que os problemas de projeto de interface específicos da aplicação tenham sido corretamente resolvidos. Durante o teste, o foco passa para a execução de aspectos da interação com o usuário específico da aplicação à medida que são manifestados pela sintaxe e semântica da interface. Além disso, o teste fornece uma avaliação final de usabilidade.

20.4.1 Estratégia de teste de interface

O teste de interface experimenta mecanismos de interação e valida aspectos estéticos da interface de usuário. A estratégia geral é (1) descobrir erros relacionados com mecanismos específicos de interface (por exemplo, erros na execução apropriada de um link de menu ou na maneira como os dados são colocados em um formulário) e (2) descobrir erros na maneira como a interface implementa as semânticas de navegação, funcionalidade da WebApp ou exibição de conteúdo. Para essa estratégia, há uma série de objetivos a ser atingidos:

FIGURA 20.2
Camadas de interação



- *Características de interface são testadas para garantir que as regras de projeto, estética e conteúdo visual relacionado estejam disponíveis para o usuário sem erro.* As características incluem fontes, o uso da cor, molduras, imagens, bordas, tabelas e características de interface relacionadas que são geradas quando ocorre a execução da WebApp.
- *Mecanismos individuais de interface são testados de uma maneira análoga ao teste de unidade.* Por exemplo, são projetados testes para experimentar todos os formulários, scripts no lado do cliente, HTML dinâmico, scripts, conteúdo concatenado e mecanismos de interface específicos da aplicação (por exemplo, um carrinho de compras para uma aplicação de e-commerce). Em muitos casos, o teste pode concentrar-se exclusivamente em um desses mecanismos (a “unidade”) excluindo todas as outras características e funções de interface.
- *Cada mecanismo de interface é testado de acordo com o contexto de um caso de uso ou uma unidade semântica de navegação, NSU, (Capítulo 13) para uma categoria específica de usuário.* Essa abordagem é análoga ao teste de integração porque os testes são executados à medida que os mecanismos de interface são integrados para permitir que um caso de uso ou uma NSU sejam executados.
- *A interface completa é testada em relação aos casos de uso selecionados e NSUs para descobrir erros nas semânticas da interface.* Essa abordagem é análoga ao teste de validação porque a finalidade é demonstrar conformidade com semânticas específicas de casos de uso ou de NSU. É nesse estágio que uma série de testes de usabilidade é executada.
- *A interface é testada segundo uma variedade de ambientes (por exemplo, navegadores) para assegurar que sejam compatíveis.* Na realidade, essas séries de testes podem ser consideradas como parte do teste de configuração.



O teste de links externos deve ocorrer durante toda a vida da WebApp. Como parte de estratégia de manutenção, testes de links devem ser agendados regularmente.

20.4.2 Testando mecanismos de interface

Quando um usuário interage com uma WebApp, a interação ocorre através de um ou mais mecanismos de interface. Uma rápida revisão das considerações para cada mecanismo de interface é apresentada nos próximos parágrafos [Spl01].

Links. Cada link de navegação é testado para assegurar que o objeto de conteúdo ou a função apropriados sejam acessados⁵. Faz-se uma lista de todos os links associados com o layout da interface (por exemplo, barras de menu, itens de índice) e então se executa cada um individualmente. Além disso, os links em cada objeto de conteúdo devem ser experimentados para descobrir as URLs incorretas ou links para objetos de conteúdo ou funções impróprios. Por fim, links para WebApps externas deverão ser testados quanto à exatidão e avaliados para determinar o risco de se tornarem inválidos com o tempo.

Formulários. Em nível macroscópico, são executados testes para assegurar que (1) rótulos identifiquem corretamente os campos no formulário e que os campos obrigatórios sejam identificados visualmente para o usuário, (2) o servidor receba todas as informações contidas no formulário e que não seja perdido nenhum dado na transmissão entre o cliente e o servidor, (3) sejam usadas as escolhas-padrão (*defaults*) apropriadas quando o usuário não seleciona de um menu pull down ou por meio de uma série de botões, (4) funções do navegador (por exemplo, a seta de retorno) não corrompam os dados introduzidos em um formulário e (5) scripts que realizam verificação de erros sobre dados introduzidos funcionem adequadamente e proporcionem mensagens de erros coerentes.

Em um nível mais específico, os testes devem assegurar que (1) os campos do formulário tenham tamanho e tipos de dados corretos, (2) o formulário estabeleça proteções apropriadas que impedem que o usuário digite cadeias de texto mais longas do que um comprimento máximo predefinido, (3) todas as opções apropriadas para menus pull-down sejam especificadas e ordenadas de maneira que tenha significado para o usuário final, (4) os recursos de “preenchimento

⁵ Esses testes podem ser realizados tanto como parte do teste de interface quanto do de navegação.



Os testes de script no lado do cliente e testes associados com HTML dinâmico deverão ser repetidos sempre que uma nova versão de navegador popular é lançada.

automático” do navegador não causem erros nas entradas de dados e (5) a tecla tab (ou alguma outra tecla) faça a movimentação apropriada entre os campos do formulário.

Script no lado do cliente. Os testes caixa-preta são realizados para descobrir quaisquer erros no processamento quando o script é executado. Eles muitas vezes são acoplados ao teste de formulário, porque a entrada de script é em geral derivada de dados fornecidos como parte de processamento de formulários. Deverá ser feito um teste de compatibilidade para assegurar que a linguagem de script escolhida funcionará corretamente nas configurações ambientais que suportam a WebApp. Além disso, para testar o próprio script, Splaine e Jaskiel [Spl01] sugerem que “você deve assegurar que os padrões de sua empresa [WebApp] definam a linguagem preferida e a versão de linguagem de script a ser usada para o lado do cliente (e o lado do servidor)”.

HTML dinâmico. Cada página Web que contenha HTML dinâmico é executada para assegurar que a exibição dinâmica esteja correta. Além disso, deverá ser feito um teste de compatibilidade para garantir que o HTML funcione corretamente nas configurações ambientais que suportam a WebApp.

Janelas pop-up. Uma série de testes garante que (1) o pop-up esteja dimensionado e posicionado corretamente, (2) o pop-up não cubra a janela original da WebApp, (3) o projeto estético do pop-up seja consistente com o da interface e (4) barras de rolagem e outros mecanismos de controle acrescentados ao pop-up estejam corretamente localizados e funcionem conforme desejado.

Scripts CGI. São feitos testes caixa-preta com ênfase na integridade dos dados (quando os dados são passados para o script CGI) e processamento de script (uma vez que dados validados tenham sido recebidos). Além disso, pode ser feito teste de desempenho para assegurar que a configuração do lado servidor possa acomodar as demandas de processamento de múltiplas chamadas de scripts CGI [Spl01].

Conteúdo encadeado (streaming). Os testes devem demonstrar que os dados encadeados são atualizados, exibidos corretamente e que podem ser suspensos sem erro e reiniciados sem dificuldade.

Cookies. São necessários testes do lado do servidor e do lado do cliente. No lado do servidor, deverão assegurar que um cookie esteja corretamente construído (contém dados corretos) e corretamente transmitido para o lado do cliente, quando solicitado um conteúdo ou uma funcionalidade específica. Além disso, a persistência apropriada do cookie é testada para assegurar que sua data de expiração esteja correta. Do lado do cliente, testes determinam se a WebApp anexa corretamente os cookies existentes de acordo com a solicitação específica (enviada ao servidor).

Mecanismos de interface específicos de aplicativo. Os testes seguem uma checklist de funcionalidade e características definidas pelo mecanismo de interface. Por exemplo, Splaine e Jaskiel [Spl01] sugerem a seguinte checklist para funcionalidade do carrinho de compras definido para uma aplicação de e-commerce:

- Faça o teste de fronteiras (Capítulo 18) do número mínimo e máximo de itens que podem ser colocados no carrinho de compras.
- Teste uma solicitação de “saída” para um carrinho de compras vazio.
- Teste a remoção apropriada de um item do carrinho de compras.
- Teste para determinar se uma compra esvazia todo o conteúdo de um carrinho de compras.
- Teste para determinar a persistência do conteúdo do carrinho de compras (isso deverá ser especificado como parte dos requisitos do cliente).
- Teste para determinar se a WebApp pode restaurar o conteúdo do carrinho de compras em alguma data futura (supondo que não foi feita nenhuma compra).

20.4.3 Testando semânticas de interface

Uma vez que cada mecanismo de interface foi testado em termos de “unidade”, o foco muda para as semânticas de interface. O teste de semânticas de interface “avalia quão bem o projeto se preocupa com os usuários, oferece diretrizes claras, fornece realimentação e mantém consistência de linguagem e abordagem” [Ngu00].

Uma revisão rigorosa do modelo de projeto de interface pode proporcionar respostas parciais às questões do parágrafo anterior. No entanto, cada cenário de caso de uso (para cada categoria de usuário) deverá ser testado uma vez implementada a WebApp. Essencialmente, um caso de uso torna-se a entrada para o projeto de uma sequência de testes. A finalidade da sequência de testes é descobrir erros que impedirão o usuário de atingir o objetivo associado ao caso de uso.

À medida que cada caso de uso é testado, é aconselhável manter uma checklist para garantir que todos os itens do menu foram usados pelo menos uma vez e que todos os links contidos em um objeto de conteúdo tenham sido utilizados. Além disso, a série de testes deve incluir seleção de opções de menu e uso de link inadequados. A finalidade é determinar se a WebApp proporciona manipulação eficaz do erro e recuperação.

20.4.4 Testes de usabilidade

Os testes de usabilidade são similares aos de semânticas de interface (Seção 20.4.3), porque também avaliam o grau com o qual os usuários podem interagir efetivamente com a WebApp e o grau em que a WebApp dirige as ações do usuário, proporciona uma boa realimentação e reforça uma abordagem de interação consistente. Em vez de concentrar-se intencionalmente nas semânticas de algum objetivo interativo, as revisões e testes de usabilidade são projetadas para determinar o grau com o qual a interface da WebApp facilita a vida do usuário.⁶

Invariavelmente o projetista contribuirá para o projeto dos testes de usabilidade, mas eles serão feitos pelos usuários finais. É aplicada a seguinte sequência de passos [Spl01]:

1. Definir uma série de categorias de teste de usabilidade e identificar objetivos para cada uma delas.
2. Projetar testes que permitirão avaliar cada um dos objetivos.
3. Selecionar os participantes que farão os testes.
4. Instrumentar a interação dos participantes com a WebApp enquanto o teste está em execução.
5. Desenvolver um mecanismo para avaliar a usabilidade da WebApp.

O teste de usabilidade pode ocorrer em diferentes níveis de abstração: (1) pode ser investigada a usabilidade de um mecanismo específico de interface (por exemplo, um formulário), (2) pode ser investigada a usabilidade de uma página Web completa (abrangendo mecanismos de interface, objetos de dados e funções relacionadas) ou (3) pode ser considerada a usabilidade da WebApp completa.

O primeiro passo no teste de utilidade é identificar uma série de categorias de usabilidade e estabelecer objetivos para cada uma das categorias. As seguintes categorias de teste e objetivos (escritos na forma de perguntas) ilustram tal abordagem:⁷

Interatividade — Os mecanismos de interação (por exemplo, menus desdobráveis, botões, ponteiros) são fáceis de entender e usar?

Layout — Os mecanismos de navegação, conteúdo e funções são colocados de maneira que permita ao usuário encontrá-los rapidamente?

WebRef
Um guia conveniente para o teste de usabilidade pode ser encontrado em www.ohref.com/guides/design/199806/0615jof.html.

? Quais características da usabilidade tornam-se o foco do teste e quais objetivos específicos são considerados?

⁶ O termo amigável ao usuário tem sido usado nesse contexto. O problema, naturalmente, é que a percepção de um usuário sobre uma interface “amigável” pode ser radicalmente diferente da de outro usuário.

⁷ Para mais informações sobre usabilidade, veja o Capítulo 11.

Clareza — O texto é bem escrito e fácil de ser entendido?⁸ As representações gráficas são fáceis de entender?

Estética — O layout, a cor, o tipo de letra e características relacionadas facilitam o uso? Os usuários se sentem “confortáveis” com a aparência e comportamento da WebApp?

Características da tela — A WebApp otimiza o uso do tamanho da tela e da resolução?

Sensibilidade ao tempo — Características importantes, funções e conteúdo podem ser usados ou acessados no tempo correto?

Personalização — A WebApp se adapta a necessidades específicas de diferentes categorias de usuário ou de usuários individuais?

Acessibilidade — A WebApp é acessível a pessoas com necessidades especiais?

Em cada uma dessas categorias é projetada uma série de testes. Em alguns casos, o “teste” pode ser uma revisão visual de uma página Web. Em outros, podem novamente ser executados testes de semânticas de interface, mas nesse caso os problemas de usabilidade são essenciais.

Como exemplo, consideramos a avaliação da usabilidade para mecanismos de interação e interface. Constantine e Lockwood [Con99] sugerem que a seguinte lista de características de interface deveria ser revista e testada quanto à usabilidade: animação, botões, cores, controle, diálogo, campos, formulários, molduras, gráficos, rótulos, links, menus, mensagens, páginas de navegação, seletores, texto e barras de ferramentas. À medida que cada característica é avaliada, ela é classificada em uma escala qualitativa pelos usuários que estão fazendo o teste. A Figura 20.3 mostra um conjunto possível de “graus” de avaliação que podem ser selecionados pelos usuários. Esses graus são aplicados a cada característica individualmente, para uma página Web completa, ou para a WebApp como um todo.

PONTO-CHAVE

As WebApps executam em uma variedade de ambientes do lado do cliente. O objetivo dos testes de compatibilidade é descobrir erros associados a um ambiente específico (por exemplo, navegador).

20.4.5 Testes de compatibilidade

Diferentes computadores, dispositivos de imagem, sistemas operacionais, navegadores e velocidades de conexão de rede podem ter influência significativa na operação da WebApp. Cada configuração de computador pode resultar em diferenças nas velocidades de processamento no lado do cliente, resoluções de tela e velocidades de conexão. Excentricidades de sistemas operacionais podem causar problemas no processamento da WebApp. Diferentes navegadores às vezes produzem resultados ligeiramente diferentes, independentemente do grau de padronização HTML na WebApp. Os plug-ins necessários podem ou não estar disponíveis para uma determinada configuração.

Em alguns casos, pequenos problemas de compatibilidade não apresentam dificuldades significativas, mas, em outros, podem ser encontrados erros graves. Por exemplo, velocidades de download podem se tornar inaceitáveis, a falta de um plug-in necessário pode tornar o conteúdo indisponível, diferenças entre navegadores podem mudar significativamente o layout da página, estilos de fontes podem ser alterados tornando o texto ilegível ou formulários podem ser organizados de forma inadequada. O teste de compatibilidade procura descobrir esses problemas antes que a WebApp entre no ar (fique on-line).

O primeiro passo no teste de compatibilidade é definir uma série de configurações de computadores “comumente encontradas” no lado do cliente e suas variantes. Essencialmente, é criada uma estrutura em árvore, identificando cada plataforma de computador, dispositivos típicos de imagem, sistemas operacionais suportados na plataforma, navegadores disponíveis, velocidades prováveis de conexão à internet e informações similares. Em seguida, é derivada uma série de testes de validação de compatibilidade, muitas vezes adaptados de testes de interfaces existentes, testes de navegação, testes de desempenho e testes de segurança. A finalidade desses testes é descobrir erros ou problemas de execução que podem ser atribuídos a diferenças em configuração.

⁸ O FOG Readability Index e outros pode ser usado para dar uma visão quantitativa da clareza. Veja mais detalhes em <http://developer.gnome.org/documents/usability/usability-reability.html>.

FIGURA 20.3

Avaliação qualitativa da usabilidade



20.5 TESTE NO NÍVEL DE COMPONENTE

O teste no nível de componente, também chamado de teste de função, concentra-se em um conjunto de testes que tentam descobrir erros nas funções da WebApp. Cada função da WebApp é um componente de software (implementado dentre uma variedade de linguagens de programação ou scripts) e pode ser testado por meio de técnicas de caixa-preta (e em alguns casos, caixa-branca) discutidas no Capítulo 18.

CASASEGURA



Teste de WebApp

Cena: Escritório de Doug Miller.

Personagens: Doug Miller (gerente do grupo de engenharia de software CasaSegura) e Vinod Raman (membro da equipe de engenharia de software do produto).

Conversa:

Doug: O que você acha da WebApp de e-commerce CasaSeguraGarantida.com V.0.0?

Vinod: O prestador de serviço do fornecedor fez um bom trabalho. Sharon [gerente de desenvolvimento do fornecedor] disse-me que estão testando conforme conversamos.

Doug: Gostaria que você e os demais membros da equipe fizessem um pequeno teste informal no site de e-commerce.

Vinod (fazendo careta): Pensei que iríamos contratar uma empresa de teste de terceiros para validar a WebApp. Ainda estamos tentando liberar esse artefato.

Doug: Vamos contratar um fornecedor para testar o desempenho e segurança, e nosso fornecedor já está testando. Apenas pensei que outro ponto de vista seria útil e, além disso, precisamos manter os custos dentro dos limites, portanto...

Vinod (suspirando): O que você está procurando?

Doug: Quero ter certeza de que a interface e toda a navegação estão sólidas.

Vinod: Suponho que podemos começar com os casos de uso para cada uma das principais funções de interface:

Conheça o CasaSegura.

Especifique o sistema CasaSegura de que você precisa.

Compre um sistema CasaSegura.

Obtenha suporte técnico.

Doug: Bom. Mas percorra os caminhos de navegação até o fim.

Vinod (examinando um caderno de casos de uso): Sim, quando você seleciona **Especifique o sistema CasaSegura de que você precisa**, isso vai levá-lo a:

Selecione componentes do CasaSegura e Obtenha recomendações de componente do CasaSegura.

Podemos exercitar as semânticas de cada caminho.

Doug: Aproveite para verificar o conteúdo que aparece em cada nó de navegação.

Vinod: Claro... E os elementos funcionais também. Quem está testando a usabilidade?

Doug: Humm... O fornecedor de teste coordenará o teste de usabilidade. Nós contratamos uma empresa de pesquisa de mercado para selecionar 20 usuários típicos para o estudo de usabilidade, mas se vocês descobrirem quaisquer problemas de usabilidade...

Vinod: Eu sei, passamos para eles.

Doug: Obrigado, Vinod.

Casos de testes em nível de componente muitas vezes são controlados por entrada no nível de formulários. Uma vez definidos os dados dos formulários, o usuário seleciona um botão ou outro mecanismo de controle para iniciar a execução. Os seguintes métodos de projeto de casos de teste (Capítulo 18) são típicos:

- *Particionamento de equivalência* – O domínio de entrada da função é dividido em categorias de entrada ou classes a partir das quais são derivados os casos de testes.
- O formulário de entrada é analisado para determinar que classes de dados são relevantes para a função. Criam-se e executam-se casos de testes para cada classe de entrada, enquanto outras classes de entrada são mantidas constantes. Por exemplo, uma aplicação de e-commerce pode implementar uma função que calcula as despesas de envio. Entre uma variedade de informações de envio fornecidas através do formulário está o código postal do usuário. São projetados casos de testes para tentar-se descobrir erros no processamento do código postal especificando valores de código postal que podem revelar diversas classes de erros (por exemplo, um código postal incompleto, um código postal correto, um código postal não existente, um formato errôneo de código postal).
- *Análise de valor-limite* – Os dados de formulários são testados nos seus limites. Por exemplo, a função que calcula o frete que vimos anteriormente solicita o número máximo de dias necessários para a entrega do produto. No formulário consta um mínimo de 2 dias e um máximo de 14. Porém, o teste de valor-limite pode entrar com valores de 0, 1, 2, 13, 14 e 15 para determinar como a função reage a dados dentro ou fora dos limites de entrada válida.⁹
- *Teste de caminho* – Se a complexidade lógica da função for alta,¹⁰ pode ser usado o teste de caminho (um método de projeto de caso de teste caixa-branca) para assegurar que todos os caminhos independentes no programa foram experimentados.

Além desses métodos de projeto de casos de teste, uma técnica chamada *teste de erro forçado* [Ngu01] é usada para gerar casos de testes que propositalmente conduzem o componente da WebApp para uma condição de erro. A finalidade é descobrir erros que ocorrem durante a manipulação de erro (por exemplo, mensagens de erro incorretas ou não existentes, falha da WebApp como consequência do erro, saída errônea causada por entrada errônea, efeitos colaterais relacionados com o processamento do componente).

Cada caso de teste no nível de componente especifica todos os valores de entrada e a saída esperada a ser fornecida pelo componente. A saída real produzida decorrente do teste é registrada para referência futura durante o suporte e a manutenção.

Em muitas situações, a execução correta de uma função da WebApp está ligada ao interfacimento correto com um banco de dados que pode ser externo a WebApp. Portanto, o teste de banco de dados torna-se parte integral do regime de teste de componente.

20.6 TESTES DE NAVEGAÇÃO

Um usuário navega por uma WebApp de maneira muito semelhante a um visitante que caminha por uma loja ou museu. Podem ser trilhados muitos caminhos, podem ser feitas muitas paradas, muitas coisas a aprender e observar, atividades a iniciar e decisões a tomar. Esse processo de navegação é previsível no sentido de que cada visitante tem uma série de objetivos quando chega. Ao mesmo tempo, o processo de navegação pode ser imprevisível porque o visitante, influenciado por alguma coisa que vê ou aprende, pode escolher um caminho ou iniciar uma ação que não é típica para o objetivo original. A tarefa do teste de navegação é (1) garantir

⁹ Nesse caso, um projeto melhor da entrada pode eliminar erros potenciais. O número máximo de dias poderia ser selecionado de um menu pull-down, impedindo o usuário de especificar entrada fora dos limites.

¹⁰ A complexidade lógica pode ser determinada computando a complexidade ciclômica do algoritmo. Veja detalhes adicionais no Capítulo 18.

que os mecanismos que permitem ao usuário navegar através da WebApp estejam todos em funcionamento e (2) confirmar que cada unidade semântica de navegação (*navigation semantic unit* — NSU) possa ser alcançada pela categoria apropriada de usuário.

20.6.1 Testando a sintaxe de navegação

A primeira fase do teste de navegação realmente começa durante o teste da interface. Os mecanismos de navegação são verificados para garantir que cada um execute sua função planejada. Splaine e Jaskiel [Spl01] sugerem os seguintes mecanismos de navegação a ser testados:

- *Links de navegação* — esses mecanismos incluem links internos dentro da WebApp, links externos para outras WebApps e âncoras dentro de uma página Web específica. Cada link deverá ser testado para assegurar que o conteúdo ou funcionalidade apropriada sejam alcançados quando o link é escolhido.
- *Redirecionamentos* — esses links entram em cena quando um usuário solicita um URL não existente ou seleciona um link cujo conteúdo foi removido ou o nome mudou. É exibida uma mensagem para o usuário, e a navegação é redirecionada para outra página (por exemplo, a página principal). Os redirecionamentos deverão ser testados solicitando-se links internos ou URLs externos incorretos e avaliando como a WebApp lida com essas solicitações.
- *Marcadores de páginas (Bookmarks)* — embora os marcadores sejam uma função do navegador, a WebApp deverá ser testada para assegurar que possa ser extraído um título de página com significado quando o marcador for criado.
- *Molduras e conjunto de molduras (Frames e framesets)* — cada moldura contém o conteúdo de uma página Web específica, e um conjunto de molduras contém múltiplas molduras e permite mostrar múltiplas páginas Web ao mesmo tempo. Por ser possível aninhar molduras e conjuntos de molduras um dentro do outro, esses mecanismos de navegação e visualização deverão ser testados quanto ao correto conteúdo, layout e dimensionamento apropriados, desempenho de download e compatibilidade de navegador.
- *Mapas de site* — o mapa de site fornece uma tabela completa de conteúdo para todas as páginas Web. Cada entrada do mapa de site deverá ser testada para garantir que os links levem o usuário ao conteúdo ou funcionalidade apropriados.
- *Dispositivos de busca interna* — WebApps complexas muitas vezes contêm centenas ou até milhares de objetos de conteúdo. Um dispositivo de busca interna permite ao usuário executar uma busca por palavra-chave dentro da WebApp para encontrar o conteúdo desejado. O teste de dispositivos de busca valida a precisão e totalidade da busca, as propriedades de manipulação de erro do dispositivo de busca e recursos avançados de busca (por exemplo, o uso de operadores booleanos no campo de busca).

Alguns dos testes citados podem ser executados por ferramentas automáticas (por exemplo, verificação de link), enquanto outros são projetados e executados manualmente. O objetivo geral é garantir que os erros em mecanismos de navegação sejam encontrados antes que a WebApp entre no ar.

20.6.2 Testando as semânticas de navegação

No Capítulo 13 a unidade semântica de navegação (NSU) é definida como “uma série de informações e estruturas de navegação relacionadas que colabora no atendimento a um subconjunto de requisitos de usuário relacionados” [Cac02]. Cada NSU é definida por um conjunto de caminhos de navegação (chamados “modos de navegação”) que conectam nós de navegação (por exemplo, páginas Web, objetos de conteúdo ou funcionalidade). Considerada como um todo, cada NSU permite ao usuário satisfazer requisitos específicos definidos por um ou mais casos de uso para uma categoria de usuário. O teste de navegação exercita cada NSU para asse-

“Não estamos perdidos. Apenas mudamos a localização.”
John M. Ford

? Quais perguntas devem ser feitas e respondidas à medida que cada NSU é testada?

gurar que esses requisitos possam ser atendidos. Você deve responder às seguintes perguntas à medida que é testada cada NSU:

- A NSU é atendida em sua totalidade sem erro?
- Cada nó de navegação (definido por uma NSU) é acessível no contexto dos caminhos de navegação definidos para a NSU?
- Se a NSU pode ser atendida usando mais de um caminho de navegação, todos os caminhos relevantes foram testados?
- Se forem fornecidas instruções pela interface de usuário para ajudar na navegação, as instruções são corretas e inteligíveis à medida que a navegação ocorre?
- Existe algum mecanismo (que não seja a seta “de retorno” do navegador) para voltar a um nó de navegação anterior e ao início do caminho de navegação?
- Os mecanismos de navegação em um nó grande de navegação (isto é, uma longa página Web) funcionam corretamente?
- Se uma função deve ser executada em um nó e o usuário opta por não fornecer entrada, o restante da NSU pode ser completado?
- Se uma função é executada em um nó e ocorre um erro no processamento da função, a NSU pode ser completada?
- Há uma maneira de interromper a navegação antes que todos os nós tenham sido alcançados, mas depois retornar ao ponto onde a navegação foi interrompida e continuar a partir dali?
- Todos os nós podem ser acessados do mapa do site? Os nomes dos nós têm significado para os usuários finais?
- Se um nó em uma NSU é alcançado de uma origem externa, é possível processar o próximo nó no caminho de navegação? É possível retornar ao nó anterior no caminho de navegação?
- O usuário entende sua localização dentro da arquitetura de conteúdo à medida que a NSU é executada?




Se não foram criadas NSUs como parte da análise ou projeto de WebApp, você pode aplicar coisas de uso para o projeto de casos de testes de navegação. O mesmo conjunto de perguntas é proposto e respondido.

O teste de navegação, bem como o teste de interface e de usabilidade, deverá ser feito por diferentes clientes quando possível. Os engenheiros da Web têm a responsabilidade pelos primeiros estágios do teste de navegação, mas os estágios posteriores deverão ser testados por outros interessados, do projeto, uma equipe de teste independente e, por último, por usuários não técnicos. O objetivo é exercitar a navegação da WebApp completamente.

20.7 TESTE DE CONFIGURAÇÃO

A variabilidade e a instabilidade da configuração são fatores importantes que tornam o teste da WebApp um desafio. Hardware, sistema operacional, navegadores, capacidade de armazenamento, velocidades de comunicação de rede e uma variedade de outros fatores do lado do cliente são difíceis de prever para cada usuário. Além disso, a configuração para um usuário pode mudar regularmente [por exemplo, atualização do sistema operacional, novo provedor e novas velocidades de conexão]. O resultado pode ser um ambiente do lado do cliente, sujeito a erros sutis e significativos. A impressão de um usuário sobre a WebApp e a maneira pela qual ele interage com ela pode ser significativamente diferente da experiência de um outro usuário, se ambos não estiverem trabalhando na mesma configuração do cliente.

O objetivo do teste de configuração não é exercitar todas as configurações possíveis no lado do cliente. Em vez disso, é testar um conjunto de prováveis configurações do cliente e do servidor para assegurar que a experiência do usuário será a mesma em todos os casos e isolar erros que podem ser específicos a uma determinada configuração.

 **Quais perguntas devem ser feitas e respondidas à medida que é feito o teste de configuração no lado do servidor?**

20.7.1 Tópicos no lado do servidor

No lado do servidor, os casos de testes de configuração são projetados para verificar se a configuração do servidor [isto é, servidor WebApp, servidor de banco de dados, sistema operacional, software de firewall, aplicações concorrentes] pode suportar a WebApp sem erro. Essencialmente, a WebApp é instalada em um ambiente servidor e testado para garantir que ele opere sem erro.

Quando são projetados os testes de configuração no lado do servidor, você deve considerar cada componente da configuração. Entre as perguntas a ser feitas e respondidas durante o teste de configuração no lado do servidor destacam-se as seguintes:

- A WebApp é totalmente compatível com o sistema operacional do servidor?
- Os arquivos de sistema, diretórios e dados de sistema relacionados são criados corretamente quando a WebApp está operacional?
- As medidas de segurança do sistema (por exemplo, firewalls ou criptografia) permitem que a WebApp seja executada e sirva os usuários sem interferência ou degradação do desempenho?
- A WebApp foi testada com a configuração de servidor distribuído¹¹ (se existir alguma) que foi escolhida?
- A WebApp está adequadamente integrada com o software de banco de dados? A WebApp é sensível a diferentes versões do software de banco de dados?
- Os scripts WebApp do lado do servidor executam corretamente?
- Os erros do administrador do sistema foram examinados quanto ao seu efeito sobre as operações da WebApp?
- Se forem usados servidores substitutos (proxy), as diferenças em sua configuração foram resolvidas com o teste on-site?

20.7.2 Tópicos no lado do cliente

No lado do cliente, o teste de configuração focaliza mais intensamente a compatibilidade da WebApp com configurações que contenham uma ou mais permutações dos seguintes componentes [Ngu01]:

- *Hardware* — CPU, memória, armazenamento e dispositivos de impressão
- *Sistemas operacionais* — Linux, Macintosh OS, Microsoft Windows, sistema operacional móvel
- *Software navegador* — Firefox, Safari, Internet Explorer, Opera, Chrome, e outros
- *Componentes da interface de usuário* — Active X, Java applets e outros
- *Plug-ins* — QuickTime, RealPlayer e muitos outros
- *Conectividade* — cabo, DSL, modem regular, T1, WiFi

Além desses componentes, outras variáveis incluem software de rede, as excentricidades do provedor de serviços (ISP) e aplicações rodando concorrentemente.

Para projetar testes de configuração do cliente, deve-se reduzir o número de variáveis de configuração a um número controlável.¹² Desse modo, cada categoria de usuário é avaliada para determinar as prováveis configurações que serão encontradas. Além disso, os dados de mercado podem ser usados para prever as combinações mais prováveis de componentes. A WebApp é então testada nesses ambientes.

¹¹ Por exemplo, pode ser usado um servidor de aplicação e um servidor de banco de dados separados. A comunicação entre as duas máquinas ocorre por meio de uma conexão em rede.

¹² Rodar testes em todas as combinações possíveis de componentes de configuração é um processo extremamente demorado.

20.8 TESTE DE SEGURANÇA

“A Internet é um lugar arriscado para fazer negócios ou armazenar valores. Invasores (hackers), plagiadores (crackers), xeretas (snoops), enganadores (spoofers)... Vândalos, criadores de vírus e criadores de programas mal intencionados estão à solta.”

Dorothy e Peter Denning



Se a WebApp for crítica para os negócios, contém dados sigilosos ou é um alvo provável para invasores, é aconselhável terceirizar o teste de segurança com um prestador de serviço especializado.

Segurança de WebApp é um assunto complexo que deve ser muito bem entendido antes de se realizar um teste de segurança efetivo.¹³ WebApps e os ambientes cliente e servidor nos quais estão alojadas representam um alvo atraente para invasores (hackers) externos, funcionários insatisfeitos, concorrentes desonestos e qualquer outro que queira roubar informações sigilosas, modificar conteúdo maliciosamente, degradar o desempenho, desabilitar funcionalidade ou atrapalhar uma pessoa, organização ou negócio.

Os testes de segurança são projetados para investigar vulnerabilidades no ambiente do lado do cliente, comunicações de rede que ocorrem quando os dados são passados do cliente para o servidor e vice-versa e no ambiente do lado do servidor. Cada um desses domínios pode ser atacado, e é tarefa do testador de segurança descobrir os pontos fracos que podem ser explorados por aqueles que têm a intenção de fazer isso.

No lado do cliente, as vulnerabilidades podem ser atribuídas muitas vezes a erros preexistentes em navegadores, programas de e-mail ou software de comunicação. Nguyen [Ngu01] descreve uma brecha típica de segurança:

Um dos erros mais mencionados é o transbordamento de buffer (*buffer overflow*), que permite que código mal intencionado seja executado na máquina do cliente. Por exemplo, introduzir em um navegador um URL muito mais longo do que o tamanho de buffer alocado para o URL vai causar um erro de sobrecarga da memória (*buffer overflow*), se o navegador não tiver código de detecção de erro para validar o tamanho da entrada do URL. Um hacker experiente pode explorar esse erro escrevendo um longo URL com código a ser executado que pode fazer o navegador travar ou alterar as configurações de segurança (de alta para baixa), ou, pior ainda, corromper dados do usuário.

Outra potencial vulnerabilidade no lado do cliente é o acesso não autorizado a cookies colocados no navegador. Sites criados com intenções maliciosas podem acessar informações contidas nos cookies legítimos e usá-las para ameaçar a privacidade do usuário, ou pior, preparar o cenário para o roubo de identidade.

Dados transferidos entre cliente e servidor são vulneráveis à enganação (*spoofing*). Essa enganação ocorre quando uma extremidade do elo de comunicação é subvertida por uma entidade com intenções maliciosas. Por exemplo, um usuário pode ser enganado por um site malicioso que age como um legítimo servidor WebApp (aparência e comportamento idênticos). A intenção é roubar senhas, informações confidenciais ou dados de crédito.

No lado do servidor, as vulnerabilidades incluem ataques que causam recusa de serviço e scripts maliciosos que podem ser passados para o lado do cliente ou usados para desabilitar operações do servidor. Além disso, bancos de dados do servidor podem ser acessados sem autorização (roubo de dados).

Para a proteção contra essas vulnerabilidades (e muitas outras), é implementado um ou mais dos seguintes elementos de segurança [Ngu01]:

- **Firewal (bloqueadores contra ataques)** — mecanismo de filtragem que é uma combinação de hardware e software que examina cada pacote de informações que chega para assegurar-se de que ele esteja vindo de uma fonte legítima, bloqueando quaisquer dados suspeitos.
- **Autenticação** — mecanismo de verificação que valida a identidade de todos os clientes e servidores, permitindo que a comunicação ocorra somente quando ambos os lados são verificados.
- **Criptografia** — mecanismo de codificação que protege dados sensíveis, modificando-os de maneira que fique impossível de serem lidos por alguém com más intenções. A cripto-

13 Livros de Cross e Fisher [Cro07], Andrews e Whittaker [And06], e Trivedi [Tri03] fornecem informações úteis sobre esse assunto.

PONTO-CHAVE

Devem ser projetados testes de segurança para experimentar firewalls, autenticação, criptografia e autorização.

grafia é fortalecida pelo uso de *certificados digitais* que permitem ao cliente verificar o destino para o qual os dados são transmitidos.

- **Autorização** — mecanismo de filtragem que permite o acesso ao ambiente do cliente ou do servidor apenas por aqueles indivíduos com códigos de autorização apropriados (por exemplo, ID de usuário e senha).

Os testes de segurança deverão ser projetados para investigar cada uma dessas tecnologias de segurança em esforço para descobrir brechas na segurança.

O projeto real de testes de segurança requer profundo conhecimento do funcionamento interno de cada elemento de segurança e de uma ampla gama de tecnologias de rede. Em muitos casos, o teste de segurança é terceirizado, atribuindo-se a empresas que se especializaram nessas tecnologias.

20.9 TESTE DE DESEMPENHO

Nada é mais frustrante do que uma WebApp que leva muitos minutos para carregar o conteúdo quando sites concorrentes fazem download de conteúdo similar em segundos. Nada é mais desgastante do que tentar entrar em uma WebApp e receber uma mensagem do tipo “servidor ocupado”, com a sugestão para você tentar mais tarde. Nada é mais desconcertante do que uma WebApp que responde instantaneamente em algumas situações e depois parece entrar em um estado de espera infinita em outras. Todas essas ocorrências acontecem diariamente na Web e todas estão relacionadas a desempenho.

O teste de desempenho é usado para descobrir problemas de desempenho que podem resultar da falta de recursos no lado do servidor, largura de banda na rede inadequada, recursos de banco de dados inadequados, recursos deficientes do sistema operacional, funcionalidade da WebApp mal projetada e outros problemas de hardware e software que podem causar degradação de desempenho cliente-servidor. A intenção é dupla: (1) entender como os sistemas respondem quando a *carga* (isto é, número de usuários, número de transações ou volume geral de dados) aumenta e (2) reunir métricas que conduzirão a modificações de projeto para melhorar o desempenho.



Alguns aspectos do desempenho da WebApp, pelo menos como são observados pelo usuário final, são difíceis de testar. A carga da rede, as excentricidades do hardware de interface de rede e problemas similares não são facilmente testados no nível da WebApp.

20.9.1 Objetivos do teste de desempenho

Os testes de desempenho são projetados para simular situações de carga do mundo real. Na medida em que cresce o número de usuários simultâneos da WebApp, ou o número de transações online aumenta, ou a quantidade de dados (download ou upload) aumenta, o teste de desempenho ajudará a responder as seguintes questões:

- O tempo de resposta do servidor degrada a um ponto em que se torna notável e inaceitável?
- Em que ponto (em termos de usuários, transações ou carga de dados) o desempenho se torna inaceitável?
- Que componentes do sistema são responsáveis pela degradação de desempenho?
- Qual o tempo médio de resposta para usuários sob uma variedade de condições de carga?
- A degradação do desempenho tem um impacto sobre a segurança do sistema?
- A confiabilidade ou precisão da WebApp é afetada quando a carga no sistema aumenta?
- O que acontece quando são aplicadas cargas maiores do que a capacidade máxima do servidor?
- A degradação de desempenho tem impacto sobre os lucros da empresa?

Para obter respostas a essas perguntas, são feitos dois testes diferentes de desempenho: (1) *teste de carga* examina cargas reais em uma variedade de níveis e em uma variedade de com-

binações, e (2) *teste de esforço (stress)* força o aumento de carga até o ponto de ruptura para determinar com que capacidade o ambiente WebApp pode lidar. Cada uma dessas estratégias é considerada a seguir.

20.9.2 Teste de carga

A finalidade do teste de carga é determinar como a WebApp e seu ambiente do lado do servidor responderá a várias condições de carga. À medida que é feito o teste, permutações das variáveis a seguir definem uma série de condições de teste:

- N , número de usuários concorrentes
- T , número de transações on-line por usuários por unidade de tempo
- D , carga de dados processados pelo servidor por transação

Em cada caso, as variáveis são definidas de acordo com os limites de operação normal do sistema. Enquanto ocorre cada uma das condições de teste, coletam-se uma ou mais das seguintes medidas: resposta média do usuário, tempo médio para o download de uma unidade padronizada de dados ou tempo médio para processar uma transação. Devem-se examinar essas medidas para determinar se uma diminuição repentina no desempenho pode ser atribuída a uma combinação específica de N , T e D .

O teste de carga também pode ser usado para avaliar velocidade de conexão recomendada para usuários da WebApp. O resultado geral, P , é calculado da seguinte maneira:

$$P = N \times T \times D$$

Como exemplo, considere um site popular de notícias esportivas. Em um momento, 20 mil usuários concorrentes enviam uma solicitação (uma transação, T) a cada 2 minutos em média. Cada transação requer que a WebApp faça o download de um novo artigo que na média tem um tamanho de 3K bytes. Portanto, o resultado pode ser calculado como:

$$\begin{aligned} P &= [20.000 \times 0,5 \times 3\text{Kb}] / 60 = 500 \text{ Kbytes/segundo} \\ &= 4 \text{ megabits/segundo} \end{aligned}$$

A conexão de rede do servidor teria, portanto, de suportar essa taxa de transferência de dados e deveria ser testada para assegurar que fosse capaz disso.

20.9.3 Teste de esforço (stress)

O *teste de esforço* é uma continuação do teste de carga, mas nesse caso as variáveis, N , T e D são forçadas a alcançar e exceder os limites operacionais. A finalidade desses testes é responder às seguintes questões:

- O sistema se degrada “suavemente” ou o servidor desliga quando é excedida a capacidade?
- O software servidor gera mensagens “servidor não disponível”? De uma maneira geral, os usuários ficam cientes de que não podem acessar o servidor?
- O servidor coloca as requisições por recursos em fila e esvazia a fila quando a demanda de capacidade diminui?
- São perdidas transações quando a capacidade é excedida?
- A integridade dos dados é afetada quando a capacidade é excedida?
- Quais valores de N , T e D forçam o ambiente servidor a falhar? Como a falha se manifesta? São mandadas notificações automáticas para o pessoal de suporte técnico no local do servidor?
- Se o sistema falha, quanto tempo demora até que volte a ficar on-line?



Se uma WebApp usa múltiplos servidores para proporcionar uma capacidade significativa, o teste de carga deve ser feito em um ambiente multisservidor.

PONTO-CHAVE

A finalidade do teste de esforço é entender melhor como o sistema falha quando é forçado além de seus limites operacionais.

- Certas funções da WebApp (por exemplo, funcionalidade de computação intensiva, recursos de encadeamento de dados) são interrompidas quando a capacidade atinge um nível de 80% ou 90%?

Uma variação do teste de esforço às vezes é chamada de teste de alternância de pico (*spike/bounce testing*) [Spl01]. Nesse regime de teste, a carga é elevada até a capacidade, depois diminuída rapidamente para as condições normais de operação e, em seguida, elevada novamente. Ao devolver a carga do sistema, você determina quão bem o servidor pode reunir recursos para atender à demanda muito alta e então liberá-los quando as condições normais se restabelecem (de forma que fiquem prontos para o próximo pico).

20.10 RESUMO

O objetivo do teste de WebApp é experimentar cada uma das muitas dimensões da qualidade da WebApp com a finalidade de encontrar erros ou descobrir problemas que podem levar a falhas de qualidade. O teste focaliza o conteúdo, função, estrutura, utilização, navegabilidade, desempenho, compatibilidade, interoperabilidade, capacidade e segurança. Incorpora revisões que ocorrem quando a WebApp é projetada e testes feitos depois que a WebApp é implantada.

A estratégia de teste para WebApp experimenta cada dimensão da qualidade examinando inicialmente “unidades” de conteúdo, funcionalidade ou navegação. Uma vez validadas as unidades individuais, o foco passa para os testes que experimentam a WebApp como um todo. Para tanto, são criados muitos testes sob a perspectiva do usuário e controlados por informações contidas em casos de uso. Um plano de teste para WebApp é desenvolvido e identifica as etapas

FERRAMENTAS DO SOFTWARE**Taxonomia de ferramentas para teste de WebApps**

Em seu artigo sobre teste de sistemas de e-commerce, Lam [Lam01] apresenta uma taxonomia útil de ferramentas automáticas com aplicabilidade direta para teste em um contexto de engenharia para Web. Acrescentamos ferramentas representativas em cada uma das categorias.¹⁴

Ferramentas de configuração e gerenciamento de conteúdo gerenciam versão e controle de alterações de objetos de conteúdo e componentes funcionais da WebApp.

Ferramentas representativas:

Uma lista abrangente está em www.daveeaton.com/scm/CMTools.html

Ferramentas de desempenho de banco de dados medem desempenho de banco de dados, como, por exemplo, o tempo necessário para executar consultas selecionadas a banco de dados. Essas ferramentas facilitam a otimização do banco de dados.

Ferramentas representativas:

BMC Software (www.bmc.com)

Depuradores são ferramentas típicas de programação que localizam e resolvem defeitos de software no código. Fazem parte da maioria dos ambientes modernos de desenvolvimento de aplicações.

Ferramentas representativas:

Accelerated Technology (www.acceleratedtechnology.com)

Apple Debugging Tools (developer.apple.com/tools/performance/)

IBM VisualAge Environment (www.ibm.com)

Microsoft Debugging Tools (www.microsoft.com)

Sistemas de controle de defeitos registram defeitos e rastreiam seu estado e solução. Algumas incluem ferramentas de relatórios para fornecer informações de gerenciamento sobre taxas de propagação de defeitos e de solução de defeitos.

Ferramentas representativas:

EXCEL Quickbigs (www.excelsoftware.com)

ForeSoft BugTrack (www.bugtrack.net)

McCabe TRUETrack (www.mccabe.com)

Ferramentas de monitoramento de rede observam o nível de tráfego na rede. São úteis para identificar gargalos na rede e teste de link entre sistemas frontal e de retaguarda.

Ferramentas representativas:

Lista em www.slac.stanford.edu/xorg/nmtf/nmtftools.html

Ferramentas de teste de regressão armazenam casos de testes e dados de teste e podem reaplicar os casos de testes após sucessivas mudanças de software.

¹⁴ As ferramentas aqui apresentadas não significam um aval, mas sim uma amostra dessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos revendedores.

Ferramentas representativas:

Compuware QARun (www.compuware.com/products/qacenter/qarun)

Rational VisualTest (www.rational.com)

Seque Software (www.seque.com)

Ferramentas de monitoramento de site monitora o desempenho de um site, muitas vezes, sob a perspectiva de usuário. Use-as para compilar estatísticas como, por exemplo, tempo de resposta de um extremo ao outro (end-to-end) e vazão (throughput) e para verificar periodicamente a disponibilidade de um site.

Ferramentas representativas:

Keynote Systems (www.keynote.com)

Ferramentas de esforço ajudam os desenvolvedores a explorar o comportamento do sistema sob altos níveis de uso operacional e a localizar os pontos de ruptura de um sistema.

Ferramentas representativas:

Mercury Interactive (www.merc-int.com)

Open-source testing tools (www.opensourcetesting.org/performance.php)

Web Performance Load Tester (www.webperformanceinc.com)

Monitores de recurso de sistema fazem parte de muitos sistemas operacionais de servidor e software servidor Web; eles monitoram recursos como espaço de disco, uso da CPU e memória.

Ferramentas representativas:

Successful Hosting.com (www.successfulhosting.com)

Quest Software Foglight (www.quest.com)

Ferramentas de geração de dados de teste ajudam os usuários a gerar dados de teste.

Ferramentas representativas:

Lista em www.softwareqatest.com/qatweb1.html

Comparadores de resultados de teste ajudam a comparar os resultados de um conjunto de testes com os resultados de outro conjunto. Use-as para verificar se as alterações no código não introduziram alterações adversas no comportamento do sistema.

Ferramentas representativas:

Lista útil em www.aptest.com/resources.html

Monitores de transação medem o desempenho de sistemas de processamento de alto volume de transações.

Ferramentas representativas:

QuotiumPro (www.quotium.com)

Software Research eValid (www.soft.com/eValid/index.html)

Ferramentas de segurança de site ajudam a detectar problemas potenciais de segurança. Pode-se muitas vezes configurar ferramentas de investigação de segurança e monitoramento para rodar em horários programados.

Ferramentas representativas:

Lista em www.timberlinetechnologies.com/products/www.html

do teste, os artefatos finais (por exemplo, casos de teste) e os mecanismos para a avaliação dos resultados. O processo de teste abrange sete diferentes tipos de teste.

Teste de conteúdo (e revisões) concentra-se nas várias categorias de conteúdo. O objetivo é descobrir erros de sintaxe e semântica que afetam a exatidão do conteúdo ou a maneira pela qual ele é apresentado ao usuário final. O teste de interface exercita os mecanismos de interação que possibilitam a um usuário se comunicar com a WebApp e validar aspectos estéticos da interface. A finalidade é descobrir erros que resultam de mecanismos de interação mal-implementados ou de omissões, inconsistências ou ambiguidades em semânticas de interface.

O teste de navegação aplica casos de uso criados como parte da atividade de modelagem, no projeto de casos de testes que experimentam cada cenário de uso em relação ao projeto de navegação. Os mecanismos de navegação são verificados para assegurar que quaisquer erros que impedem a realização de um caso de uso sejam identificados e corrigidos. O teste de componente experimenta as unidades de conteúdo e funcional da WebApp.

O teste de configuração tenta descobrir erros e/ou problemas de compatibilidade específicos de um ambiente particular de cliente ou servidor. São então aplicados testes para descobrir erros associados a cada configuração possível. O teste de segurança incorpora uma série de testes projetados para explorar vulnerabilidades na WebApp e seu ambiente. A finalidade é encontrar brechas de segurança. O teste de desempenho abrange uma série de testes projetados para avaliar o tempo de resposta e a confiabilidade da WebApp quando aumentam as demandas de recursos do servidor.

PROBLEMAS E PONTOS A PONDERAR

20.1. Existem quaisquer situações nas quais o teste de WebApp deveria ser totalmente desconsiderado?

20.2. Discuta os objetivos do teste em um contexto de WebApp.

- 20.3.** Compatibilidade é uma dimensão de qualidade importante. O que deve ser testado para garantir que exista compatibilidade em uma WebApp?
- 20.4.** Quais erros tendem a ser mais sérios — erros no cliente ou erros no servidor? Por quê?
- 20.5.** Quais elementos da WebApp podem ser “testados em unidade”? Que tipos de testes devem ser executados apenas depois que os elementos da WebApp estiverem integrados?
- 20.6.** É sempre necessário desenvolver um plano formal de teste escrito? Explique.
- 20.7.** É correto dizer que a estratégia geral de teste para WebApp começa com elementos visíveis ao usuário e passa para os elementos de tecnologia? Há exceções a essa estratégia?
- 20.8.** O teste de conteúdo está *realmente* testando em um sentido convencional? Explique.
- 20.9.** Descreva os passos associados ao teste de banco de dados para uma WebApp. O teste de banco de dados é predominantemente uma atividade do lado do cliente ou do lado do servidor?
- 20.10.** Qual a diferença entre teste que está associado a mecanismos de interface e teste que cuida de semânticas de interface?
- 20.11.** Suponha que você esteja desenvolvendo uma farmácia on-line (**YouCornerPharmacy.com**) dedicada aos aposentados e idosos. A farmácia tem funções típicas, mas também mantém uma base de dados de cada cliente para que possa fornecer informações sobre medicamentos e alertar sobre interações de certos medicamentos. Discuta quaisquer testes especiais de utilização para essa WebApp.
- 20.12.** Suponha que você tenha implementado uma função de verificação de interação de medicamentos para a (Problema 20.11). Discuta os tipos de testes no nível de componente que teriam de ser feitos para garantir que essa função funcione corretamente. [Nota: Teria de ser usada uma base de dados para implementar essa função.]
- 20.13.** Qual a diferença entre teste de sintaxe de navegação e teste de semânticas de navegação?
- 20.14.** É possível testar todas as configurações que uma WebApp pode encontrar no lado do servidor? E no lado do cliente? Se não, como você seleciona um conjunto significativo de testes de configuração?
- 20.15.** Qual o objetivo do teste de segurança? Quem executa essa atividade de teste?
- 20.16.** A **YouCornerPharmacy.com** (Problema 20.11) tornou-se extraordinariamente bem-sucedida, e o número de usuários aumentou significativamente nos primeiros dois meses de operação. Trace um gráfico que mostra o tempo de resposta provável em função do número de usuários para um conjunto fixo de recursos no servidor. Inclua legendas no gráfico para indicar pontos de interesse na “curva de resposta”.
- 20.17.** Em resposta ao seu sucesso, a **YouCornerPharmacy.com** (Problema 20.11) implementou um servidor especial para cuidar de refis (novas doses de medicamentos) das receitas. Na média, 1.000 usuários concorrentes enviam uma solicitação de refil a cada dois minutos. A WebApp faz o download de um bloco de 500 bytes de dados em resposta. Qual a vazão/taxa de saída (*throughput*) aproximada necessária para esse servidor em megabits por segundo?
- 20.18.** Qual a diferença entre teste de carga e teste de esforço?

LEITURAS E FONTES DE INFORMAÇÃO COMPLEMENTARES

A literatura para teste de WebApp continua a evoluir. Livros de Andrews e Whittaker (*How to Break Web Software*, Addison-Wesley, 2006), Ash (*The Web Testing Companion*, Wiley, 2003), Nguyen and his colleagues (*Testing Applications for the Web*, 2d ed., Wiley, 2003), Dustin and his colleagues (*Quality Web Systems*, Addison-Wesley, 2002), e Splaine and Jaskiel [Spl01] estão entre os tratados mais completos do assunto publicado até hoje. Mosley (*Client-Server*

Software Testing on the Desktop and the Web, Prentice Hall, 1999) examina os problemas de teste tanto no lado do cliente quanto no do servidor.

Informações úteis sobre estratégias de teste de WebApp e métodos, bem como uma discussão conveniente sobre ferramentas de teste automáticas, são apresentadas por Stottlemeyer (*Automated Web Testing Toolkit*, Wiley, 2001). Graham and her colleagues (*Software Test Automation*, Addison-Wesley, 1999) oferecem material adicional sobre ferramentas automáticas.

Microsoft (*Performance Testing Guidance for Web Applications*, Microsoft Press, 2008) e Subraya (*Integrated Approach to Web Performance Testing*, IRM Press, 2006) apresentam tratamentos detalhados sobre teste de desempenho para WebApps. Chirillo (*Hack Attacks Revealed*, 2d ed., Wiley, 2003), Splaine (*Testing Web Security*, Wiley, 2002), Klevinsky e colegas (*Hack I.T.*):

Security through Penetration Testing, Addison-Wesley, 2002), e Skoudis (*Counter Hack*, Prentice Hall, 2001) proporcionam muitas informações úteis para aqueles que devem projetar testes de segurança. Além disso, livros que tratam de teste de segurança para software em geral podem oferecer orientações importantes para aqueles que devem testar WebApps. Entre os títulos representativos estão: Basta and Halton (*Computer Security and Penetration Testing*, Thomson Delmar Learning, 2007), Wysopal e colegas (*The Art of Software Security Testing*, Addison-Wesley, 2006), e Gallagher e colegas (*Hunting Security Bugs*, Microsoft Press, 2006).

Há uma grande variedade de recursos de informação sobre teste de WebApp disponível na Internet. Uma lista atualizada de referências na Web, relevante ao teste de WebApp, pode ser encontrada no site www.mhhe.com/engcs/compsci/pressman/professional/olc/ser/htm.