Engenharia de Software

Capítulo 8

Teste de Software

1. Introdução

O desenvolvimento de software utilizando as metodologias, técnicas e ferramentas da Engenharia de Software não oferece a total garantia de qualidade do produto obtido, apesar de melhorá-la significativamente. Por esta razão, uma etapa fundamental na obtenção de um alto nível de qualidade do software a ser produzido é aquela onde são realizados os procedimentos de teste, uma vez que esta é a última etapa de revisão da especificação, do projeto e da codificação.

A realização, de forma cuidadosa e criteriosa, dos procedimentos associados ao teste de um software assume uma importância cada vez maior dado o impacto sobre o funcionamento (e o custo) que este componente tem assumido nos últimos anos. Por esta razão, o esforço despendido para realizar a etapa de teste pode chegar a 40% do esforço total empregado no desenvolvimento do software.

No caso de programas que serão utilizados em sistemas críticos (aqueles sistemas dos quais dependem vidas humanas, como controle de vôo e a supervisão de reatores nucleares), a atividade de teste pode custar de 3 a 5 vezes o valor gasto nas demais atividades de desenvolvimento do software.

O objetivo deste capítulo é apresentar, de forma breve, os principais conceitos e técnicas relacionados ao teste de software.

2. Fundamentos do teste de software

2.1. Objetivos

Os objetivos do teste de software podem ser expressos, de forma mais clara, pela observação das três regras definidas por Myers:

- A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro;
- Um bom caso de teste é aquele que apresenta uma elevada probabilidade de revelar um erro ainda não descoberto;
- Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

As três regras expressam o objetivo primordial do teste que é o de encontrar erro, contrariando a falsa idéia de que uma atividade de teste bem sucedida é aquela em que nenhum erro foi encontrado.

A etapa de teste deve ser conduzida de modo que o maior número de erros possível seja encontrado com um menor dispêndio de tempo e esforço.

2.2. O projeto de casos de teste

A realização, com sucesso, da etapa de teste de um software deve ter, como ponto de partida, uma atividade de projeto dos casos de teste deste software. Projetar casos de teste para um software pode ser uma atividade tão complexa quanto a de projeto do próprio software, mas ela é necessária como única forma de conduzir, de forma eficiente e eficaz, o processo de teste.

Os princípios básicos do teste de qualquer produto resultante de uma tarefa de engenharia são:

- conhecida a função a ser desempenhada pelo produto, testes são executados para demonstrar que cada função é completamente operacional, este primeiro princípio deu origem a uma importante abordagem de teste, conhecida como o *teste de caixa preta (black box)*;
- com base no conhecimento do funcionamento interno do produto, realiza-se testes para assegurar de que todas as peças destes estão completamente ajustadas e realizando a contento sua função; à abordagem originada por este segundo princípio, foi dado o nome de *teste de caixa branca (white box)*, devido ao fato de que maior ênfase é dada ao desempenho interno do sistema (ou do produto).

2.2.1. O software e o teste de caixa preta

Quando o procedimento de teste está relacionado ao produto de software, o teste de caixa preta refere-se a todo teste que implica na verificação do funcionamento do software através de suas interfaces, o que, geralmente, permite verificar a operacionalidade de todas as suas funções. É importante observar que, no teste de caixa preta, a forma como o software está organizado internamente não tem real importância, mesmo que isto possa ter algum impacto na operação de alguma função observada em sua interface.

2.2.2. O software e o teste de caixa branca

Um teste de caixa branca num produto de software está relacionado a um exame minucioso de sua estrutura interna e detalhes procedimentais. Os caminhos lógicos definidos no software são exaustivamente testados, pondo à prova conjuntos bem definidos de condições ou laços. Durante o teste, o "status" do programa pode ser examinado diversas vezes para eventual comparação com condições de estado esperadas para aquela situação.

Apesar da importância do teste de caixa branca, não se deve guardar a falsa idéia de que a realização de testes de caixa branca num produto de software vai oferecer a garantia de 100% de correção deste ao seu final. Isto porque, mesmo no caso de programas de pequeno e médio porte, a diversidade de caminhos lógicos pode atingir um número bastante elevado, representando um grande obstáculo para o sucesso completo desta atividade..

3. MODALIDADES DE TESTE

3.1. Testes estáticos e dinâmicos

Uma primeira divisão que pode ser estabelecida em relação ao teste de software corresponde à forma de utilização do código obtido na etapa de implementação (ou codificação). Segundo esta ótica, pode-se organizar os testes em **estáticos** e **dinâmicos**.

3.1.1. Os Testes Estáticos

Os testes estáticos são aqueles realizados sobre o código-fonte do software, utilizando como técnica básica a inspeção visual. Este tipo de teste é de simples implementação, uma vez que não há necessidade de execução do programa para obter-se resultados. Eles podem ser utilizados utilizando a técnica de leitura cruzada, onde um leitor é atribuído para avaliar o trabalho de cada programador do software. Uma outra forma de realizar o teste é por inspeção, onde uma equipe designada analisa o código à luz de um questionário especialmente concebido — a check list.

Nos dois casos, os responsáveis pela realização do teste não realizam nenhum tipo de correção no código, limitando-se a assinalar os erros encontrados.

Eventualmente, o teste estático pode ser automatizado com o auxílio de ferramentas de análise estática, que podem ser simples geradores de referências cruzadas ou, no caso de ferramentas mais sofisticadas, serem dotadas de funções de análise do fluxo de dados do programa.

3.1.2. Os Testes Dinâmicos

Os testes dinâmicos são os procedimentos baseados na execução do código binário do programa, sendo esta execução realizada com base em subconjuntos de dados — o jogo de teste.

A escolha do subconjunto de dados a ser utilizado para o teste pode ser feita com base em aspectos estruturais do software (obtido a partir do código-fonte) ou em aspectos funcionais (a partir da especificação do programa).

3.2. Testes de unidade, integração, validação e sistema

Uma outra forma de subdividir o teste de software é quanto ao seu objetivo na busca por erros do programa. Sob este ponto de vista, encontra-se os seguintes tipos de teste:

3.2.1. O Teste de Unidade

O teste de unidade objetiva a verificação de erros existentes nas unidades de projeto do mesmo, à qual daremos o nome de **módulo**. Nesta modalidade de teste, é importante utilizar as informações contidas no documento de projeto detalhado do software, as quais servirão de guia para sua aplicação. O teste de unidade é, de certa forma, uma técnica de teste de caixa branca, podendo ser realizado em paralelo sobre diferentes módulos.

3.2.2. O Teste de Integração

O Teste de Integração, como o nome indica, objetiva a busca de erros surgidos quando da integração das diferentes unidades componentes do software. É importante lembrar que, o fato de se ter analisado os módulos do software de forma exaustiva (através de procedimentos de teste de unidade), não há nenhuma garantia de que estes, uma vez colocados em conjunto para funcionar, não apresentarão anomalias de comportamento.

Uma das maiores causas de erros encontrados durante o teste de integração são os chamados erros de interface, devido, principalmente, às incompatibilidades de interface entre módulos que deverão trabalhar de forma cooperativa.

3.2.3. O Teste de Validação

Ao final do teste de integração, o software é finalmente estruturado na forma de um pacote ou sistema. A forma mais simples de definição do teste de validação é a verificação de que o software como um todo cumpre corretamente a função para a qual ele foi especificado. É importante lembrar que, no documento de especificação do software,

quando no início da sua concepção, são definidos os chamados critérios de validação, que servirão de guia para o julgamento de aprovação ou não do software nesta etapa de testes.

3.2.4. O Teste de Sistema

Neste tipo de teste, são verificados os aspectos de funcionamento do software, integrado aos demais elementos do sistema, como o hardware e outros elementos. Dependendo do destino do software, é somente neste momento do teste em que alguns peris de funcionamento do software podem ser efetivamente testados (particularmente, aqueles aspectos do funcionamento que dependem da interação dos demais elementos do sistema).

Apresentada uma breve descrição destes tipos de teste, informações mais detalhadas serão apresentadas nas seções que seguem.

4. TESTE DE UNIDADE

4.1. ASPECTOS A SEREM TESTADOS

A figura 8.1 ilustra a forma como são desenvolvidos os testes de unidade. Os aspectos verificados no contexto deste teste, normalmente são:

- a *interface*, em busca de erros de passagem de dados para dentro e para fora do módulo;
- as *estruturas de dados locais*, para se prover a garantia de que todos os dados armazenados localmente mantêm sua integridade ao longo da execução do módulo;

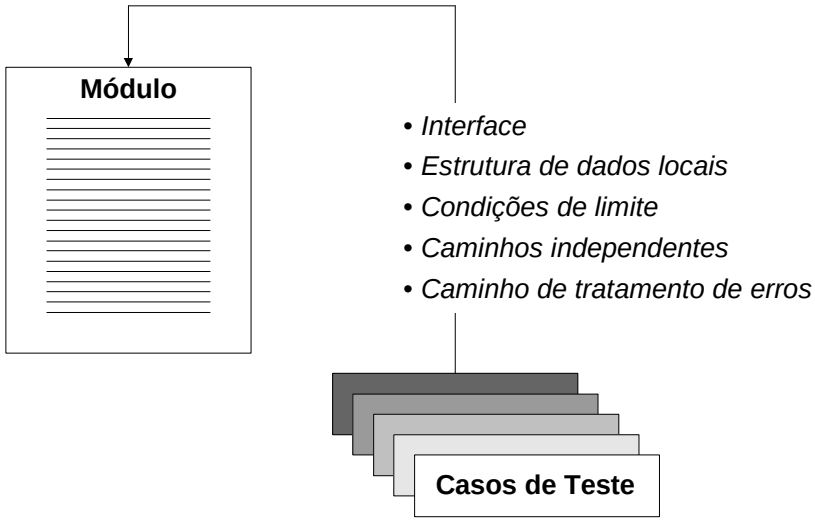

Figura 8.1 - Teste de Unidade

- as *condições limite*, que permitem verificar que o módulo executa respeitando os valores máximos e mínimos estabelecidos para seu processamento;
- os caminhos independentes (ou caminhos básicos) da estrutura de controle são analisados para se ter garantia de que todas as instruções do módulo foram executadas pelo menos uma vez;
- finalmente, os *caminhos de tratamento de erros* (se existirem), serão testados para observar a robustez do módulo.

4.1.1. O teste de interface

O teste de interface é o primeiro e o mais importante teste a ser aplicado sobre uma unidade de software. Ele deve ser realizado em primeiro lugar, pois, qualquer anomalia observada durante a realização deste teste põe em dúvida os resultados obtidos nos demais testes sobre a mesma unidade. Durante a realização deste teste, os seguintes aspectos devem ser observados:

- coerência (em termos de números, atributos e sistemas de unidades) entre argumentos de um módulo e os parâmetros de entrada;
- coerência (em termos de números, atributos e sistemas de unidades) entre argumentos passados aos módulos chamados e os parâmetros de entrada;
- verificação sobre a consistência dos argumentos e a ordem de passagem destes nas funções embutidas;
- existência de referências a parâmetros não associados ao ponto de entrada atual;
- alteração de argumentos de entrada/saída;
- consistência de variáveis globais ao longo dos módulos;
- passagem de restrições sobre argumentos.

No caso de unidades que envolvam tratamento de entradas/saídas, todas as operações envolvendo tratamento de arquivos ou programação de dispositivos periféricos devem ser cuidadosamente (ou exaustivamente) verificadas.

4.1.2. O teste de estruturas de dados

Com relação as estruturas de dados, devem ser verificadas não apenas aquelas que serão utilizadas exclusivamente no contexto da unidade, mas também aquelas definidas em termos globais. Praticamente todas as linguagens de programação oferecem mecanismos para a definição de estruturas de dados com diferentes escopos. Neste teste, deve ser verificado o bom uso destes mecanismos e se esta utilização está sendo feita de modo coerente com o que foi estabelecido no projeto.

Os erros mais comuns detectados neste tipo de teste são:

- digitação inconsistente ou imprópria (por exemplo, identificadores de uma variável ou tipo de dados digitados de forma incorreta);
- iniciação incorreta de variáveis (valores iniciais incorretos);
- inconsistência nos tipos de dados;
- *underflow* e *overflow*.

4.1.3. O teste de condições limite

Este teste busca verificar que o que foi definido a nível de projeto para as condições limite de uma unidade está sendo respeitado a nível de implementação. Por exemplo, um módulo encarregado de implementar um mecanismo de temporização deve ser testado para observar se o tempo máximo de espera está compatível com o que foi especificado.

4.1.4. O teste de caminhos de execução

O sucesso da realização deste tipo de teste depende fortemente da complexidade do algoritmo implementado a nível da unidade sob teste. É que neste caso, é necessário que os casos de teste a serem aplicados permitam percorrer os diversos caminhos de execução da unidade, os quais serão em maior número quanto mais complexa for a unidade. Os erros mais comuns verificados neste tipo de teste são:

- precedência aritmética incorreta;
- operações envolvendo dados de tipos diferentes;
- inicialização incorreta;
- erros deprecisão;
- representação incorreta de uma expressão;
- laços mal definidos;
- comparação de diferentes tipos de dados;
- operadores lógicos utilizados incorretamente;
- etc.

4.1.5. O teste de caminhos de tratamento de erros

É desejável que um projeto de software estabeleça caminhos de tratamento de erros para as unidades que o compõem, permitindo que os problemas que venham a ocorrer durante a utilização de um programa, especialmente no caso de programas interativos, possam ser recuperados e a execução do programa possa ser retomada normalmente. Entretanto, embora muitos projetistas e programadores tenham o hábito de inserir este tipo de tratamento, raramente eles são testados.

É importante que estes caminhos venham a ser testados também, para detectar os erros mais comuns deste tipo de tratamento que são:

- descrição incompreensível do erro (por exemplo, **erro 1356**);
- descrição incorreta do erro (o erro indicado não corresponde ao erro ocorrido);
- ocorrência de intervenção do sistema antes da indicação ou tratamento do erro pela unidade;
- processamento incorreto das condições de exceção (por exemplo, tratar ou idicar a ocorrência de um erro que não aconteceu);
- descrição imprecisa do erro (não informa como localizar ou evitar que o erro volte a ocorrer).

4.2. Procedimento do teste de unidade

O teste de unidade é considerado uma atividade vinculada à codificação. Uma vez desenvolvido o código-fonte, os casos de teste devem ser projetados. O projeto dos casos de teste pode ser uma conseqüência da revisão do código-fonte da unidade, o que vai permitir estabelecer as condições para analizar a unidade segundo os diferentes pontos de vista citados acima. Parte do projeto dos casos de teste deve ser também a especificação dos resultados esperados em cada um deles.

As unidades de software não são programas autônomos, sendo dependentes de outras unidades. Por esta razão, o teste de unidade exige a definição de módulos de software específicos, denominados *Drivers* ou *Stubs*. Como pode ser observado na figura 8.2, um *driver* faz o papel (numa versão bastante simplificada) do programa principal ou de uma unidade de nível superior que ativa a unidade a ser testada comunicando-se com ela através da interface, imprimindo dados que sejam importantes para a tarefa de análise daquela unidade. Os *stubs* são construídos para desempenhar o papel das unidades de nível mais baixo (ou mesmo, de componentes de hardware do sistema) com o fim de viabilizar a execução da unidade sob teste. Tanto os drivers quanto os stubs são programas adicionais a serem desenvolvidos e, por esta razão, devem ser escritos da forma mais simples possível para minimizar o esforço adicional de desenvolvimento de software. Entretanto, esta "simplicidade" nem sempre é facilmente obtida, pois determinadas unidades exigem a existência de drivers ou stubs mais sofisticados para que a atividade de teste seja eficaz.

5. Teste de Integração

O teste de integração é realizado no sentido de assegurar que, uma vez que as unidades foram testadas e seu bom comportamento foi verificado individualmente, estas vão trabalhar de forma cooperativa e harmoniosa quando forem associadas. O problema é colocá-los juntos, considerando que cada um deles foi definido com uma interface. Quem ainda não tentou ligar um plug de três pinos numa tomada com dois furos?

Os erros mais comuns de integração são:

- perdas de dados através das interfaces;
- efeitos inesperados da combinação de duas ou mais funções;
- estruturas de dados globais apresentando problemas;
- imprecisões individualmente aceitáveis podem gerar imprecisões absurdas;
- etc.

5.1. Sistemática do teste de integração

A idéia básica por trás do teste de integração é a construção, passo-a-passo, do programa, associando suas unidades e testando esta associação, até que se atinja a totalidade do programa projetado.

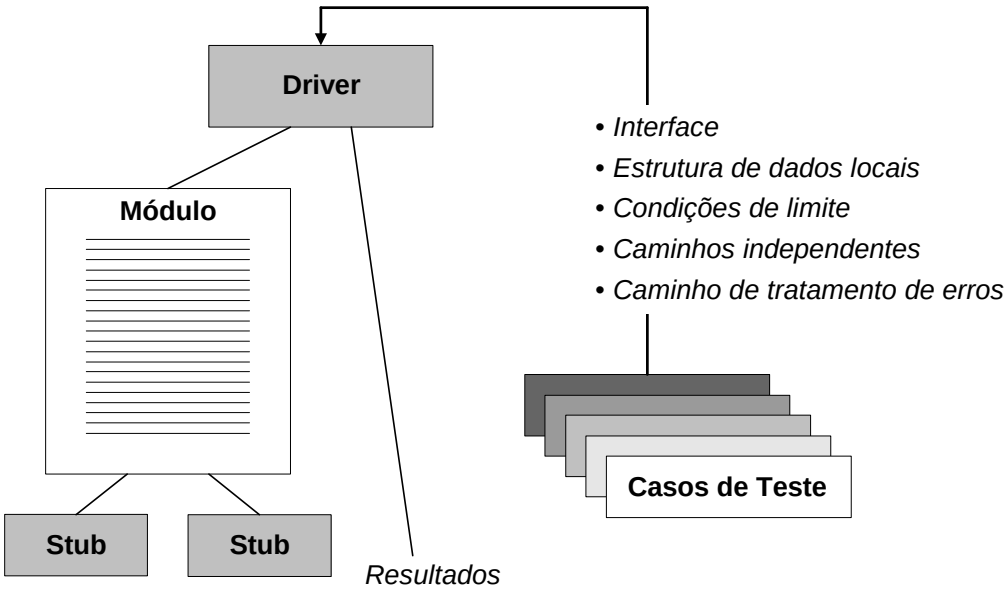

Figura 8.2 - Ambiente de teste de unidade.

Uma abordagem muitas vezes adotada é a abordagem *big bang*, um processo de integração não-incremental na qual todos os módulos são associados e o programa é testado como um todo. Na maioria dos testes realizados segundo esta abordagem, o resultado não é outro senão catastrófico. A correção dos erros torna-se uma tarefa extremamente complexa, principalmente porque é bastante difícil isolar as causas dos erros dada a complexidade do programa completo. Mesmo quando alguns erros são detectados e corrigidos, outros aparecem e o processo caótico recomeça.

A integração incremental tem-se mostrado mais eficiente neste contexto, uma vez que, segundo esta abordagem, o programa vai sendo "construído" aos poucos e testado por partes. Neste tipo de integração, o isolamento das causas de erros e suas correções são obtidos mais facilmente. Além disso, o teste das interfaces pode ser feito com maior eficácia.

As seções a seguir vão apresentar a descrição de duas estratégias de teste de integração bastante clássicas: a integração *top-down* e a integração *bottom-up*.

5.2. Integração Top-Down

A integração *top-down*, ilustrada na figura 8.3, é uma estratégia de integração incremental onde os módulos são integrados segundo um movimento de cima para baixo. O processo de integração inicia-se no módulo de controle principal (representado na figura 8.3 pelo módulo $\mathbf{M_1}$), sendo que o caminho de descida pode ser definido de duas formas: *depth-first* (descida em profundidade) ou *breadth-first* (descida em largura).

Na ilustração da figura 8.3, a descida em depth-first englobaria inicialmente todos os módulos relacionados a um caminho de controle principal. A escolha deste caminho depende, principalmente, das características do software em desenvolvimento. Supondo que o caminho principal fosse aquele liderado pelo módulo $\mathbf{M_2}$, a integração seria feita abrangendo os módulos $\mathbf{M_5}$, $\mathbf{M_6}$ e $\mathbf{M_8}$. Só então os módulos mais à direita seriam integrados. No caso da integração breadth-first, todos os módulos subordinados a um dado nível seriam integrados primeiro. No caso do exemplo, a integração partiria de $\mathbf{M_1}$ e atingiria inicialmente os módulos $\mathbf{M_2}$, $\mathbf{M_3}$ e $\mathbf{M_4}$.

O processo de integração é conduzido considerando 5 passos:

- o módulo principal atua como um *Driver*, e os módulos de nível superior à porção sob teste são substituídos por stubs especialmente construídos;

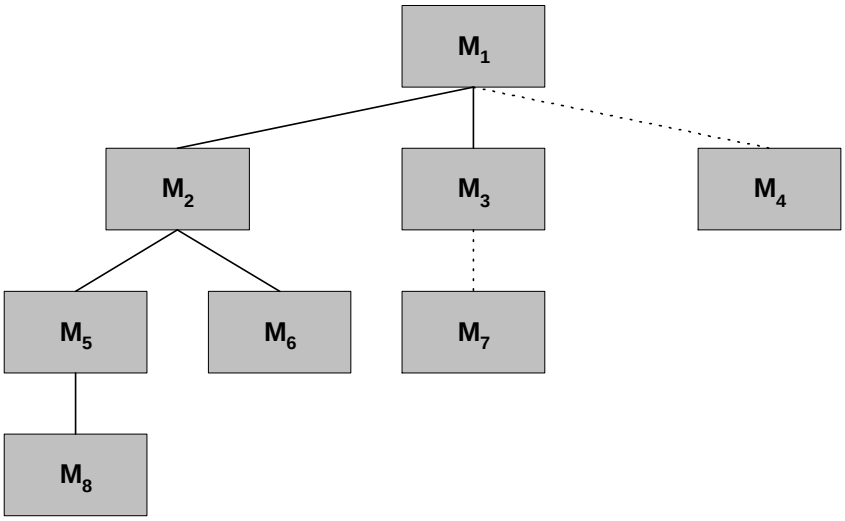

Figura 8.3 - Integração *Top-Down*.

- dependendo do tipo de descida escolhida (*depth-first* ou *breadth-first*), os stubs podem ir sendo substituídos, gradualmente, por módulos reais, já testados;
- à medida que os módulos são integrados, os testes vão sendo realizados;
- quando uma bateria de testes é terminada, os stubs vão sendo substituídos por módulos reais;
- finalmente, pode ser realizado um teste de regressão, que tem por objetivo garantir que novos erros não tenham sido introduzidos durante o processo de descida.

5.3. Integração Bottom-Up

O teste de integração segundo a abordagem *bottom-up* sugere uma integração igualmente incremental, mas no sentido inverso, ou seja, dos níveis mais baixos para o nível mais alto.

Como ilustrado pela figura 8.4, a construção se inicia a partir dos módulos ditos *atômicos* (os módulos do nível mais baixo), facilitando, de certa forma, integração pois, quando se para de um nível para um superior, os módulos subordinados necessários para testar a integração neste novo nível estão disponíveis (e testados).

Os passos normalmente executados nesta estratégia de teste são:

- os módulos de nível mais baixo são agrupados em *clusters* (ou *construções*), que executam uma função específica do software;
- um driver é construído para coordenar a entrada e a saída dos casos de teste;
- o *cluster* é testado;
- os drivers são removidos e os *clusters* são combinados com módulos dos níveis superiores para uma nova etapa, até que todo o programa tenha sido remontado.

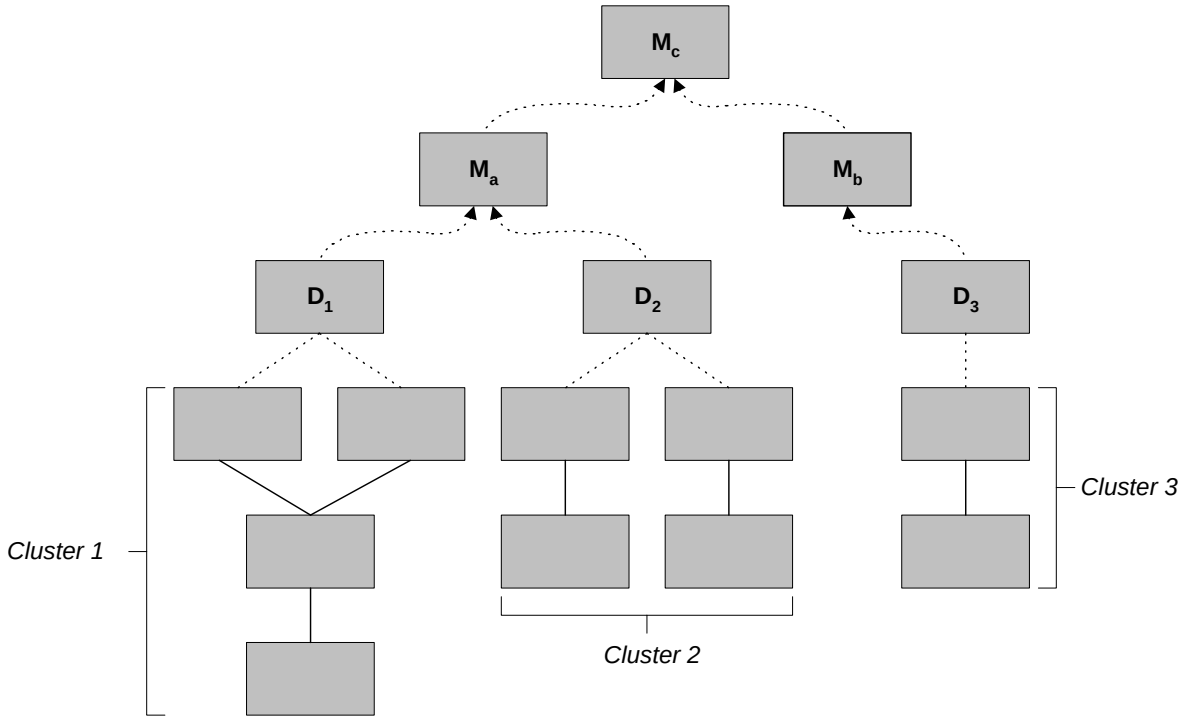

Figura 8.4 - Integração *Bottom-Up*.

5.4. Comparação entre as duas estratégias

Quando se tenta confrontar as duas estratégias descritas nas seções precedentes, não é difícil antecipar que a vantagem de um pode constituir-se na desvantagem do outro. No caso da abordagem *top-down*, a maior desvantagem é, sem dúvida, a necessidade de se construir *stubs* para representar módulos subordinados que não foram suficientemente testados (por razões óbvias). Por outro lado, a possibilidade de se ter resultados sobre as principais de controle (uma vez que são os módulos de mais alto nível que suportam estas funções), constitui-se num ponto a favor desta estratégia.

Já no caso da estratégia *bottom-up*, pode-se dizer que o programa é completamente inexistente até que o último módulo seja finalmente integrado. Mas esta deficiência é largamente compensada pelo fato de haver muito maior facilidade para projetar os casos de teste e pela não exigência na criação de *stubs* para a realização dos testes dos módulos dos níveis mais altos.

6. Teste de Validação

O teste de validação tem por objetivo determinar se o programa, mesmo que funcionando corretamente (em conseqüência das etapas de teste anteriores... de unidade e de integração) apresenta as propriedades e a funcionalidade definidas na etapa de especificação dos requisitos.

Pode-se dizer que um teste de validação bem sucedido é aquele que dá um máximo de respostas sobre a capacidade do software apresentar (ou não) um funcionamento o mais próximo possível daquele esperado pelo cliente (ou usuário). O problema é como medir esta proximidade. Nesta medida, um conjunto de informações importantes são aquelas definidas no documento de especificação de requisitos. Esta é uma das razões pelas quais é importante que se busque, na fase de especificação dos requisitos, utilizar parâmetros mensuráveis para definir os requisitos do software (rever o capítulo 5).

6.1. O processo de teste de validação

O teste de validação é realizado através de um conjunto de testes de caixa preta cujo objetivo é determinar a conformidade do software com os requisitos definidos nas fases preliminares da concepção. Um documento de plano de testes (eventualmente definido na etapa de Análise de Requisitos) define quantos e quais testes serão realizados.

Os requisitos são analizados sob vários pontos de vista: funcionais, desempenho, documentação, interface, e outros requisitos (portabilidade, compatibilidade, remoção de erros, etc.).

O resultado do teste de validação pode apresentar um dos dois resultados:

1. as características de função, desempenho e outros aspectos enquadram-se nos parâmetros estabelecidos na especificação de requisitos;

2. é descoberto um "desvio" das especificações... neste caso, uma lista de deficiências é criada para relacionar de que forma (ou quanto) as características do software obtido afasta-se dos requisitos.

No caso de descoberta de erros, estes dificilmente serão corrigidos antes do período definido para a conclusão do programa. Eventualmente, uma negociação (extensão de prazo de entrega, por exemplo) deverá ser realizada em conjunto com o cliente para possibilitar a eliminação destas deficiências.

6.2. Revisão de configuração

Um aspecto de importância no processo de validação é a chamada *revisão de configuração*. Ela consiste em garantir que todos os elementos de configuração do software tenham sido adequadamente desenvolvidos e catalogados corretamente com todos os detalhes necessários que deverão servir de apoio à fase de manutenção do software. Esta revisão, algumas vezes denominada auditoria, está ilustrada na figura 8.5.

6.3. Testes Alfa e Beta

Apesar do esforço na validação de um software, na maioria das vezes é extremamente difícil para o desenvolvedor ou programador prever as diferentes maneiras como o software será utilizado. Principalmente no caso de softwares interativos, os usuários muitas vezes experimentam comandos e entradas de dados os mais diversos, o que pode ser um verdadeiro atentado à robustez de um sistema. Da mesma forma, dados de saída ou formatos de apresentação de resultados podem parecer ótimos para o analista, mas completamente inadequados para o usuário.

Quando um software é construído sob demanda, uma bateria de testes de aceitação é conduzida, envolvendo o cliente. A duração dos testes de aceitação pode variar entre algumas semanas e vários meses, dependendo da complexidade ou natureza da aplicação.

No caso de um software de prateleira, aquele software que é construído para um segmento do mercado de software e que é destinado à utilização por um grande conjunto grande de usuários, os editores de software fazem uso de um processo denominado *teste alfa* e *teste beta*.

6.3.1. Testes Alfa

Os testes alfa são realizados com base no envolvimento de um cliente, ou numa amostra de clientes, sendo realizado nas instalações do desenvolvedor do software. O software é utilizado num ambiente natural, sendo que o desenvolvedor observa o comportamento do cliente e vai registrando as anomalias detectadas durante a utilização

6.3.2. Testes Beta

Já os testes beta são conduzidos em uma ou mais instalações do cliente pelo usuário final do software.

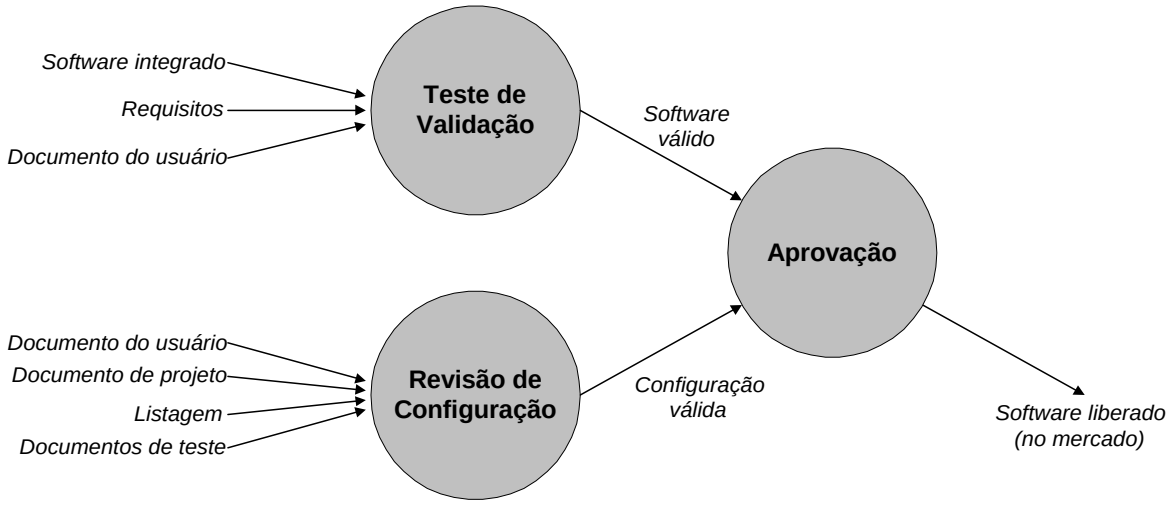

Figura 8.5 - A revisão de configuração.

Neste caso, é praticamente impossível haver um controle da parte do desenvolvedor sobre os erros encontrados. Estes normalmente são registrados pelo próprio usuário e encaminhados ao desenvolvedor na forma de um relatório escrito. Atualmente, muitas empresas fazem uso deste processo, encaminhando (ou disponibilizando via Internet) versões beta de seus softwares, sendo que as informações sobre erros encontrados podem ser fornecidas ao desenvolvedor através de correio eletrônico ou através do acesso a páginas Web especialmente desenvolvidas.

Infelizmente porém, muitos usuários de versões beta não têm a preocupação de contribuir efetivamente para a validação, apenas fazendo uso do software para seu próprio interesse.

7. Teste de Sistema

Como já foi mencionado, o software definido apenas como um elemento de um sistemas que pode envolver hardware, elementos humanos e outros. Por esta razão, uma vez validado, ele será incorporado aos demais elementos.

Um problema comumente encontrado neste tipo de teste é o clássico "apontar o dedo", onde o responsável pelo desenvolvimento de um elemento do sistema tenta livrar-se da responsabilidade do erro, acusando outro elemento do sistema como causador do erro.

Por esta razão, é importante que o Engenheiro de Software tente prever potenciais problemas de integração a outros elementos do sistema e inclua no software caminhos de tratamento de erros para este sistema. Por exemplo, num software de comunicação podem ser incluídos módulos de tratamento de erros que informem quando um determinado dispositivo de E/S não está respondendo corretamente.

O teste de sistema inclui diversas modalidades de teste, cujo objetivo é testar o sistema computacional como um todo. Embora cada teste tenha uma finalidade diferente, o objetivo global acaba sendo atingido, uma vez que estes abrangem todos os elementos constituintes do sistema, verificando se estes foram adequadamente integrados.

7.1. O teste de recuperação

Neste tipo de teste, o objetivo é observar a capacidade do sistema para recuperar-se da ocorrência de falhas, num tempo previamente determinado. Em alguns casos (cada vez mais freqüentes), exige-se que o sistema seja tolerante a falhas, ou seja, que nele seja previsto processamento que permita retomar seu funcionamento mesmo no caso de ocorrência de algumas falhas.

Neste tipo de teste, falhas são provocadas artificialmente (por uma técnica denominada *injeção de falhas*), de modo a analisar a capacidade do sistema do ponto de vista da recuperação. Os valores de tempo exigido para recuperação do sistema (seja por ação automática ou devido à intervenção de um operador humano) devem ser registrados e confrontados aos valores especificados.

7.2. O teste de segurança

O teste de segurança visa garantir que o sistema não vai provocar danos recuperáveis ou não ao sistema pela sua própria ação. Por exemplo, num sistema de informação acadêmica, um aluno deve ter autorização de acesso para a visualização das notas obtidas nas disciplinas que cursou (histórico escolar), mas não deve ter a possibilidade de alterar seus valores.

No teste de segurança, o analista deve desempenhar um papel semelhante ao de um *hacker*, tentando contornar todos os mecanismos de segurança implementados no mesmo. As ações a experimentar são as mais diversas:

- derrubar o sistema como um todo;
- acessar informações confidenciais;
- modificar informações de bases de dados;
- interferir no funcionamento do sistema;
- introduzir vírus de computador no sistema;
- sobrecarregar o sistema pela multiplicação de processos executando;
- etc.

7.3. O teste de estresse

O teste de estresse, como o nome indica, consiste em verificar como o sistema vai se comportar em situações limite. Este limite pode ser verificado sob diferentes pontos de vista, dependendo das características do software. Alguns aspectos que podem ser observados neste contexto são:

- limite em termos de quantidade de usuários conectados a um determinado sistema servidor;
- quantidade de utilização de memória;
- uso em diferentes versões de processadores;
- quantidade de bloqueios encontrados num dado período de utilização;
- etc.

7.4. O teste de desempenho

Este tipo de teste consiste em verificar se os requisitos de desempenho estão sendo atendidos para o sistema como um todo. Este aspecto, que pode não Ter grande importância em alguns sistemas (quais?), torna-se crítico em aplicações envolvendo sistemas embarcados e sistemas multimídias, que pertencem à classe dos sistemas tempo-real. Nestes sistemas, o não atendimento a um requisito de tempo pode afetar de forma irreversível a função do sistema e, no caso de alguns sistemas (os chamados sistemas críticos), a não realização desta função ou o não atendimento a estes requisitos temporais pode resultar em prejuízos catastróficos (risco a vidas humanas ou grandes prejuízos financeiros).

Os testes de desempenho são realizados, normalmente, com o auxílio de instrumentação de hardware e de software que permitam medir como os recursos do sistema estão sendo utilizados. O uso da instrumentação pode facilitar a coleta de dados e o registro de status do sistema nos seus diferentes pontos de funcionamento.