A Psicologia dos Testes de Software

Uma das principais causas de testes de software deficientes é o fato de que a maioria dos programadores começa com uma definição errada do termo. Eles podem dizer:

- **"Testar é o processo de demonstrar que não há erros."**
- **"O propósito do teste é mostrar que um programa executa suas funções pretendidas corretamente."**
- **"O teste é o processo de estabelecer confiança de que um programa faz o que deveria fazer."**

Essas definições estão de cabeça para baixo.

Quando você testa um programa, deseja agregar valor a ele. Agregar valor por meio do teste significa aumentar a qualidade ou a confiabilidade do programa. Aumentar a confiabilidade do programa significa encontrar e remover erros.

Portanto, não teste um programa para mostrar que ele funciona; em vez disso, você deve partir da suposição de que o programa contém erros (uma suposição válida para quase qualquer programa) e, então, testá-lo para encontrar o maior número possível de erros.

Assim, uma definição mais apropriada seria esta:

Testar é o processo de executar um programa com a intenção de encontrar erros.

Embora isso possa soar como um jogo de semântica sutil, é realmente uma distinção importante.
Entender a verdadeira definição de teste de software pode fazer uma diferença profunda no
sucesso de seus esforços.
Os seres humanos tendem a ser altamente orientados para objetivos, e estabelecer o objetivo adequado tem um efeito psicológico importante. Se nosso objetivo é demonstrar que um programa não tem erros, então seremos subconscientemente direcionados para esse objetivo; ou seja, tendemos a selecionar dados de teste que têm uma baixa
probabilidade de fazer o programa falhar. Por outro lado, se nosso objetivo é demonstrar que um
programa tem erros, nossos dados de teste terão uma probabilidade maior de encontrar erros. A última abordagem adicionará mais valor ao programa do que a primeira.

Um caso de teste que encontra um novo erro dificilmente pode ser considerado mal sucedido; ao contrário, provou ser um investimento valioso. Um caso de teste mal sucedido é aquele que faz com que um programa produza o resultado correto sem encontrar nenhum erro.

Considere a analogia de uma pessoa que visita um médico por causa de uma sensação geral de mal-estar. Se o médico solicitar alguns exames laboratoriais que não localizam o problema, não chamamos os exames laboratoriais de "bem-sucedidos"; eles foram exames malsucedidos, pois o patrimônio líquido do paciente foi reduzido pelas custosas taxas de laboratório, o paciente ainda está doente e pode questionar a capacidade do médico como diagnosticador. No entanto, se um exame laboratorial determinar que o paciente tem uma úlcera péptica, o exame é bem-sucedido porque o médico agora pode iniciar o tratamento adequado. Portanto, a profissão médica parece usar essas palavras no sentido correto. A analogia, claro, é que devemos pensar no programa, quando começamos a testá-lo, como o paciente doente.

Princípios de Teste de Software

Dando continuidade à premissa principal deste capítulo, de que as considerações mais importantes em testes de software são questões de psicologia, podemos identificar um conjunto de princípios ou diretrizes vitais para os testes. A maioria desses princípios pode parecer óbvia, mas frequentemente são negligenciados. A Tabela 2.1 resume esses princípios importantes, e cada um é discutido em mais detalhes nos parágrafos seguintes.

Princípio 1: Uma parte necessária de um caso de teste é a definição do resultado ou saída esperada.

Este princípio óbvio é um dos erros mais frequentes em testes de programas. Novamente, trata-se de algo baseado na psicologia humana. Se o resultado esperado de um caso de teste não foi previamente definido, é provável que um resultado plausível, mas errado, seja interpretado como correto devido ao fenômeno de "os olhos veem o que querem ver". Em outras palavras, apesar da definição destrutiva correta do teste, ainda existe o desejo subconsciente de ver o resultado correto. Uma maneira de combater isso é incentivar um exame detalhado de toda a saída, especificando com precisão, de antemão, a saída esperada do programa. Portanto, um caso de teste deve consistir de dois componentes:

1. Uma descrição dos dados de entrada para o programa.
2. Uma descrição precisa da saída correta do programa para aquele conjunto de dados de entrada.

Um problema pode ser caracterizado como um fato ou grupo de fatos para os quais não temos uma explicação aceitável, que parecem incomuns ou que não se encaixam com nossas expectativas ou preconceitos. Deveria ser óbvio que algumas crenças anteriores são necessárias para que algo pareça problemático. Se não há expectativas, não pode haver surpresas.

Princípio 2: Um programador deve evitar tentar testar seu próprio programa.

Qualquer escritor sabe—ou deveria saber—que é uma má ideia tentar editar ou revisar o próprio trabalho. Você sabe o que a peça deveria dizer e pode não perceber quando ela diz algo diferente. E você realmente não quer encontrar erros no seu próprio trabalho. O mesmo se aplica aos autores de software.

Outro problema surge com a mudança de foco em um projeto de software. Depois que um programador projetou e codificou um programa de maneira construtiva, é extremamente difícil mudar repentinamente de perspectiva para olhar o programa com um olhar destrutivo.

Como muitos proprietários de imóveis sabem, remover papel de parede (um processo destrutivo) não é fácil, mas é quase insuportavelmente deprimente se foram suas próprias mãos que colocaram o papel de parede em primeiro lugar. Da mesma forma, a maioria dos programadores não consegue testar efetivamente seus próprios programas porque não conseguem mudar de mentalidade para tentar expor erros. Além disso, um programador pode, subconscientemente, evitar encontrar erros por medo de retaliação de colegas, de um supervisor, de um cliente ou do proprietário do programa ou sistema que está sendo desenvolvido.

Além desses problemas psicológicos, há um segundo problema significativo: o programa pode conter erros devido ao mal-entendido do programador sobre a declaração do problema ou a especificação. Se for esse o caso, é provável que o programador leve o mesmo mal-entendido para os testes de seu próprio programa.

Isso não significa que seja impossível para um programador testar seu próprio programa. Em vez disso, implica que o teste é mais eficaz e bem-sucedido quando feito por outra pessoa. Vale ressaltar que esse argumento não se aplica à depuração (correção de erros conhecidos); a depuração é mais eficientemente realizada pelo programador original.

Princípio 3: Uma organização de programação não deve testar seus próprios programas.

O argumento aqui é similar ao argumento anterior. Um projeto ou organização de programação é, em muitos sentidos, uma organização viva com problemas psicológicos semelhantes aos dos programadores individuais. Além disso, em muitos ambientes, uma organização de programação ou um gerente de projeto é amplamente avaliado pela capacidade de produzir um programa até uma determinada data e por um custo específico. Uma razão para isso é que é fácil medir os objetivos de tempo e custo, mas é extremamente difícil quantificar a confiabilidade de um programa. Portanto, é difícil para uma organização de programação ser objetiva ao testar seus próprios programas, porque o processo de teste, se abordado com a definição adequada, pode ser visto

como algo que diminui a probabilidade de atender ao cronograma e aos objetivos de custo.
Novamente, isso não quer dizer que seja impossível para uma organização de programação encontrar alguns de seus erros, porque as organizações conseguem fazer isso com algum grau de sucesso. Em vez disso, implica que é mais econômico que o teste seja realizado por uma parte objetiva e independente.

Princípio 4: Inspecione minuciosamente os resultados de cada teste.

Este é provavelmente o princípio mais óbvio, mas, novamente, é algo que frequentemente é negligenciado. Vimos diversos experimentos que mostram que muitos participantes não conseguiram detectar certos erros, mesmo quando os sintomas desses erros eram claramente observáveis nas listagens de saída. Em outras palavras, erros encontrados em testes posteriores muitas vezes são perdidos nos resultados dos testes anteriores.

Princípio 5: Os casos de teste devem ser escritos para condições de entrada que sejam inválidas e inesperadas, assim como para aquelas que são válidas e esperadas.

Há uma tendência natural, ao testar um programa, de se concentrar nas condições de entrada válidas e esperadas, negligenciando as condições inválidas e inesperadas. Por exemplo, essa tendência frequentemente aparece ao testar o programa de triângulos no Capítulo 1.
Poucas pessoas, por exemplo, alimentam o programa com os números 1, 2, 5 para garantir que o programa não interprete erroneamente isso como um triângulo escaleno, em vez de um triângulo inválido. Além disso, muitos erros que são descobertos repentinamente em programas em produção surgem quando o programa é usado de uma maneira nova ou inesperada. Portanto, casos de teste representando condições de entrada inesperadas e inválidas parecem ter uma maior taxa de detecção de erros do que casos de teste para condições de entrada válidas.

Princípio 6: Examinar um programa para ver se ele não faz o que deveria fazer é apenas metade da batalha; a outra metade é verificar se o programa faz o que não deveria fazer.

Este é um corolário do princípio anterior. Os programas devem ser examinados quanto a efeitos colaterais indesejados. Por exemplo, um programa de folha de pagamento que produz os contracheques corretos ainda é um programa errôneo se ele também produzir cheques extras para empregados inexistentes ou se sobrescrever o primeiro registro do arquivo de pessoal.

Princípio 7: Evite casos de teste descartáveis, a menos que o programa seja realmente um programa descartável.
Esse problema é mais comumente observado em sistemas interativos para testar programas. Uma prática comum é sentar-se em um terminal e inventar casos de teste na hora, enviando-os pelo programa. O principal problema é que os casos de teste representam um investimento valioso que, nesse ambiente, desaparece após a conclusão dos testes. Sempre que o programa precisar ser testado novamente (por exemplo, após corrigir um erro ou fazer uma melhoria), os casos de teste precisam ser reinventados. Mais frequentemente do que não, como essa reinvenção exige uma quantidade considerável de trabalho, as pessoas tendem a evitá-la. Portanto, o reteste do programa raramente é tão rigoroso quanto o teste original, o que significa que, se a modificação causar a falha de uma parte do programa que antes funcionava, esse erro muitas vezes passa despercebido. Salvar os casos de teste e executá-los novamente após alterações em outros componentes do programa é conhecido como teste de regressão.

Princípio 8: Não planeje um esforço de testes sob a suposição tácita de que nenhum erro será encontrado.
Este é um erro comum cometido por gerentes de projeto e é um sinal do uso da definição incorreta de testes — ou seja, a suposição de que o teste é o processo de mostrar que o programa funciona corretamente. Mais uma vez, a definição de testes é o processo de executar um programa com a intenção de encontrar erros.

Princípio 9: A probabilidade da existência de mais erros em uma seção de um programa é proporcional ao número de erros já encontrados nessa seção.
Esse fenômeno é ilustrado na Figura 2.2. À primeira vista, pode parecer sem sentido, mas é um fenômeno presente em muitos programas. Por exemplo, se um programa consiste em dois módulos, classes ou sub-rotinas

A e B, e cinco erros foram encontrados no módulo A e apenas um erro foi encontrado no módulo B, e se o módulo A não foi propositalmente submetido a um teste mais rigoroso, então este princípio nos diz que a probabilidade de mais erros no módulo A é maior do que a probabilidade de mais erros no módulo B.

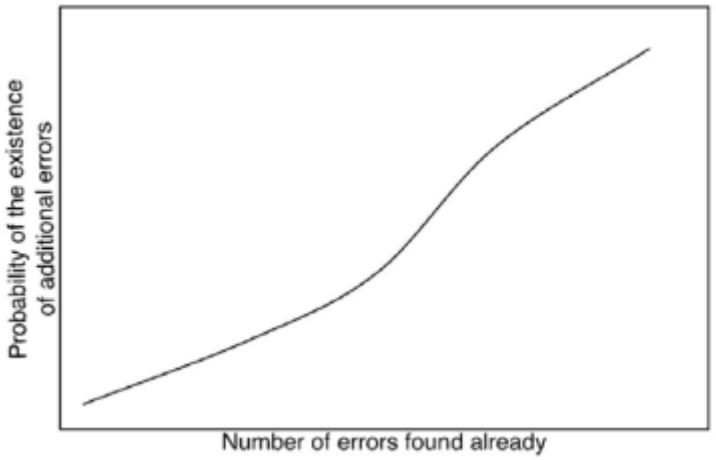

Figura 2.2: A Surpreendente Relação entre Erros Restantes/Erros Encontrados.

Outra forma de enunciar esse princípio é dizer que os erros tendem a surgir em agrupamentos e que, em um programa típico, algumas seções parecem ser muito mais propensas a erros do que outras, embora ninguém tenha fornecido uma boa explicação para o motivo disso ocorrer. Esse fenômeno é útil porque nos dá uma visão ou um retorno sobre o processo de testes. Se uma determinada seção de um programa parece ser muito mais propensa a erros do que outras seções, então esse fenômeno nos diz que, em termos de retorno sobre nosso investimento em testes, os esforços adicionais de teste devem ser concentrados nessa seção propensa a erros.

Princípio 10: Testar é uma tarefa extremamente criativa e intelectualmente desafiadora.
Provavelmente, a criatividade necessária para testar um grande programa supera a criatividade exigida para projetá-lo. Já vimos que é impossível testar um programa o suficiente para garantir a ausência total de erros. As metodologias discutidas mais adiante neste livro permitem desenvolver um conjunto razoável de casos de teste para um programa, mas essas metodologias ainda exigem uma quantidade significativa de criatividade.

Resumo
À medida que você avança na leitura deste livro, tenha em mente estes três princípios importantes de teste:

- Testar é o processo de executar um programa com a intenção de encontrar erros.
- Um bom caso de teste é aquele que tem alta probabilidade de detectar um erro ainda não descoberto.
- Um caso de teste bem-sucedido é aquele que detecta um erro ainda não descoberto.