# UNIDADE 3 – TESTES, MANUTENÇÃO E ENGENHARIA REVERSA

## MÓDULO 1 – TESTE DE *SOFTWARE*

01

## 1 - O QUE É TESTE?

Teste é uma palavra originada da língua inglesa (*test*) e significa teste, experimento, prova, comprovação. Muitas vezes este conceito faz referência às provas destinadas a avaliar conhecimentos, aptidões ou competências.

A palavra teste pode ser usada como sinônimo de exame. Os exames são bastante frequentes no âmbito educativo pelo fato de permitirem **avaliar** os conhecimentos adquiridos pelos estudantes. Os exames podem ser orais ou escritos, com perguntas de respostas abertas, em que o aluno pode responder livremente ou perguntas de respostas com múltiplas escolhas em uma lista.

Também se fazem testes para obter licença para dirigir, isto é, para tirar a carteira de habilitação. No Brasil fazemos testes sobre o código de trânsito, os quais contêm as regras para se dirigir. É feito também o teste de condução, que é o prático. Para fazer esses exames o aluno deve ter feito aulas necessárias para adquirir os conhecimentos e estar preparado. Para que lhe seja emitida a carteira, deverá superar com êxito ambos os exames. No Brasil, quem dirige sem portar carteira de motorista está cometendo uma infração de trânsito.


02

Outro teste muito comum é o teste psicológico (ou psicotécnico), que são ferramentas que permitem avaliar ou medir as características psicológicas de uma pessoa. As respostas dadas por uma pessoa são comparadas por meio de métodos estatísticos ou qualitativos com as respostas de outros indivíduos que tenham completado o mesmo teste, o que permite realizar uma classificação.

Enfim, existem testes nos mais diferentes ramos de atividades. Vimos nos exemplos anteriores que o **teste serve para avaliar algo**, podendo servir como forma de prevenção de algum tipo de problema.

A avaliação de um produto antes de ser enviado ao cliente é fundamental para a imagem do produto, pois um defeito detectado pelo cliente pode determinar o fim do produto, mesmo que ele seja considerado um bom produto, que atenda à sociedade.

Todos ocupamos, em um determinado tempo, a posição de cliente. Somos clientes no restaurante na hora da refeição, quando compramos um DVD, quando compramos um sorvete ou um chocolate. Se encontrarmos algum objeto estranho na comida do restaurante, provavelmente você não voltará mais lá. E o pior, falará com várias pessoas sobre o ocorrido. Um descuido qualquer no preparo do alimento pode trazer consequências trágicas para o restaurante.

É importante que as empresas tenham um **processo de qualidade** para seus produtos. Esse processo deverá prever testes e verificações em amostras do produto. Em se tratando de *softwares*, esta é uma regra básica.

03

## 2 – Por que testar softwares?

Cada vez mais os *software*s estão presentes na vida das pessoas, nos celulares, tabletes, bancos, carros, procedimentos médicos, entre outros. Consequentemente os *software*s estão se tornando mais complexos, devido ao surgimento de novas tecnologias. Muito provavelmente você já teve alguma experiência com um *software* que não funcionou adequadamente como o esperado.

Aplicativos de celulares que param de funcionar e travam, sites que não abrem, enfim, *software*s que não funcionam corretamente. Isso pode levar a muitos problemas e não inspiram confiança aos usuários.

Imagine um *software* usado em um hospital, que controla os medicamentos e dosagens ministradas para os pacientes que estão internados. Se esse *software* apresentar problemas no armazenamento de informações, ele poderá indicar medicamentos errados com dosagens erradas, para pessoas erradas. Isso poderia colocar em risco a vida das pessoas e obviamente abalar a reputação do hospital. Sistemas que envolvem vidas não podem ter defeitos dessa magnitude, os testes devem ser intensificados à exaustão.

Outro exemplo conhecido foi a pane no site que fazia a venda de ingressos para o show da Madonna no Brasil, devido ao grande número de acessos simultâneos, causou atraso nas respostas das compras dos clientes e gerou informações múltiplas que o sistema não conseguiu interpretar. Problemas como esse geram custos extras ao projeto, pois há a necessidade de ações de contorno. No caso houve publicações

de pedido de desculpa e aviso às pessoas que compraram para validarem suas compras por telefone ou pessoalmente, recebendo um novo ingresso. Vejam os custos adicionais que o projeto do show teve, por causa de um erro no sistema.

04

Para evitar problemas como os que citamos são necessários investimentos em **testes**. Testes nos sistemas e em documentações reduzem os riscos da ocorrência de defeitos do *software* no ambiente de produção, contribuindo assim para a qualidade dos sistemas. É sempre bom frisar que quanto mais cedo os defeitos forem encontrados, o custo de correção é menor em relação ao encontrado nas fases posteriores.

**Mas o que é teste de *software*?**

Teste de *software* é a atividade realizada ao longo do desenvolvimento, que verifica se o *software* e seus componentes atendem às suas especificações. Esses testes são uma garantia de que o sistema foi desenvolvido corretamente e que este possui as características de qualidade esperadas.

Em termos mais simples, podemos dizer que o teste de *software* é o processo de execução de um produto para determinar se ele atingiu suas especificações e se funcionou corretamente no ambiente para o qual foi projetado.


Como o objetivo principal do teste é **evidenciar eventuais falhas** em um produto, para que sejam corrigidas pela equipe de desenvolvimento antes da entrega final, é comum dizer que esta atividade tem caráter “destrutivo”, e não “construtivo”, pois tem em vista o aumento da confiança de um produto por meio da exposição de seus problemas, porém antes de sua entrega ao usuário final.

05

O grande desafio das empresas é produzir *software*s com qualidade, em um curto espaço de tempo, com baixo custo e atender às expectativas do cliente com o produto desenvolvido, ou seja, atender aos **requisitos** impostos pelo cliente. Realizar testes dentro do processo de desenvolvimento é de grande importância para o projeto e para a empresa, pois os processos já estão definidos e acompanhados, independente da metodologia adotada.

Os testes têm por finalidade agregar qualidade ao produto podendo também fazer uma medição desta qualidade em relação aos defeitos encontrados.


Não necessariamente poucos defeitos deixam o *software* mais confiável, pois fica a dúvida se os defeitos não encontrados aparecerão mais à frente no projeto. Mas com os testes é possível também antecipar a descoberta de falhas e incompatibilidades, reduzindo assim o custo do projeto.

As tarefas de teste então definidas no **cronograma do projeto de desenvolvimento do *software*** para se obter um nível aceitável da aplicação quando entregue em produção para utilização do cliente. Importante dar mais foco nos testes nos pontos críticos do sistema, os quais são de grande importância para o negócio e que caso não sejam tratados adequadamente, podem gerar prejuízos inestimáveis quando o *software* estiver na fase de produção. A falta de controle ou excesso de preciosismo por parte da equipe de testes pode tornar os testes onerosos para o projeto e caso a relação custo x benefício não seja interessante para a empresa, pode-se considerar o momento ideal para interrupção dos testes.

No sentido de se tornarem mais competitivas, as organizações de *software* vêm investindo cada vez mais na qualidade de seus produtos e serviços de *software*. A qualidade de *software* está diretamente relacionada a um gerenciamento rigoroso de requisitos, à gerência efetiva de projetos e a um processo de desenvolvimento bem definido, gerenciado e com melhoria contínua. Atividades de verificação e uso de métricas para controle de projetos e processo também estão inseridas nesse contexto, contribuindo para tomadas de decisão e para antecipação de problemas.

06

### 2.1 - Conceitos importantes sobre teste

Testar *software* não é somente executá-lo com a intenção de encontrar erros, existem várias outras atividades envolvidas: planejamento, controle, checagem dos resultados, avaliação de conclusão dos testes, geração de relatórios como também a revisão dos documentos, dentre outros.

Há um conceito que se confunde com a atividade de teste: a atividade de **depuração**.

A **atividade de teste** pode demonstrar falhas que são causadas por defeitos, enquanto a **depuração** é uma atividade de desenvolvimento que repara o código e checa se os defeitos foram corrigidos

corretamente para, então, ser feito um teste de confirmação por um testador com a intenção de certificar se o mesmo foi eliminado.

Outros conceitos chaves utilizados em testes são listados a seguir:

- **Defeito**;
- **Erro**;
- **Falha**;
- **Bug**;
- ***Testware***;
- **Caso de Teste**;
- **Script de Teste**.

**Defeito**
Pode ser definido como o resultado de um erro encontrado em um código ou em um documento, e também um problema ocasionado no hardware, por natureza conhecida ou desconhecida.

**Erro**
Pode ser definido como um engano cometido por pessoas.

**Falha**
Pode ser definido como o resultado ou manisfestação de um ou mais defeitos.

**Bug**
Pode ser definido como um erro de lógica na programação de um determinado *software*.

***Testware***
É toda a documentação de teste.

**Caso de Teste**
É uma descrição de um teste a ser executada. Um ou mais casos de teste costumam estar relacionados a um caso de uso.

**Script de Teste**
É a automação da execução de um caso de teste.

07

## 3 - Estratégia de testes

Toda organização possui processos formais ou informais, que são implementados para desenvolvimento de *softwares*. Esse processo pode incluir tanto produtos finais, que são usados pelos clientes, como *software* executável, manual do usuário, documento de requisitos etc..

Segundo Presmam, **a estratégia de teste de *software*** fornece um roteiro que descreve os passos a serem executados como parte do teste, definindo **quando** esses passos são planejados e então executados, e **quanto trabalho**, tempo e recursos serão necessários. A execução de todo o teste pode ser feita de forma combinada, conforme definida na estratégia.

Geralmente define-se na estratégia que os testes serão realizados pela equipe de desenvolvimento ou por engenheiros de *software*. No entanto, é ideal que pelo menos os testes de sistemas sejam feitos por uma equipe de testes independente.

É importante que os grandes sistemas e programas também sejam testados por outras pessoas que não os seus desenvolvedores, e que tais pessoas sejam especialistas em testes.

Várias técnicas são utilizadas para identificar defeitos nos produtos de trabalho. Esses defeitos são eliminados através de retrabalho, que têm efeito imediato na produtividade do projeto. Defeitos também são encontrados em atividades de teste e podem ser analisados, a fim de se identificar suas causas. A partir dessa análise, lições aprendidas podem ser usadas para criar futuros produtos e prevenir futuros defeitos e, dessa forma, ter impacto positivo na qualidade do produto e na produtividade do projeto.

08

Na estratégia de execução dos testes temos que nos preocupar com a responsabilidade, que é um problema não surgir durante o funcionamento do sistema. Podemos definir como estratégia algumas **ações de prevenção**, como por exemplo:

- **Treinamento**

Treinar e preparar pessoas na equipe para realizar os testes nos *softwares* durante todo o ciclo de vida do projeto.

- **Planejamento**

Programar os testes de acordo com a evolução da construção do projeto.

- **Atuação do grupo de garantia da qualidade**

Solicitar à equipe da qualidade da empresa que realize inspeções e auditorias da qualidade, a fim de manter a garantia de que o processo está sendo seguido de forma correta.

- **Uso de lições aprendidas**

Visitar a base histórica dos testes realizados nos aplicativos da empresa, a fim de capturar as melhores práticas de uso das ferramentas e experiências já realizadas para determinados códigos e linguagens de desenvolvimento.

- **Melhoria de processo**

A partir das lições aprendidas é possível elaborar um plano de melhoria do processo de teste, verificando também a necessidade de aquisição de ferramentas novas do mercado.

**09**

Além da prevenção temos também na elaboração da estratégia dos testes a utilização de **técnicas de detecção de erros**, como por exemplo:

- **Compilação**

No ato da compilação do código, o compilador já identifica se o programa está ou não com algum tipo de erro; o compilador transforma a linguagem de programação em linguagem de máquina.

- **Revisão por pares (*peer reviews*)**

A revisão por pares é um tipo de avaliação de software em que um produto de trabalho (documento, código ou outro) é examinado pelo seu autor e um ou mais colegas, a fim de avaliar o seu conteúdo técnico e de qualidade. Não é necessário que sejam pessoas do mesmo segmento, mas pessoas que buscam o mesmo resultado. O resultado geralmente é o produto sem defeitos.

- **Teste e simulação**

Toda vez que tiver um produto construído deve ocorrer os testes e simulações antes de entregar para o cliente. Isso é fundamental, todo produto deve ser testado antes da entrega final.

- **Auditorias**

As auditorias fazem parte do controle da Qualidade e são importantes para verificar se o processo de desenvolvimento está sendo seguido conforme previsto no planejamento, pois se presume que, quando o processo ocorre conforme o estabelecido, há grande chance de o produto ter o resultado esperado. Durante a auditoria, ao se constatar algum desvio, é gerado um relatório para correção. Como já estudado anteriormente, o processo define o caminho, o meio e os papéis das pessoas que irão executar. O processo antes de ser implantado é estudado e definido, geralmente, pela Qualidade com o apoio da organização.

- **Verificações**

As verificações do processo de qualidade analisam o processo de teste em um determinado projeto, se está de acordo ou não com o processo definido. Algumas empresas aprofundam essas verificações, observando detalhes dos testes como, por exemplo, conteúdo de documentos. A documentação analisada faz parte do processo de teste, como o Plano de teste, Roteiros de testes, casos de testes, dentre outros. Os documentos mostram como serão realizados os testes e o que será testado, além das ferramentas que serão utilizadas. A verificação é muito semelhante à auditoria, só que tem objetivo de correção ou serve, em alguns casos, como uma preparação para a auditoria.

**10**

A disciplina de **testes de *software*** tem ganhado grande reconhecimento nos últimos anos por parte da comunidade técnica. Com o crescimento da complexidade dos projetos e, muitas vezes, o grande número de envolvidos no desenvolvimento de um *software*, torna-se importante uma formalização da nomenclatura, processos e artefatos utilizados para garantir a qualidade de um *software* através dos testes.

Presman afirma que **teste** é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente. Por essa razão, deverá ser definido para o processo de *software* um **modelo** para o teste, um conjunto de etapas no qual se podem colocar técnicas específicas de projeto de caso de testes e métodos de teste.

Muitas estratégias de testes de *software* já foram propostas no mercado. O importante é que, para executar um teste eficaz, deve-se proceder as revisões técnicas, nas quais os erros podem ser eliminados antes do começo do teste. O teste começa no nível de componente e progride em direção à integração do sistema computacional como um todo, ou seja, da menor parte do código, passando para funcionalidades e chegando ao todo, que é o projeto construído. Diferentes técnicas de teste são apropriadas para diferentes abordagens de engenharia de *software* e em diferentes pontos no tempo. O teste, como dito anteriormente, é feito pelo desenvolvedor do *software* e por grupo independente de teste.

11

### 3.1 - Processo de testes e ferramentas

Dentro do processo de desenvolvimento de *software* existem ciclos de testes. O ciclo de teste é caracterizado por atividades que visam garantir a qualidade do produto final. A busca pelo aumento da qualidade ressalta a importância dos testes no desenvolvimento de *software*. Porém, teste de *software* não é uma atividade trivial, exige conhecimentos, procedimentos e infraestrutura específicos, como **ferramentas apropriadas** para agilizar o andamento dos testes.

No ciclo de desenvolvimento de *software*s, a realização de testes tem espaço desde a fase de *design* até o lançamento do produto. Eles conferem confiabilidade ao *software*, reorientam o desenvolvimento do *design* e do código, e poupam gastos desnecessários, quando detectam erros nas fases iniciais do desenvolvimento de um *software*.

A necessidade e a importância dos testes aumentam de acordo com o tipo de uso que o *software* terá – os que podem causar danos à vida humana ou levar a grandes perdas financeiras são críticos e devem ser disponibilizados para uso apenas após um processo de testes criterioso. Além disso, quanto mais precoce a detecção de falhas ocorre, menores os gastos do projeto com reparos e replanejamento.

É por esse motivo que a utilização de ferramentas de suporte a testes tem se tornado uma regra no desenvolvimento de *software*. A IBM Rational possui um conjunto de ferramentas para gerenciar, realizar e controlar os testes e é utilizado em grandes companhias, que estão interessadas em melhorar o processo de testes como um todo.

Os testes também fazem parte dos procedimentos seguidos para garantir a qualidade do processo de desenvolvimento de *software*s, através de certificações concedidas por organizações que avaliam o processo considerando modelos de qualidade, como o CMMI (Capability Maturity Model Integration) e a ISO-12207.

**IBM Rational**

As aplicações da IBM Rational incluem:

- ferramenta para desenvolvimento de casos de teste como o Rational Functional Tester,
- ferramenta de desenvolvimento com suporte a teste como o Rational *Software* Architect,

ferramenta para o gerenciamento de ciclo de vida de testes e defeitos como o Rational Team Concert e o Rational Quality Manager, entre outros.

12

## 4 - Testes no mundo

Nos últimos anos vêm surgindo instituições e consórcios que visam delimitar o teste de *software* como uma área de conhecimento com vida própria. O International *Software* Testing Qualifications Board (ISTQB) é uma das instituições mais respeitadas da área de teste de *software*, pois propõe uma uniformização dos conceitos e nomenclatura da disciplina. Grandes corporações, inclusive a IBM, têm boa parte de seus profissionais certificados pelos exames da ISTQB e processos de desenvolvimento que utilizam o modelo ISTQB em seus artefatos e relatórios de testes de *software*. O ISTQB possui ramificações no Brasil (BSTQB), Estados Unidos (ASTQB) e Inglaterra (ISEB).

### 4.1 - Plano de teste

Para que os testes suportem todos os aspectos mencionados do desenvolvimento de *software*s, é preciso que eles sejam projetados, planejados e realizados de acordo com o tipo de *software* e as necessidades que ele irá atender. Assim a primeira etapa para a realização dos testes é criar o **plano de teste**.

O plano de teste consiste em um documento contendo uma abordagem sistemática para o teste de *software*. Ele geralmente constitui numa modelagem detalhada do fluxo de trabalho durante o processo.

13

Um plano de teste tem o papel semelhante ao de um mapa. E como um mapa, você terá uma fonte de informação similar, onde conhecerá seus objetivos, aonde quer chegar e terá a certeza de ter alcançado sua meta.

Perceba que o planejamento é de suma importância para monitorar a execução de atividades, sendo também importante conhecer o papel dos riscos no planejamento, bem como diferenciar estratégias de planos.

O plano pode englobar três **atividades** principais:

1. Definir um **cronograma de atividades** estabelecendo as atividades que devem ser realizadas, as etapas a serem seguidas e a ordem de execução;
2. Fazer **alocação de recursos** definindo quem realiza as atividades e quais ferramentas serão utilizadas;
3. Definir **marcos de projeto** estabelecendo os marcos a serem alcançados, os marcos são os objetivos estabelecidos.

O plano tem a atividade de monitoração ou supervisão, que visa avaliar se o progresso que tem sido alcançado está em conformidade com o que foi estabelecido no planejamento. Com isso podemos responder a pergunta: ***como está indo o projeto***?

Outros documentos muito provavelmente deverão ser construídos antes do plano de testes, como por exemplo, o documento de requisitos.

14

**4.2 – Teste: oportunidade de ouro**


Os testes representam a última oportunidade de detectar erros antes de o *software* ser entregue aos usuários, ou seja, uma oportunidade de ouro de tirar o máximo de problemas que poderão ocorrer antes de o usuário utilizar o *software*.

A atividade de testes pode ser feita de forma manual ou automática e tem por **objetivos**:

- Produzir casos de teste que tenham elevadas probabilidades de revelar um erro ainda não descoberto, com uma quantidade mínima de tempo e esforço;
- Comparar o resultado dos testes com os resultados esperados, a fim de produzir uma indicação da qualidade e da confiabilidade do *software*. Quando há diferenças, inicia-se um processo de **depuração** para descobrir a causa.

A realização de testes não consegue mostrar ou provar que um *software* ou programa está correto. O máximo que os testes de um *software* conseguem provar é que ele contém **defeitos** ou **erros**. Quando os testes realizados com um determinado *software* não encontram erros, haverá sempre duas possibilidades:

- A qualidade do *software* é aceitável;
- Os testes foram inadequados.

A segunda afirmação parece contradizer o significado dos testes, mas está correta. Imagine que o *software* passou por todos os testes e durante a produção ocorre um problema qualquer. Logo uma das possibilidades é que o *software* não teve os testes realizados adequadamente. Então, seguindo essa linha, quanto mais erros forem encontrados nos testes, em tempo de construção, é melhor para o sistema.

15

## Resumo

Os testes estão presentes em vários produtos que utilizamos: a carteira de motorista é um exemplo. Testes são feitos em linha de produção de carros e testes mais específicos ajudam os alunos a decidirem o futuro profissional.

No mesmo contexto temos os testes de *software*, que tentam garantir *software*s mais confiáveis antes da entrega para o cliente. Para isso é importante que se tenha uma estratégia para a execução dos testes do *software*.

Teste de *software* é a atividade realizada ao longo do desenvolvimento, que verifica se o *software* e seus componentes atendem às suas especificações. Esses testes são uma garantia de que o sistema foi desenvolvido corretamente e que este possui as características de qualidade esperadas.

A atividade de teste pode demonstrar falhas que são causadas por defeitos, enquanto a depuração é uma atividade de desenvolvimento que repara o código e checa se os defeitos foram corrigidos corretamente para, então, ser feito um teste de confirmação por um testador com a intenção de certificar se o mesmo foi eliminado.

Para executar o teste com melhor efetividade é fundamental que a empresa tenha um plano de teste com processos definidos e utilize ferramentas de apoio. Também é importante criar um ciclo de teste, que é caracterizado por atividades que visam garantir a qualidade do produto final.

O teste é a última oportunidade de detectar erros antes de colocar o sistema em produção, por isso não pode relaxar nesse momento importante. O que não for detectado poderá ocorrer na hora da execução pelo usuário, aí pode gerar uma desconfiança sobre o *software* disponibilizado.

## UNIDADE 3 – TESTES, MANUTENÇÃO E ENGENHARIA REVERSA
## MÓDULO 2 – CONCEITOS IMPORTANTES DE TESTES

**01**

## 1 - AMBIENTES DE TESTES

Para que realizemos os testes no projeto ou no sistema temos que ter um local adequado para essa finalidade. As empresas utilizam os ambientes de testes para realizar todos os testes necessários antes da entrega do produto.

Os ambientes de testes devem ser definidos pelo nível de testes a ser executado, ou seja, quanto maior o nível, mais o ambiente de teste deverá ser capaz de reproduzir as características do ambiente de produção.

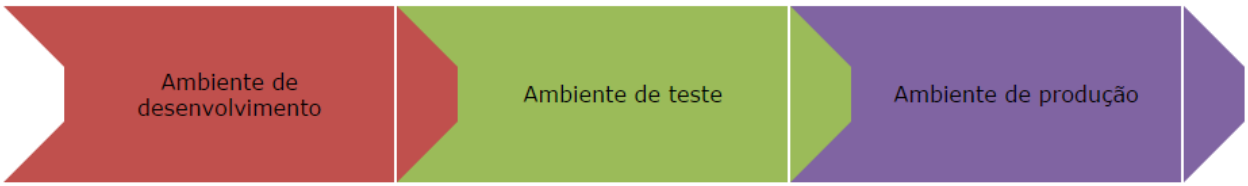
O ambiente de teste é o ambiente onde os clientes realizarão os seus testes, no sentido de homologar a versão do sistema disponibilizada pela equipe de desenvolvimento.

É possível ter mais de um ambiente para teste, o que vai depender da criticidade do sistema. Geralmente, sistemas bancários possuem mais de quatro ambientes para testes. Para esses sistemas, que são críticos, deve-se executar o máximo de testes possíveis, ou seja, um nível elevado de testes.

Sistemas menos importantes para a organização podem ter testes mais brandos. Tudo depende das necessidades e importância do projeto.

O ambiente de testes deve ser isolado, com processamento independente e características similares ao ambiente de desenvolvimento e produção e deve ser restrito à equipe de testes para garantir a integridade dos testes realizados.

Veremos os níveis ou etapas de testes mais para frente nesse módulo.




02

### 1.2 - Técnica de teste

A técnica é o processo que assegura o funcionamento correto de alguns aspectos do *software* ou de uma unidade do *software*, que significa "como testar".

Existem dois tipos de técnicas de teste de *software*: a Estrutural e a Funcional.

**a) Estrutural**

Também conhecida como técnica de **Caixa Branca**, essa técnica tende a revelar erros que ocorrem durante a codificação do programa.

Essa técnica não determina o funcionamento correto da aplicação, mas investiga o funcionamento do *software* e analisa sua estrutura interna, por isso também é chamado de caixa-branca ou de vidro.

Os testes podem se restringir a componentes específicos ou abranger o *software* como um todo, mas a abordagem depende dos processos executados pelo *software*. Uma prática comum é escolher processos críticos do *software* e executá-los de várias maneiras diferentes, com o objetivo de detectar falhas e medir sua performance em cenários distintos. Esse tipo de técnica também é útil em todos os níveis de teste, mas com uma abordagem diferente dependendo do nível. Para testes de **componentes e integração**, a estrutura do *software* é o foco; em testes de **sistema e aceitação**, a estrutura relativa ao uso do *software* (como menus e componentes principais) se torna o alvo dos testes.

A IBM possui, dentro da suite Rational, a ferramenta Rational Purify que é um depurador em tempo de execução, baseada na técnica de análise estática de código-fonte. Além disso, algumas funcionalidades que viabilizam testes do tipo caixa-branca podem ser encontradas embutidas no Rational *Software* Architect e no Rational Application Developer.

**análise estática de código-fonte**
A análise do código fonte é um método de depuração de programa feito por meio da análise do código sem executar o programa. O processo proporciona uma compreensão da estrutura do código, e pode ajudar a assegurar que o código adere às normas da empresa. Ferramentas automatizadas podem ajudar programadores e desenvolvedores na realização de uma análise estática. O processo de análise de código por inspeção visual (por meio de olhar para a impressão, por exemplo), sem o auxílio de ferramentas automatizadas, é às vezes chamado de entendimento ou compreensão do programa.

03

**c) Funcional**

Também conhecida como técnica de **Caixa Preta**, essa técnica garante o cumprimento dos requisitos pelo sistema.

A abordagem funcional concentra-se nas interfaces do *software* e visa mostrar que as entradas são aceitas, as saídas são as esperadas e a integridade dos dados é mantida. Aplica-se, principalmente, aos testes de validação, de sistemas e aceitação, mas pode ser também usada com os testes unitários.

O desenvolvimento dos testes que se baseia nas especificações do *software* no teste caixa-preta foca em sua funcionalidade e verifica se ele cumpre o que foi proposto. Geralmente são definidos dados de entrada e os esperados dados de saída. Ao final de cada ciclo de testes, os dados de saída são comparados com aqueles definidos no Plano de Testes e a aprovação depende da paridade entre eles.

A técnica da caixa-preta é útil para todos os níveis de teste, desde os de componentes até os de sistema e aceitação, pois é fundamental que o *software* atenda suas especificações. Um exemplo de ferramenta de suporte ao teste de caixa-preta é o Rational Functional Tester (RFT).

04

## 2 - Tipos e níveis de testes

### 2.1 – Tipos

Os tipos de testes definem o que será testado no sistema e estes estão divididos de acordo com seu objetivo particular. Podemos destacar os seguintes tipos:

| TIPO DE TESTE | OBJETIVO |
|---|---|
| **Teste de Requisitos** | É um teste caixa preta que verifica se o sistema é executado conforme o que foi especificado. São realizados através da criação de condições de testes e cheklists de funcionalidades. |
| **Teste de Regressão** | é um teste caixa preta que testa se algo mudou em relação ao que já estava funcionando corretamente, ou seja, é voltar a testar segmentos já testados após uma mudança em outra parte do *software*. Os testes de regressão devem ser feitos tanto no *software* quanto na documentação. |
| **Teste de Tratamento de Erros** | é um teste caixa preta que determina a capacidade do *software* de tratar transações incorretas. Esse tipo de teste requer que o testador pense negativamente e conduza testes como: entrar com dados cadastrais impróprios, tais como preços, salários, etc., para determinar o comportamento do *software* na gestão desses erros. Produzir um conjunto de transações contendo erros e introduzi-los no sistema para determinar se este administra os problemas. |
| **Teste de Suporte Manual** | É um teste caixa preta que verifica se os procedimentos de suporte manual estão documentados e completos. Verifica e determina se as responsabilidades pelo suporte manual foram estabelecidas. |
| **Teste de Interconexão** | É um teste caixa preta que garante que a interconexão entre os *software*s de aplicação funcione corretamente, pois, *software*s de aplicação costumam estar conectados com outros *software*s de mesmo tipo. |
| **Teste de Controle** | É um teste caixa preta que assegura que o processamento seja realizado conforme sua intenção. Entre os controles estão a validação de dados, a integridade dos arquivos, as trilhas de auditoria, o backup e a recuperação, a documentação, entre outros. |
| **Teste Paralelo** | É um teste caixa preta que compara os resultados do sistema atual com a versão anterior determinando se os resultados do novo sistema são consistentes com o processamento do antigo sistema ou da antiga versão. O teste paralelo exige que os mesmos dados de entrada rodem em duas versões da mesma aplicação. Por exemplo: caso a versão mude e os requisitos não, os dados de saída das duas versões devem ser iguais. |
| **Teste de Execução** | é um teste caixa branca que verifica os tempos de resposta, de processamento e o desempenho (performance), avaliando o comportamento do *software* no ambiente de produção e verificando se as premissas de desempenho são atendidas. Em um sistema que possui dez módulos diferentes e que foi desenvolvido por equipes diferentes, o teste de execução avalia o sistema como um todo, é como se o teste de execução fosse um “play” no sistema. |
| **Teste de Estresse** | é um teste caixa branca que avalia o comportamento do *software* sob condições críticas, tais como restrições significativas de memória, espaço em disco, etc., ou seja, coloca o *software* sob condições mínimas de operação. |
| **Teste de Recuperação** | é um teste caixa branca que a recuperação é a capacidade de reiniciar |

| | |
|---|---|
| | operações após a perda da integridade de uma aplicação como, por exemplo: Ao desligar o computador, queda de energia elétrica, entre outros. O teste de recuperação garante a continuidade das operações após um desastre. |
| **Teste de Operação** | É um teste caixa branca que avalia o processo e sua execução, desenhado para estabelecer se o sistema é executável durante a operação normal. É um tipo de teste muito específico, depende do *software* a ser testado um exemplo é o *software* de "Call Center". |
| **Teste de Conformidade** | é um teste caixa branca que verifica se o *software* foi desenvolvido de acordo com padrões, normas, procedimentos e guias de TI. |
| **Teste de Segurança** | é um teste caixa branca que avalia a adequação dos procedimentos de proteção e as contra medidas projetadas, para garantir a confidencialidade das informações e a proteção dos dados contra o acesso não autorizado de terceiros. |

Muitos outros tipos de testes são aceitáveis, contudo é fundamental ter em mente os requisitos funcionais e não funcionais (garantia e utilidade) do negócio, para definir exatamente o nível de testes que se pretende estabelecer para uma determinada aplicação. Testar demais é tão ineficiente quanto testar pouco.

05

### 2.2 - Níveis ou estágios de Teste

É a dimensão do teste que determina a fase do desenvolvimento que se aplica um determinado teste, ou seja, *quando* deverei testar.

Os estágios de teste fornecem a indicação sobre o foco do teste e os tipos de problemas a serem encontrados. O conceito "níveis de teste" está relacionado com o modelo a seguir, que representa quando as atividades de teste acontecem em decorrência do ciclo de vida do *software*.

Os níveis de teste estão listados a seguir

| | |
|---|---|
| • **Teste de Unidade ou teste Unitário** | É aplicado aos menores componentes de código; é feito pelos programadores e testa as unidades individuais: funções, objetos e componentes. Tem como objetivo testar individualmente cada um dos componentes, como programas ou módulos, procurando garantir que funcionem adequadamente. |
| • **Teste de Integração ou Iteração** | É feito ao término de cada iteração para validar a execução das funções referentes aos casos de uso, sendo feito normalmente pelo analista de sistemas. Tem como objetivo testar o relacionamento entre as diversas unidades integradas. Em outras palavras, garantir que a interface entre os módulos funcione adequadamente, pois não há garantias de que unidades testadas em separado funcionarão em conjunto. |
| • **Teste de Sistema** | Executa o sistema como um todo para validar a execução das funções acompanhando cenários elaborados, que são os casos de teste, por um analista de testes em um ambiente de testes. Tem por objetivo primordial colocar completamente à prova todo o sistema, baseado em um computador. Em outras palavras, testa a integração do *software* com o ambiente operacional, hardware, pessoas e dados reais. |
| • **Teste de Aceitação** | É feito antes da implantação do *software*, o cliente é quem executa este tipo de teste no ambiente de homologação, tem como objetivo verificar se o *software* está pronto para ser utilizado pelos usuários finais. O fato é que o desenvolvedor nunca conseguirá prever como o usuário realmente usará um *software* numa situação real. |

Os **testes de aceitação** podem ser de dois tipos:

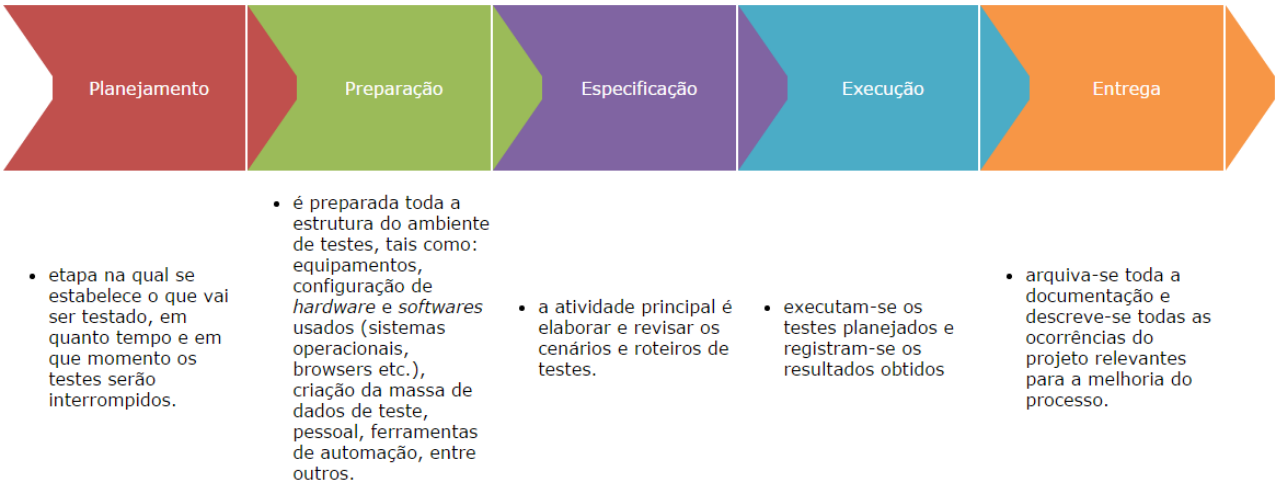
- **Testes alfa**: são feitos por um determinado cliente, geralmente nas instalações do desenvolvedor, que observa e registra os erros ou problemas.

- **Testes beta**: são realizados por possíveis clientes, em suas próprias instalações, sem a supervisão do desenvolvedor. Cada cliente relata os problemas encontrados ao desenvolvedor posteriormente.

07

## 3 - Ciclo de Vida de Testes

O ciclo de vida de testes é composto pelas seguintes etapas:




Para se obter resultados positivos nos projetos de testes é necessário que este inicie desde a especificação dos requisitos do sistema a ser implementado, ou seja, tão logo dê início o projeto de desenvolvimento do *software*, inicia-se também o projeto de testes de *software*.

08

### 3.1 - O que é um Caso de Teste?

A especificação de casos de teste é um documento que inclui a descrição do ambiente de testes, as pré e pós condições do teste, a descrição do teste e seus objetivos, as entradas e saídas esperadas e as dependências entre os testes.

Ele é um documento bem detalhado, que poderá ser usado como referência em todas as etapas do teste e pode sofrer atualizações para melhor se adequar ao design de testes e ao comportamento que o *software* apresentar durante os testes.

Por fim, a especificação dos procedimentos de teste é definida pelo padrão IEEE 829, como o documento que reúne o propósito dos testes, seus requerimentos específicos e finalmente o passo a passo detalhado de realização dos testes. É um documento ainda mais técnico que os demais, que

descreve com detalhes e em sequência os passos a serem seguidos em cada caso de teste, seja ele manual ou automatizado. Ele também é chamado de **script de testes** e faz parte do modelo de Plano de Teste.

Vale lembrar que o IEEE (Institute of Electrical and Electronic Engineers), fundação organizacional sem fins lucrativos, é responsável por promover o conhecimento nas áreas de engenharia elétrica, eletrônica e computação, define padrões para diversas áreas e práticas presentes na engenharia de *software*.

09

### 3.2 - Processo de Teste

O processo de testes de *software* representa uma estrutura das etapas, atividades, artefatos, papéis e responsabilidades buscando padronizar os trabalhos para um melhor controle dos projetos de testes.

O objetivo de um processo de teste (com metodologia própria, ciclo de vida etc.) é minimizar os riscos causados por defeitos provenientes do processo de desenvolvimento como também a redução de custos de correção de defeitos, pois o custo do *software* tende a ser menor quando o *software* é bem testado.

Os principais participantes no processo de testes são:

- **Gerente de Teste**;
- **Líder de Teste**;
- **Analista de Teste**;
- **Arquiteto de Teste**;
- **Testador**;
- **Automatizador**.

Uma pessoa pode assumir mais de um dos papéis citados acima como, por exemplo, um testador pode exercer o papel de um automatizador de testes também.

**Gerente de Teste**
Tem como papel defender a qualidade dos testes, planejar e gerenciar os recursos e resolver os problemas que representam obstáculos ao esforço de teste.

**Líder de Teste**
Pessoa responsável pela liderança de um projeto de teste específico, normalmente relacionado a um projeto de desenvolvimento, seja um projeto novo ou uma manutenção.

**Analista de Teste**
Elabora e modela os casos e roteiros de testes. Deve focar seu trabalho nas técnicas de teste adequadas à fase de teste trabalhada.

**Arquiteto de Teste**
É responsável por montar a infraestrutura de testes como: ambiente, ferramentas, capacitação da equipe, entre outros.

**Testador**
Executa os testes, o mesmo deve observar as condições de teste e respectivos passos de teste documentados pelo analista de teste e evidenciar os resultados de execução.

**Automatizador**
Tem como papel automatizar as situações de teste em ferramentas observando as condições de teste documentadas pelo analista de teste e automatizar a execução desses testes na ferramenta utilizada. Normalmente são gerados *scripts* de teste que permitam a execução de ciclos de teste sempre que se julgar necessário.

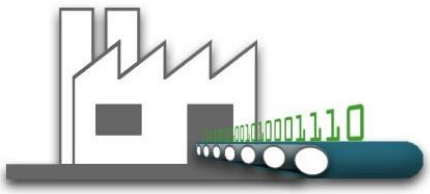
10

### 3.3 - Fábrica de Teste

A fábrica de teste é um aprimoramento do modelo de fábrica de *software*. A criação das fábricas de *software* incrementou a velocidade de desenvolvimento e manutenção de aplicações, mas não resolveu questões relacionadas à qualidade dos produtos entregues.

A fábrica de teste é uma estrutura independente de profissionais com alta especialização e capacitação em processos e ferramentas de testes de *software* e tem como objetivo medir e avaliar a qualidade dos sistemas que estão sendo modificados, adaptados e construídos.

O papel de um fábrica de testes é aplicar o maior volume de testes simulando os principais cenários de negócios, avaliando antecipadamente a conformidade do produto com as especificações do cliente.



11

### 3.4 - Ferramentas de Teste

O processo de testes pode ser automatizado, através do uso de ferramentas específicas para essa atividade. Alguns tipos de ferramentas de apoio aos testes são descritos a seguir:

- **Ferramenta de geração de massa de dados**

Geram dados para serem usados em testes. A geração dos dados é frequentemente baseada em regras de formação definidas pelos casos de teste.

- **Ferramenta de teste de API**

Testa método a método de uma classe, a partir de uma série de combinações de parâmetros. Utiliza a abordagem caixa preta para cada método.

- **Ferramenta de teste de GUI (Graphic User Interface)**

Grava em um script a execução da interface gráfica (cliques do mouse e entradas de teclado) e repete a entrada quantas vezes forem necessárias. O script gerado pode ser modificado através do uso de uma linguagem própria de script.

- **Ferramenta de teste de cobertura**

Após a execução da aplicação, indica quais os trechos do código foram ou não executados, bem como o número de vezes que determinado método/trecho foi executado. Utiliza a abordagem caixa branca.

- **Ferramenta de teste de carga e stress**

Simula acessos simultâneos a uma aplicação multiusuário, bem como o envio e recuperação de altos volumes de dados.

- **Ferramenta de teste de desempenho/gargalos**

Analisa o desempenho do *software*, quando em execução, e detecta potenciais pontos de gargalo no código fonte.

12

### 3.5 – O fator experiência na execução dos testes

O desenvolvimento dos testes com base na experiência usa o conhecimento e a intuição do engenheiro de testes e usuários experientes para a determinação das condições de teste e a criação de casos de testes.

Engenheiros de testes experientes na área do *software* têm mais facilidade em imaginar cenários em que o produto possa apresentar vulnerabilidade. E o envolvimento de usuários enriquece o Plano de Testes, pois eles têm outra perspectiva do *software* e das necessidades que ele deve atender.

Para testes de sistemas de baixo risco essa técnica costuma ser a única usada. Nos demais casos, ela é complementar às demais, e muito útil quando não há uma especificação de *software* bem definida. Neste caso é possível valer-se de ferramentas de gerenciamento de ciclo de vida e defeitos, como o Rational Team Concert e o Rational Quality Manager, que fornecem informações valiosas sobre o histórico do desenvolvimento do projeto. Essas informações podem ser utilizadas para a criação de casos de teste focados nas áreas críticas onde houve maior volume de defeitos, por exemplo.

É possível também fazer comparações entre versões diferentes de um mesmo *software* sob a perspectiva do desempenho, utilizando o Rational Performance Tester.

13

**Resumo**

Para que realizemos os testes no projeto ou no sistema necessitamos de um local adequado. O ambiente de teste é o ambiente onde os clientes realizarão os seus testes, no sentido de homologar a versão do sistema disponibilizada pela equipe de desenvolvimento.

O ciclo de vida do projeto contém também um ciclo de testes, que possuiu várias etapas como: Planejamento, Preparação, Especificação, Execução e Entrega. Para a TI temos diversos testes, teste de requisitos a testes de aceitação, é uma grande carga de testes, tudo para que o cliente não tenha problemas na utilização.

Papeis e responsabilidades definem as atividades de uma equipe que realiza os testes, assim como uma fábrica de testes, que trabalha especificamente testando o *software* de outra empresa que construiu os códigos. Os principais envolvidos nessas atividades são o Gerente de Teste, Analista de teste, dentre outros.

Para realização dos testes podemos utilizar ferramentas específicas para cada finalidade, para que se possa agilizar e apoiar os testes. Ferramentas essas específicas para testes de cobertura, carga, desempenho, dentre outras.

O importante é que os engenheiros de testes experientes na área do *software* têm mais facilidade em imaginar cenários em que o produto possa apresentar vulnerabilidade, proporcionando assim, uma redução nos problemas que podem ocorrer futuramente.

# UNIDADE 3 – TESTES, MANUTENÇÃO E ENGENHARIA REVERSA
# MÓDULO 3 – MANUTENÇÃO NA *SOFTWARE* NA ENGENHARIA DE *SOFTWARE*

01

## 1 - CRISE DO SOFTWARE

O termo **crise do *software*** surgiu nos anos 70, quando a engenharia de *software* praticamente inexistia. Esse termo estava relacionado às dificuldades enfrentadas no desenvolvimento de *software*, inerentes ao aumento das demandas e da complexidade destas, aliado à inexistência de técnicas apropriadas para resolver esses desafios e, principalmente, o que fazer com as mudanças nos *software*s.

Imagine a complexidade de um *software* utilizado em aparelhos celulares com reconhecimento de voz, por meio de inteligência artificial. Parece simples para o mundo atual, mas no início muitos não viam essa possibilidade, a não ser em filmes de ficção científica. Com a evolução, *software*s de milhões de linhas de instrução apareceram como sistemas operacionais de mercado conhecidos mundialmente. Além do aumento grandioso da complexidade, temos também o fenômeno da urgência em se desenvolver esses *software*s, devido às necessidades de mercado e às mudanças do dia a dia.

Esses fatos fizeram reaparecer o conceito de **crise do *software***, o que pode ser verificado nas mais diversas manifestações, tais como:

- *Software* de baixa qualidade;
- Projetos com prazos e custos maiores que os planejados;
- *Software* não atendendo aos requisitos dos *stakeholders*;
- Custos e dificuldades no processo de manutenção.

02

A crise iniciada nos anos 70 se referia aos *softwares* desenvolvidos na época, que apresentavam em sua maioria uma qualidade inferior e custos muito superiores ao previsto. Para se ter uma ideia, entre 50% a 80% dos *softwares* desenvolvidos não apresentavam as configurações desejadas e cerca de 90% dos *softwares* tinham seu custo final entre 150% a 400% maior que o previsto.

Analisando os sintomas expostos e o que vemos hoje em dia, podemos verificar que **a crise ainda está presente**, mesmo que em menor escala, porém atualmente há inúmeros mecanismos, metodologias e ferramentas para evitar prejuízos mais sérios. E não se engane: mesmo dispondo de técnicas apropriadas na atualidade, ainda presenciaremos esses problemas em um futuro próximo.

Mesmo empresas que têm conhecimento de avançadas técnicas de controle de qualidade às vezes não conseguem coloca-las em prática, sejam por pressão do próprio cliente, seja devido aos prazos ou à necessidade de fazer dinheiro rápido.

Entraremos no mundo da **manutenção**, que corresponde hoje à maioria das demandas na área da tecnologia. Imagine todos os sistemas que foram criados nesses 20, 30 anos decorridos, como eles estão hoje? A resposta é: **estão em constante manutenção**.

03

## 2 - Manutenção de software

Manutenção de *software* é definida como o processo de modificação de um produto de *software*, componente ou sistema após a sua instalação, de forma a corrigi-lo, melhorá-lo ou adaptá-lo para uma mudança no ambiente operacional.

A manutenção de *Software* é, certamente, bem mais do que "consertar erros". A manutenção envolve também a evolução, adaptação e aperfeiçoamento do sistema pronto.

Para a evolução dos sistemas é importante haver **manutenibilidade de *software***, que é o atributo que caracteriza a facilidade de modificação ou adaptação de um *software*.

Em alguns sistemas, a manutenibilidade é quantificada em termos de tempo médio requerido para efetivar a revisão do *software* para eliminar um erro ou realizar uma evolução. Esse atributo é muito significativo para um *software*, uma vez que a etapa de manutenção pode consumir até 65% do custo total de um produto.

Vimos anteriormente que a primeira vez em que aparecem **falhas de *software*** é durante a fase de teste. Detecção de falhas, depuração, correção e teste de regressão são as atividades executadas durante a fase de **teste e integração**. Nesta fase, muitos dos problemas do *software* vêm à tona e são corrigidos, porém, fatalmente, outros somente aparecerão após a sua implantação, quando em operação pelos usuários.

Portanto, após a implantação do sistema em produção, o ciclo de vida do projeto termina e inicia o **ciclo de vida da manutenção do sistema**. Nesta fase, a correção de um erro ou o melhoramento funcional obriga que a modificação seja analisada, implementada, testada, documentada e integrada. Por este motivo, cada manutenção acaba sendo sempre um processo trabalhoso e delicado, buscando que o novo código não introduza novos erros e que a documentação seja atualizada para refletir as modificações.

04

Alguns **problemas que podem dificultar a atividade de manutenção**, principalmente para aqueles *softwares* que atravessam muitos anos em atividade. São eles:

- A documentação do *software* se deteriorou através dos anos, e já não reflete a funcionalidade atual;
- Nenhum dos desenvolvedores do *software* está disponível para esclarecer as dúvidas;
- Forte integração do módulo que sofrerá manutenção com outros módulos;
- Alta complexidade algorítmica do módulo a ser modificado;
- Inexistência de controle de configuração do *software*;
- Inexistência de um ambiente para teste da manutenção;
- Alterações frequentes no *software* motivadas, por exemplo, por necessidades legais.

Não é comum para a indústria de *software* desativar sistemas pelo fato de apresentarem algumas destas características e, por este motivo, devemos procurar alcançar as seguintes **metas durante o desenvolvimento**:

- Usuários e desenvolvedores devem estar bem informados sobre a importância da manutenção, e requisitos de manutenibilidade devem ser detalhados e documentados, principalmente quando o *software* tem perspectiva de modificações constantes, como é o caso de sistemas que se baseiam em legislação que muda com frequência;
- Estabelecimento de um conteúdo mínimo de documentação requerida que auxilie efetivamente o processo de manutenção;
- Determinar os atributos de design necessários para a manutenibilidade requerida e inclusão destes no plano de garantia de qualidade de *software*;
- Estabelecer um processo de manutenção efetivo, se possível, suportado por ferramenta automatizada.

05

Vale ressaltar que a fase do ciclo de vida do projeto é onde geralmente se consome a maior parte dos recursos financeiros e humanos em nível mundial. E a manutenibilidade é a que menos atenção recebe quando estamos planejando o desenvolvimento de um *software*.

A palavra manutenibilidade é relativamente nova. Esse termo, considerado “técnico”, sequer consta na maioria dos dicionários, mas vem ganhando destaque no meio da tecnologia da informação, justamente pelo motivo que move as organizações: custo da manutenção.

Pensar em manutenibilidade durante o desenvolvimento de um *software*, é pensar em atributos que irão facilitar a implementação de modificações e, provavelmente, permitirão minimizar um problema constante da área de Tecnologia da Informação: pessoas que desenvolvem um único projeto e ficam permanentemente na fase de manutenção deste produto. Elas ficam impossibilitadas de atuar em novos projetos, porque não conseguem produzir documentação que facilite a inserção de outros técnicos para ajudar neste trabalho.


06

### 2.1 - O Processo de Manutenção

A meta primordial deste processo é facilitar a acomodação de mudanças, que são inevitáveis em produtos de *software*, reduzindo a quantidade de esforço. O processo de manutenção deve ser amplo o bastante para suportar manutenções de vários tipos: para a correção de um erro (corretiva), para a modificação de uma funcionalidade existente (adaptativa) ou para inclusão de funcionalidade (perfectiva). Veremos cada um desses tipos mais adiante neste módulo.

Pedidos de mudança podem originar, também, por necessidade do usuário ou por diversos outros motivos, **alterações legais**. É importante que haja um processo formal e documentado para que a mudança seja efetivada ou rejeitada, registrando a informação, independente do veredicto da mudança. Muitas vezes, diversas ações de mudança são empacotadas em um único bloco, para facilitar a implementação e o controle.

O procedimento para conduzir a ação da manutenção deve incluir os seguintes **passos**:


**rejeitada**
Mesmo a informação de rejeição deve ser registrada pelo fato de ter sido solicitada novamente por outra fonte.

07

## 3 - Tipos de manutenção de software

Podemos dividir a manutenção de *Software* em três tipos, depois que um programa é liberado para uso:

| **Manutenção Corretiva** (Correção de erros e falhas) | **Manutenção Evolutiva** (Evolução/adaptações e novas funcionalidades) | **Manutenção Preventiva** (Identificação de potenciais problemas) |
|---|---|---|
| • Embora muitos programas sejam testados exaustivamente antes de serem lançados, são encontrados erros no decorrer de utilização do *software*. O processo que inclui diagnóstico e correção de um ou mais erros é denominado manutenção corretiva. | • **Adaptativa:** é o tipo de manutenção realizada para adaptar o *software* a novas plataformas, novo hardware, ou elementos periféricos do sistema que são frequentemente atualizados ou modificados.<br>• **Perfectiva:** é o tipo de manutenção que visa melhorar a funcionalidade e o desempenho do *software*. Essa atividade é responsável pela maior parte de todo o esforço despendido em manutenção de *software*. | • É quando o *software* é modificado para melhorar a confiabilidade e a manutenibilidade futura, ou para oferecer uma base melhor para futuras ampliações. Essa atividade é caracterizada pelas técnicas de engenharia reversa e reengenharia |

08

### 3.1 - Dificuldades na manutenção

As dificuldades na manutenção do *software* dependem de vários fatores, elencaremos os três principais:

- **Tamanho do Sistema**

Conforme o tamanho da aplicação, a manutenção pode levar meses, tempo em que poderia ser iniciado outro sistema. O que agrava nesse caso é quando a aplicação interage com outros sistemas, que são os relacionamentos externos da aplicação. Por exemplo, o sistema administrativo da empresa se relaciona com o sistema do RH (Recursos Humanos), conforme a manutenção no sistema administrativo pode afetar o relacionamento com o sistema de RH. Outro caso é se o sistema administrativo necessitar de mais informações das pessoas que trabalham na empresa. Caso essa informação não esteja contida no sistema de RH, muito provavelmente a manutenção ocorrerá nos dois sistemas, no administrativo para receber a informação e no sistema de RH para adaptar a necessidade do sistema administrativo.

- **Idade do Sistema**

Sistemas antigos usavam tecnologias antigas e grande parte não era documentado. Nesses casos temos um grande problema para tratar, o que pode levar meses para entender como o sistema foi construído e como será manutenido. Uma possível solução pode ser desenvolver um novo sistema.

- **Experiência e Conhecimento dos Mantenedores**

O conhecimento do sistema no momento da manutenção diminui o impacto de uma mudança. Uma pessoa experiente e com conhecimento da aplicação não realizará a manutenção com risco de afetar

outras funcionalidades. Sistemas antigos principalmente, geram muitos impactos na sua manutenção quando ela é realizada por pessoas sem conhecimento e experiência na aplicação. Esses impactos podem causar prejuízos para a empresa. Prejuízos não são somente financeiros, podem estar relacionados com a imagem da empresa. Atualmente é muito trabalhosa a manutenção desses sistemas antigos, pois muitos deles não possuem a documentação e os técnicos que desenvolveram já estão, em sua maioria, inativos.

**09**

A maioria dos problemas associados à manutenção de *software* pode remeter-se também a deficiências na maneira segundo a qual o *software* foi planejado e desenvolvido. O que acontece é a seguinte situação: "pague agora ou pague muito mais depois", ou seja, ou o projeto é bem feito, ou você vai gastar muito dinheiro com a manutenção posterior. A **falta de controle e disciplina** nas atividades de desenvolvimento da engenharia de *software* quase sempre se traduz em problemas durante a manutenção do *software*.

Alguns dos muitos **problemas** clássicos nos projetos e manutenções:

- Comumente é difícil ou impossível rastrear a evolução do *software* através de muitas versões ou lançamentos. As mudanças não estão adequadamente documentadas.
- Comumente é difícil ou impossível rastrear o processo através do qual o *software* foi criado.

Se o projeto do sistema tiver seguido um padrão de desenvolvimento, com uma metodologia, é muito provável existir uma configuração de *software* completa. Assim a tarefa de manutenção inicia-se com uma avaliação da documentação de projeto. As características estruturais, de desempenho, e de interface importantes do *software* são determinadas. O impacto das modificações ou correções exigidas é avaliado, e uma abordagem é planejada. O projeto é modificado e revisado. Um novo código-fonte é desenvolvido, testes de regressão são levados a efeito, e o *software* é liberado novamente.

Essa sequência de eventos constitui uma manutenção organizada e com menos risco de insucesso. Mas sem dúvida é muito mais seguro que a manutenção de algo que não se sabe o que é e o que tem dentro do código.

**Regressão**
É a repetição de testes passados para garantir que as modificações não introduziram falhas no *software* anteriormente operacional.

10

### 3.2 - Custo de manutenção

Existe uma série de **motivos que influenciam os custos de manutenção** de um sistema. Entre eles podemos enumerar os seguintes pontos como diretamente relacionados a esses custos:

- Documentação existente relacionada ao sistema e a qualidade da mesma;
- Qualidade do *software*;
- Design do sistema;
- Paradigma de programação utilizado, códigos antigos com linguagens antigas podem ser problemas;
- Tamanho do *software*, relacionado à quantidade de linhas de código;
- Qualidade técnica da equipe responsável pela manutenção;
- Regras de negócio embarcadas no sistema;
- Plataforma de desenvolvimento.

Desses pontos citados acima, o mais preocupante é a **qualidade da equipe técnica** responsável pelo desenvolvimento. Você poderá passar por todos esses pontos sem preocupação, mas se a sua equipe não for competente, seus custos serão enormes, além de comprometer a imagem do seu projeto para o cliente.

O custo de manutenção de *Software* tem aumentado fortemente durante os últimos 20 anos. Acompanhe:

- Década de 70 – Custo de 35% a 40% do orçamento do *Software*.
- Década de 80 – Custo de 60% a 70% do orçamento do *Software*.
- Década de 90 e 2000 – Custo maior que 90% do orçamento do *software*. **Saiba+**
- Depois de 2000 – Custo se mantém elevado, continua sendo maior que 90% do orçamento do software.

**Saiba+**
Para você ter ideia da importância dos custos de manutenção, acompanhe as informações a seguir. Trata-se de exemplos significativos relacionados a custos de manutenção de *software* no mercado de TI norte-americano na década de 90:

- O custo anual relacionado à manutenção de *software* nos estados Unidos está estimado em mais de US$ 70 bilhões (Sutherland, 1995; Edelstein, 1993).
- Nos EUA, o governo federal gastou sozinho cerca de US$ 8,38 bilhões durante o período de cinco anos devido ao bug do milênio.

A Nokia Inc. utilizou cerca de US$ 90 milhões relacionados a correções preventivas do bug do milênio.

11

Alguns **fatores** influenciam significativamente nos custos. Podemos citar alguns:

- Idade e estrutura do *Software* - enquanto o programa envelhece, sua estrutura se degrada tornando mais difícil de compreender e mudar.
- Falta de documentação do *Software*.
- Grau de complexidade do *Software*.

Além disso, temos outro fator preocupante, que são os **custos intangíveis**, os quais não estão previstos e não se consegue medir. Como por exemplo:

- Oportunidade de desenvolvimento que é postergada ou perdida porque os recursos disponíveis estão atendendo as demandas e tarefas de manutenção;
- Insatisfação do cliente quando solicitações de reparo ou modificação não podem ser resolvidas ou desenvolvidas oportunamente no tempo necessário;
- Redução da qualidade global do *software* como resultado de mudanças que introduzem erros no *software* mantido;
- Perda de produtividade, queda na moral dos funcionários ou uma perda de reputação na comunidade.

12

## Resumo

Nos anos 70 tivemos a Crise do *Software*. O que proporcionou a evolução dos *software*s e trouxe complexidade para o mundo atual, e com isso os problemas com manutenções apareceram. Sistemas construídos nesses 20 e 30 anos passados hoje estão em alta com as manutenções.

Manutenção de *software* é definida como o processo de modificação de um produto de *software*.

Uma construção de projeto de forma desorganizada e não planejada traz problemas para se realizar a manutenção. Já o contrário, projeto usando uma metodologia, pode ser mais fácil na fase da manutenção.

A manutenção de *software* é, certamente, bem mais do que "consertar erros". A manutenção envolve também a evolução, adaptação e aperfeiçoamento do sistema pronto.

Existe uma série de motivos que influenciam os custos de manutenção de um sistema. Os custos da manutenção estão se elevando e é importante pensar na manutenibilidade na construção do projeto. Conforme é construído o projeto, pode ter custos menos elevados nas manutenções futuras.

As manutenções são planejadas e divididas em 3 tipos: Evolutivas, que tratam as evoluções, corretivas, que cuidam das correções e preventivas, que é a prevenção de erros futuros.

O custo de manutenção de *Software* tem aumentado fortemente durante os últimos anos. Além disso, temos outro fator preocupante, que são os custos intangíveis, os quais não estão previstos e não se consegue medir.

As dificuldades na manutenção do *software* dependem de vários fatores, tais como: tamanho do sistema, idade do sistema, experiência e conhecimento dos mantenedores.

## UNIDADE 3 – TESTES, MANUTENÇÃO E ENGENHARIA REVERSA

## MÓDULO 4 – ENGENHARIA REVERSA

**01**

### 1 - ENGENHARIA REVERSA NA TI

Vimos anteriormente a importância da manutenção do *software* e agora aprofundaremos mais no assunto focando na engenharia reversa de *software*, que alguns autores entendem como sendo técnicas de manutenção de *software*.

A engenharia reversa é uma atividade que trabalha com um produto pronto tentando entender *como* este produto funciona, *o que* ele faz exatamente e *como* ele se comporta em todas as circunstâncias. No caso da TI, fazemos engenharia reversa quando queremos trocar, modificar um *software* por outro, com as mesmas características ou entender como este funciona, muitas das vezes não temos acesso a sua documentação ou a documentação está incompleta e desatualizada.

Segundo Pressman, a engenharia reversa tem como principio a desmontagem das caixas pretas do *software*, de seus segredos, de trás para frente, ou seja, o processo de recuperação do projeto com especificação e documentação dos procedimentos, arquitetura e dados.

A engenharia reversa também pode ser usada na criação de novos sistemas a partir de sistemas antigos, utilizando as informações recuperadas para alterar ou reconstituir o sistema existente, num esforço para melhorar sua qualidade global, reimplementando a função do sistema, adicionando novas funções ou melhora de desempenho global.

Por exemplo, imagine um sistema, o qual precisa ser modificado devido a regulamentações que evoluíram. O sistema foi feito há 20 anos, em uma linguagem de programação antiga. O analista que o desenvolveu já saiu da empresa há muitos anos, e não se sabe onde ele reside. A empresa pode ser multada e até ser impedida de continuar atendendo o mercado se não realizar essa mudança. Caso a empresa fique impedida de atuar, trará prejuízos e poderá até causar demissões, visto que o sistema representa cerca de 20% de retorno financeiro. A empresa precisa urgentemente realizar a manutenção ou substituir o sistema por outro. Pelas características do problema, muito dificilmente conseguiremos outro técnico com esse conhecimento. A engenharia reversa permite que a documentação seja elaborada e a tecnologia velha seja especificada.

02

Além da situação apresentada no exemplo, **quando utilizar a engenharia reversa?**

As organizações podem lançar mão da engenharia reversa nas seguintes situações:

- O sistema foi iniciado há muitos anos.
- O sistema tem pouca documentação e ela não foi atualizada, o que quer dizer que a documentação descreve um estado anterior do sistema, mas não a configuração atual.
- As pessoas que criaram o sistema deixaram a empresa, ninguém pode explicar muitas decisões que foram tomadas.
- Algumas partes do sistema foram implementadas com métodos desconhecidos ou sem método algum.
- Muitos programadores diferentes implementaram pequenas partes do sistema. Cada um usava um método e um estilo particular de programação.
- O sistema é implementado numa linguagem de programação antiga (Cobol, Fortran, APL etc.) para a qual existem poucas ferramentas.

E **por que utilizar a engenharia reversa?** É simples: porque todo sistema tem que evoluir. Essa evolução é causada por diversos motivos:

- Para ser adaptado a novos computadores (mais barato, mais rápido ou porque ninguém mantém mais os velhos).
- Para ser adaptado a novos softwares (novas bibliotecas, novas linguagem de programação, novas ferramentas).
- Para ser adaptado a novas regras (por exemplo, troca de moeda em todos os países da Europa).
- Para disponibilizar novas funcionalidades que outras empresas usam.

03

A engenharia reversa pode ser aplicada a qualquer tipo de programa. Por exemplo, se quisermos construir um editor de cronograma compatível com o MS-Project, precisamos entender a representação que ele usa para ler os documentos ou poder salvar documentos no formato do Project.


No exemplo citado anteriormente (sistema construído há mais de 20 anos), a sua reconstrução deverá, a princípio, possuir características não muito diferentes do atual, evitando a rejeição do aplicativo novo. O velho programa tem uma interface específica, e costuma ser acionado de maneira bem definida à qual os usuários da empresa já estão familiarizados. O novo programa deve tentar acatar a mesma interface.

É importante observar que, apesar da evolução dos sistemas e informação, sistemas antigos como o que citamos aqui são muito mais frequentes do que se imagina.

04

## 2 - Engenharia reversa fora da TI

O termo engenharia reversa é utilizado há muito tempo, e teve seu auge no período da Segunda Guerra Mundial. Aviões capturados eram desmontados e estudados. Após esses estudos era possível melhorar o projeto, construindo uma arma melhor que combatesse o inimigo. Para isso, usava-se a engenharia reversa.

Atualmente é comum vermos no mercado inúmeros produtos sendo lançados em versões similares: mesma máquina, mesmo sistema operacional e até as mesmas funcionalidades.

05

É importante, contudo, não confundir engenharia reversa com “pirataria” de produtos, ainda que, em alguns momentos, esta utilize a engenharia reversa.

Fique Atento!

A fabricação de telefones celulares é um exemplo. Inúmeros aparelhos, copiados de marcas famosas, são despejados no mercado e vendidos como produtos originais. Aparentemente podem ser muito semelhantes, mas a qualidade técnica do equipamento é muito inferior. Aparelhos *pirateados* geralmente tem a durabilidade reduzida e não possuem alguns atrativos como informado na hora da venda, como por exemplo, a prova de água e queda.


Porém, nem tudo é crime, pois como qualquer tecnologia, a engenharia reversa pode ser usada para fins acadêmicos, no campo da pesquisa, e em qualquer área de conhecimento.

No Brasil, não existe uma lei específica sobre engenharia reversa. Apesar disso, quando ocorre engenharia reversa, costuma-se proceder de duas maneiras: caso a engenharia reversa não tenha como objetivo a pirataria ou infração de algum direito autoral, não é considerada crime; caso contrário, a Lei de *Software* e também de Direitos Autorais protege seus autores.

**Pirataria**
Pirataria, também chamada de pirataria moderna, é a prática de reproduzir, distribuir, ou mesmo vender produtos sem autorização dos proprietários de um produto ou de uma marca. De acordo com a legislação vigente no país, a pirataria é crime, e a pena pode chegar a quatro anos de reclusão e multa.

06

### 2.1 - Diferença entre engenharia reversa e reengenharia

A engenharia reversa consiste em apenas analisar o sistema ou a ferramenta para criar uma representação dela. Já a reengenharia vai além: analisa-se o projeto, cria-se uma representação do mesmo e, através dessa representação, monta-se uma nova estrutura que funcione exatamente como a primeira, mas que não seja meramente uma cópia dela.

A engenharia reversa de *software* consiste em analisar um determinado sistema para criar representações deste em um nível mais alto de abstração.

A reengenharia de *software* também pode ser vista como uma maneira de "Voltar atrás no ciclo de desenvolvimento do *software*".

07

## 3 - Tipos de engenharia reversa

Mostraremos aqui dois tipos de engenharia reversa de *software*: *sem* o código-fonte e *com* o código-fonte.

### 3.1 - Engenharia reversa sem o código-fonte

O código-fonte do *software* não está disponível, e todos os esforços para descobrir uma possível fonte do código para o *software* são considerados como engenharia reversa.

Podemos assim tentar realizar a engenharia reversa de *software* por vários métodos. Dentre os principais métodos, podemos citar:

- **Análise de fluxo de dados**

Analisa através da observação da troca de informações que envolve o sistema. Algumas ferramentas conseguem capturar o tráfego de dados do *software* por onde os dados percorrem. Esse caminho percorrido pelos dados poderá ser implementado no novo projeto, que será desenvolvido, tendo o mesmo comportamento.

- **Desmontador**

Convertendo a linguagem de máquina em linguagem de montagem, conseguimos assim obter a linguagem de máquina diretamente do programa. O código em linguagem de montagem geralmente contém constantes simbólicas, comentários e rótulos de endereçamento, o desmontador consegue reverter apenas parcialmente o processo de montagem, assim o código produzido por um desmontador é mais difícil de entender necessitando de um decompilação.

- **Decompilador**

Neste método utiliza-se um decompilador, um programa que tenta recriar o código-fonte em uma linguagem de alto nível, tendo disponível apenas o código de máquina.

08

### 3.2 - Engenharia reversa com o código-fonte

Nesse caso, o código-fonte já está disponível e provavelmente a documentação é escassa ou totalmente desatualizada.

O primeiro trabalho que se deve fazer: extrair informações, ou seja, coletar dados sobre o sistema a ser estudado. As atividades da engenharia reversa se fazem sobre essas informações extraídas, mais do que sobre o próprio sistema. As informações podem ser extraídas de várias **fontes**:

- o código-fonte,
- o sistema em execução,
- os dados,
- a documentação e
- outras fontes.

A análise do **código-fonte** permite extrair as **informações** mais básicas do sistema, como:

- Quais são os componentes básicos do sistema: arquivos, rotinas, tipos, variáveis, classes etc.;
- Que relações de definição conectam um componente com seu conteúdo (onde ele se encontra);
- Que relações de referência conectam um componente com aqueles que o usam. Exemplo

Para executar quaisquer desses procedimentos é preciso conhecer a sintaxe da linguagem de programação usada. Dependendo das necessidades, podem-se usar programas específicos (*parsers*), que vão buscar só um tipo particular de informação.

Outra fonte de informação é o **banco de dados**, que representa um papel importante na engenharia reversa de um sistema. Contudo, a engenharia reversa de dados é também um trabalho específico que pode ser feito independentemente de qualquer sistema que possa manipular esses dados. Um exemplo seria a mudança do banco de dados.

**Exemplo**
Se uma rotina A chamar a rotina B, A depende de B porque uma modificação na definição de B pode ter consequências sobre a execução de A.

09

A **documentação**, quando atualizada, contempla tudo o que está sendo usado pelo computador para fazer funcionar o sistema. Entretanto, como a documentação é destinada aos humanos, é impossível a

utilização de ferramentas para extrair informações de forma automática. Assim, o aprendizado do sistema teria como base a leitura dessa documentação. A extração via ferramenta só seria possível caso houvesse uma padronização na escrita ou a automatização dos documentos principais. Ferramentas que constroem os documentos do processo durante o desenvolvimento do *software* possibilitam a extração de informações.

Finalmente, também é possível usar **outras fontes de informação**. Por exemplo, para definir subsistemas, seria imperativo localizar quem escreveu cada parte do código. Se duas partes do código tiverem sido desenvolvidas por uma mesma pessoa, é presumível que pertençam ao mesmo subsistema.

Após recolhidas todas as informações, parte-se, então, para o **tratamento dos fatos**, que contempla algumas das principais atividades envolvidas na engenharia reversa. Dentre as informações colhidas, devem-se eliminar incontáveis detalhes irrelevantes, separando o que é de fato importante.

Em o *software* legado, podem existir várias anomalias no código, como por exemplo, partes do programa que nunca serão executadas, os chamados **códigos mortos**, e trechos de código que foram copiados e ligeiramente modificados. Essas anomalias complicam o código, tornando-o mais extenso e dificultando o estudo e entendimento do sistema por parte do programador. Para resolver esse problema, o melhor a fazer é deletar o código morto e no caso dos clones. Pode-se, ainda, alterar o código para suprimir os clones ou comentar o código informando que existem clones.

10

## 4 - Sistemas legados

Um fator importante que pode ajudar muito a engenharia reversa é quando o técnico que desenvolveu o sistema ainda faz parte da empresa. É muito vantajoso quando se tem pessoas que conheçam o sistema, pois esse aspecto encurta consideravelmente o caminho que será percorrido no processo de engenharia reversa.

O que as empresas deveriam fazer com os sistemas legados, que ainda possuem pessoas originárias do desenvolvimento do projeto?

A resposta parece muito simples e fácil de responder: é melhor construir outro sistema enquanto ainda se tem o conhecimento no projeto.

Mas isso não é o que acontece na maioria das vezes. O dia a dia de uma empresa é muito corrido e ocupado por diversas demandas e, quando menos se espera, as pessoas se aposentam ou deixam a empresa. Em geral, as empresas não se interessam em remodelar sistemas antigos que ainda funcionam, comumente optam por iniciar um novo projeto.

Contudo, os custos são altos e para aprovar o orçamento de um projeto cujo sistema está em funcionamento é muito difícil. Somente se dá a devida importância, na maioria das vezes, quando o

sistema apresenta falhas sérias e problemas nas novas manutenções, o que pode ocasionar prejuízos financeiros e comprometimento da imagem da empresa.

O mais adequado seria iniciar a reengenharia no sistema enquanto ainda se possa contar com membros da equipe que o desenvolveu, dado o conhecimento que ainda se tem do sistema, o que pode minimizar eventuais problemas e não deixar que o problema seja resolvido apenas por meio da engenharia reversa.

11

**4.1 - Ciclo de vida do sistema**

Existe uma estimativa de vida para um sistema de informação. Uma característica importante a ser lembrada é que normalmente um sistema de informação tem um ciclo de vida. A vida útil dos sistemas se dá na sua fase de utilização plena ou maturidade e ao longo de sua vida podem ocorrer manutenções, melhorias, implementações e eventuais correções de erros. A "morte" ocorre quando o sistema é substituído ou está em desuso, o que ocorre, muitas vezes, devido à utilização de tecnologia desatualizada.

Os sistemas de informação operacionais geralmente não morrem, mas os gerenciais e estratégicos (executivos) precisam de atualizações frequentes e implementações de novas tecnologias para se manterem funcionais, atendendo às necessidades da empresa.

O **ciclo de vida natural** de um sistema de informação abrange as seguintes fases:


Quando as três primeiras fases não são bem elaboradas, a morte do sistema de informação pode ocorrer antes do esperado.

**Concepção ou iniciação:**
É o nascimento do sistema, também chamado de projeto.

**Construção:**
Execução do sistema.

**Implantação:**
Disponibilização do sistema ao cliente e/ou usuário;

**Implementações:**
Agregação de funções, melhorias e eventuais correções de erros.

**Maturidade:**
Utilização plena do sistema, com satisfação integral do cliente e/ou usuário.

**Declínio:**
Dificuldade de continuidade, impossibilidade de agregação de funções necessárias, insatisfação do cliente e/ou usuário.

**Manutenção:**
Elaboração de manutenções, visando à tentativa de sobrevivência do sistema.

**Morte:**
Descontinuidade do sistema, substituição.

12

## 5 - Aspectos Legais

A engenharia reversa pode também gerar problemas de legalidade. Imagine uma empresa criando uma cópia de um produto que tem boa saída no mercado, porém esse produto pertence a outra empresa. As questões legais variam de país para país. E, mesmo assim, ainda existem países que não possuem leis específicas sobre o assunto.

A "Digital Millenium Copyright Act" dos Estados Unidos é uma lei aprovada em 1998 que, entre várias medidas para proteger direitos autorais na informática, também faz restrições em relação à engenharia reversa. Só é permitida para fins de **analisar compatibilidade com outros *softwares*** e/ou hardware.

Na União Europeia, o "EU Copyright Directive", de 2001, é similar ao "Digital Millenium Copyright Acts", porém não é tão restritiva. Só são feitas restrições caso o objetivo final da engenharia reversa seja a cópia de algum programa ou quebra de patente com objetivo de lucro. Caso seja pra fins acadêmicos ou

de compatibilidade, a princípio não existem restrições. Na Suíça, a lei a respeito do assunto é bastante curiosa e, de certo modo, polemica. A Lei Suíça de Concorrência Desleal de 1986 exige dos competidores a realização de investimentos em engenharia reversa mesmo quando a tecnologia não seja secreta. Os tribunais suíços, porém, têm rejeitado ou limitado severamente a aplicação de tal norma, pela inexistência de prazo e limites.

No Japão, a Lei Japonesa de Concorrência Desleal de 1993 proíbe a imitação servil, mesmo no caso de produtos não patenteados, nem protegidos por direitos autorais. A lei japonesa impõe limites claros à aplicação da norma de apropriação ilícita. Ou seja, a proibição de imitação não impede o progresso técnico, ressalva o domínio das patentes para proteger ideias e conceitos, e o interesse social na padronização e compatibilização industrial.

**Limites**
Os limites impostos pela lei japonesa são:

- o “lead time” vigora apenas por três anos;
- não se protegem as ideias e os conceitos técnicos;
- e ressalva-se o caso de modificações ou aperfeiçoamento técnico efetuado pelo competidor com base no item copiado;
- a necessidade de padronização e compatibilização de produtos e o uso de elementos de caráter estritamente funcional.

13

Os problemas legais relacionados à engenharia reversa são bastante comuns e difíceis de serem resolvidos. Por exemplo:

Segundo o site www.findlaw.com, o caso da Lotus Development Corp. vs. Borland International foi uma disputa judicial entre duas empresas produtoras de *Software*. A Lotus produzia o Lotus 1-2-3, e a Borland, o Quattro Pro. A Borland produziu seu programa de computador com a interface idêntica ao da Lotus, de maneira que os usuários da Lotus 1-2-3 pudessem usar o Quattro Pro sem dificuldades. A Lotus entrou com ação em face da Borland por infração de Copyright. Grande parte da controvérsia foi a respeito da possibilidade de se proteger pelo direto do autor a mera interface do programa. Por fim, a decisão final achou "absurdo" sugerir que "se alguém faz uso de vários programas diferentes, seja forçado a aprender como efetuar cada a mesma operação de maneiras diferentes em cada programa utilizado". A corte decidiu que, se uma empresa atinge um monopólio, por consequência a maioria do mercado fica bem adaptada à interface de seu programa. Desse modo, é justo que um competidor utilize a mesma interface como meio de concorrência.

No Brasil, segundo o Advogado José O. A. Motta Júnior, ao contrário dos EUA, existe uma série de previsões legais para situações não atingidas pelo Direito do Autor. Estão previstas pelo art. 46 da Lei 9.610/98 (Lei dos Direitos Autorais) e pelo art. 6º da Lei do *Software*. Anteriormente, eram previstos pelo art. 49 da antiga Lei 5.988/73.


Na verdade, sob um ponto de vista científico, esses artigos confundem os limites ao direito do autor propriamente dito com certas atividades que deveriam ficar à margem da proteção ao direito do autor. O direito a uma cópia privada sem o intuito de lucro, na verdade, é uma expressão do princípio da liberação para uso privado.

14

## Resumo

A Engenharia Reversa surgiu da necessidade de reconstruir sistemas antigos, que são chamados de legados. Engenharia reversa é uma atividade que trabalha com um produto pronto tentando entender como este produto funciona. A Engenharia reversa pode acontecer na TI ou fora dela também, não é uma restrição. Em alguns casos pode ser confundida com plágio, mas não é. Temos também outro conceito importante, a reengenharia, que é o desenvolvimento de algo existente, só que novo. Acontece também com sistemas legados que precisam ser reconstruídos.

Todo sistema tem o seu tempo de vida, desde a criação do projeto, as manutenções e sua morte. O ciclo de vida de um *software* possui as seguintes fases: Concepção, Construção, Implantação, Implementação, Maturidade, Declínio, Manutenção e Morte.

A TI evolui muito rápido e os sistemas acabam ficando defasados necessitando, em alguns, de serem substituídos. Outro ponto a observar são os aspectos legais, há muitos processos de pirataria e clonagem no mundo. As leis que regulamentam não evoluem na mesma velocidade da tecnologia, o que causa processos intermináveis no mundo a fora.

Outro ponto a observar é que a engenharia reversa pode também gerar problemas de legalidade, as cópias podem ser entendidas como quebra dos direitos autorais. Em países como o Japão a Lei Japonesa de Concorrência Desleal de 1993 proíbe a imitação servil, mesmo no caso de produtos não patenteados, nem protegidos por direitos autorais.