Faculdade de Tecnologia De Jundiaí (FATEC-Jundiaí)  
Curso de Análise e Desenvolvimento de Sistemas

André Batista  
Elton Ramos  
Fábio Massaretto  
Rodrigo Orlando

Teste de Software  
Engenharia de Software II

Jundiaí  
2015

André Batista  
Elton Ramos  
Fábio Massaretto  
Rodrigo Orlando

Teste de Software

Trabalho de Engenharia de Software II sobre teste de software e suas aplicações.

Jundiaí  
2015

Dedico esse trabalho aos meus colegas da turma de análise e desenvolvimento de sistemas.

Nesse trabalho agradeço a Faculdade de Tecnologia de Jundiaí por nos permitir o acesso aos computadores assim podendo realiza-lo. À professora Adani C. Sacilotti pela orientação e dicas nesse nosso trabalho sobre teste de software.

Esse trabalho tem como objetivo dar uma pequena introdução ao conceito de informática, e de forma simples mostrando os diversos recursos e dispositivos para quem está começando ou querendo aprender uma nova informação. Discutiremos desde os assuntos relacionados ao hardware quanto ao software.

This work has as objective give us a short introduction to the concept of Informatics and in a simple way show off the variety of resources and devices for whom is starting out or wanting to learn new information. We will discuss since subjects related about hardware as to software.

[Figure 1: Confiabilidade do Software 14](file:///C:\Users\Fábio\Documents\GitHub\SoftwareTesting\Teste%20de%20Software.docx#_Toc430444508)

[Figure 2: Noticia de falha de segurança da Symantec 16](file:///C:\Users\Fábio\Documents\GitHub\SoftwareTesting\Teste%20de%20Software.docx#_Toc430444509)

[Figure 3: Fases do Desenvolvimento de Software 17](file:///C:\Users\Fábio\Documents\GitHub\SoftwareTesting\Teste%20de%20Software.docx#_Toc430444510)

[Figure 4: Qualidade do Software. 18](file:///C:\Users\Fábio\Documents\GitHub\SoftwareTesting\Teste%20de%20Software.docx#_Toc430444511)

[Figure 5: Requisitos e ciclos 20](file:///C:\Users\Fábio\Documents\GitHub\SoftwareTesting\Teste%20de%20Software.docx#_Toc430444512)

[Figure 6: Ambientes de teste de Software 31](#_Toc430444513)

[Figure 7: Ambiente de maquinas virtuais 35](#_Toc430444514)

[Figure 8: Impacto vs. Probabilidade 50](#_Toc430444515)

[Figure 9: Matriz de priorização de riscos 50](#_Toc430444516)

[Figure 10: Ciclo básico de vida de um projeto. 52](#_Toc430444517)

[Figure 11: Plano de teste 56](#_Toc430444518)

[Figure 12: Documentação da Elaboração dos Testes 57](#_Toc430444519)

[Figure 13: Casos de teste 58](#_Toc430444520)

[Figure 14: Exemplo de caso de teste 59](#_Toc430444521)

[Figure 15: Taxonomia 69](#_Toc430444522)

[Table 1: Organização testes atributos de acordo com os tipos de testes 19](#_Toc429924769)

Contents

[O que é testar? 13](#_Toc430443886)

[Qual o objetivo dos testes? 13](#_Toc430443887)

[Confiabilidade do Software x Defeitos 13](#_Toc430443888)

[Confiabilidade do Software 14](#_Toc430443889)

[Porque defeitos ocorrem no software? 14](#_Toc430443890)

[Vale a pena investir em testes? 14](#_Toc430443891)

[Realidade atual do mercado de software 15](#_Toc430443892)

[O custo da não-qualidade 15](#_Toc430443893)

[Exemplo 1 16](#_Toc430443894)

[Exemplo 2 17](#_Toc430443895)

[O custo da correção dos defeitos 17](#_Toc430443896)

[Porque testar é necessário? 18](#_Toc430443897)

[Qualidade do Software 18](#_Toc430443898)

[A abordagem tradicional dos testes 18](#_Toc430443899)

[A melhor abordagem para os testes 19](#_Toc430443900)

[Por que as empresas não testam? 19](#_Toc430443901)

[Processo de teste 19](#_Toc430443902)

[Cenário atual: 19](#_Toc430443903)

[Cenário desejável: 20](#_Toc430443904)

[Objetivo do processo de teste 20](#_Toc430443905)

[O que é estratégia de teste? 21](#_Toc430443906)

[O que deve conter uma estratégia de teste? 21](#_Toc430443907)

[Desenvolvimento da estratégia de teste 21](#_Toc430443908)

[Testes Unitários 21](#_Toc430443909)

[Testes de Sistema 22](#_Toc430443910)

[Tipos de Testes de Sistema 23](#_Toc430443911)

[Teste de Aceitação 23](#_Toc430443912)

[Características da qualidade 23](#_Toc430443913)

[Técnicas de teste 24](#_Toc430443914)

[Técnica de teste estrutural 24](#_Toc430443915)

[Teste de estresse 24](#_Toc430443916)

[Teste de contingência 25](#_Toc430443917)

[Teste de segurança 25](#_Toc430443918)

[Teste de performance 26](#_Toc430443919)

[Teste de conformidade 27](#_Toc430443920)

[Teste de funcionalidade 27](#_Toc430443921)

[Teste de regressão 28](#_Toc430443922)

[Teste de interconexão 28](#_Toc430443923)

[Teste de usabilidade 28](#_Toc430443924)

[Baseando-se nos riscos 29](#_Toc430443925)

[Ambiente de teste 30](#_Toc430443926)

[O que é ambiente de teste? 30](#_Toc430443927)

[Elementos do ambiente de testes 30](#_Toc430443928)

[Maneiras de testar o software 31](#_Toc430443929)

[Preparação do ambiente 32](#_Toc430443930)

[Escopo 32](#_Toc430443931)

[Equipe 33](#_Toc430443932)

[Volume dos dados 33](#_Toc430443933)

[Origem dos dados 33](#_Toc430443934)

[Interfaces 34](#_Toc430443935)

[Ambientes virtuais 35](#_Toc430443936)

[Preparação do ambiente de teste 35](#_Toc430443937)

[Automação de testes 39](#_Toc430443938)

[Técnica x Ferramenta 39](#_Toc430443939)

[Mas, afinal, o que é automação de testes? 39](#_Toc430443940)

[Quando utilizar ferramentas de automação de testes? 39](#_Toc430443941)

[Ferramentas de teste 40](#_Toc430443942)

[Ferramentas de gerenciamento 40](#_Toc430443943)

[Ferramentas de verificação de código 40](#_Toc430443944)

[Ferramentas de execução de testes 41](#_Toc430443945)

[Análise de riscos 42](#_Toc430443946)

[O que é risco? 42](#_Toc430443947)

[Conceituando riscos - Exemplos 43](#_Toc430443948)

[Categoria de riscos 43](#_Toc430443949)

[Gerenciamento de riscos 43](#_Toc430443950)

[Identificação dos riscos: 44](#_Toc430443951)

[Analisar os riscos 49](#_Toc430443952)

[Priorizar os riscos 50](#_Toc430443953)

[Planejar resolução dos riscos 51](#_Toc430443954)

[Plano de mitigação 51](#_Toc430443955)

[Plano de contingência 51](#_Toc430443956)

[Planejamento de testes 52](#_Toc430443957)

[O que é um projeto? 52](#_Toc430443958)

[Porque investir em planejamento? 52](#_Toc430443959)

[Estratégia de Testes – Método Rex Black 53](#_Toc430443960)

[Plano de testes 53](#_Toc430443961)

[Padrões e planos de testes 53](#_Toc430443962)

[Plano de Testes – Padrão PMI 53](#_Toc430443963)

[Plano de Testes – Padrão QAI 54](#_Toc430443964)

[Plano de Testes – Padrão IEEE 829 55](#_Toc430443965)

[Construindo um Plano de Testes 56](#_Toc430443966)

[Documentação da Elaboração dos Testes 57](#_Toc430443967)

[Estrutura básica de um Caso de Teste 57](#_Toc430443968)

[Composição de um Caso de Teste 58](#_Toc430443969)

[Exemplo de Caso de Teste 59](#_Toc430443970)

[Papéis e Responsabilidades 59](#_Toc430443971)

[Testes exaustivos 60](#_Toc430443972)

[Métodos para refinamento de Casos de Teste 61](#_Toc430443973)

[Classe de Equivalência 61](#_Toc430443974)

[Análise de Valor-Limite 62](#_Toc430443975)

[Refinamento por Probabilidade de Erro 63](#_Toc430443976)

[Desafios 64](#_Toc430443977)

[Execução dos testes 65](#_Toc430443978)

[Forma estática: 65](#_Toc430443979)

[Forma dinâmica: 65](#_Toc430443980)

[Como os tipos de testes devem ser feitos? 66](#_Toc430443981)

[E o que acontece após a execução dos testes? 66](#_Toc430443982)

[Quando os testes terminam? 67](#_Toc430443983)

[O que acontece se o software for entregado muito cedo? 67](#_Toc430443984)

[O que acontece se o software for entregado muito tarde? 67](#_Toc430443985)

[Gestão de defeitos 68](#_Toc430443986)

[Aonde as falhas podem ser identificadas? 68](#_Toc430443987)

[O que acontece depois que o defeito é reconhecido? 68](#_Toc430443988)

[Como se soluciona os defeitos? 68](#_Toc430443989)

[Relatórios de teste 70](#_Toc430443990)

[Gerência de comunicação 70](#_Toc430443991)

[Bibliografias 73](#_Toc430443992)

Com a crescente demanda por softwares para automatização de atividades humanas, cada vez mais é necessário a produção de softwares com qualidade e menor tempo para a produzi-lo.

Com o passar dos anos a engenharia de software evoluiu significativamente e com isso surgiram novos métodos, técnicas e ferramentas para a produção de softwares com qualidade. O teste de software visa encontrar erro que podem se tornar defeitos no software e ocasionar a falha no sistema ou seja o teste de software tem como objetivo mostrar que o software faz o que não deveria fazer, ocasionando a falha. Quanto menor a quantidade de falhas maior será a qualidade do software.

O teste de produto de software envolve basicamente quatro etapas: Planejamento de testes, projeto de casos de testes, execução e avaliação dos resultados do teste.

# O que é testar?

Encontramos na literatura algumas definições sobre que é a atividade de testar:

* “Testar é analisar um programa com a intenção de descobrir erros e defeitos.” ( Myers)
* Testar é exercitar ou simular a operação de um programa ou sistema.
* Testar é confiar que um sistema faz o que se espera que ele faça e não faz o que se espera que não faça.
* Testar é medir a qualidade e funcionalidade de um sistema.
* “O teste de programas pode ser usado para mostrar a presença de defeitos, mas nunca para mostrar a sua ausência.” (Dijkstra)

# Qual o objetivo dos testes?

O objetivo principal dos testes é reduzir a probabilidade da ocorrência de um defeito quando o software estiver em produção, minimizando os riscos para o negócio e garantindo que as necessidades do cliente estão sendo atendidas

“Uma pessoa inteligente resolve um problema. Uma pessoa sábia evitá-o”. [Einsten]

Erro, Defeito e Falha

Erro (error): é uma ação humana que produz um resultado incorreto. Defeito (fault): A manifestação de um erro no software também conhecido como Bug se executado, o defeito pode causar uma falha. Falha (Failure): diferença indesejável entre o observado e o esperado. (Defeito encontrado)

Falha é um evento; defeito é um estado do software, causado por um erro.

# Confiabilidade do Software x Defeitos

Confiabilidade do Software é a probabilidade que o software não causará uma falha no sistema por um tempo especificado, sob condições determinadas.

* Pode existir um software livre de defeitos?
* Por existir softwares confiáveis mas que possuem defeitos?

## Confiabilidade do Software

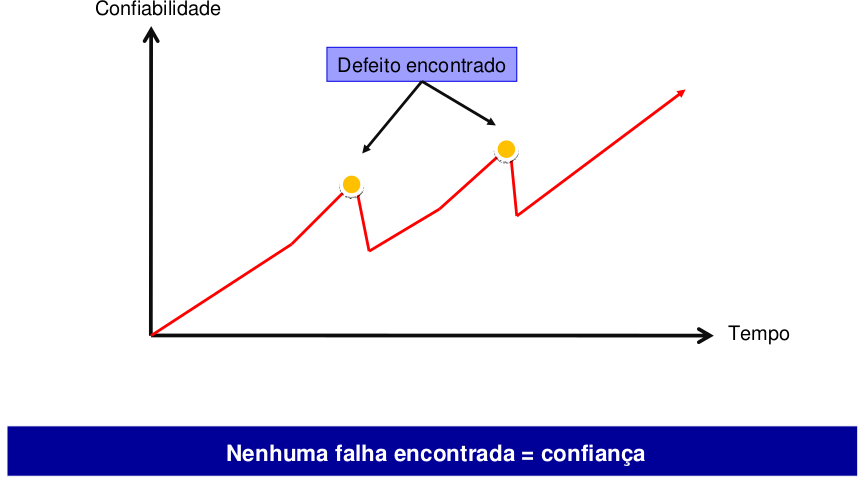


Figure 1: Confiabilidade do Software

# Porque defeitos ocorrem no software?

1. Softwares são escritos por humanos
2. As pessoas não conhecem e não dominam tudo;
3. As pessoas tem habilidades, mas não são perfeitas;
4. As pessoas cometem erros;
5. Softwares são desenvolvidos sob crescente pressão para entregá-los em prazos rigorosos
6. Sem tempo para checar as atividades realizadas;
7. Sistemas podem estar incompletos.

# Vale a pena investir em testes?

Investir em teste reduz em 70% o índice de re-trabalho de correção de falhas em produção, reduz em 50% do tempo de homologação de uma nova versão, aumenta em 90% o índice de falhas detectadas antes da produção, diminui em 95% os erros em produção aumenta a abrangência dos testes.

# Realidade atual do mercado de software

Hoje nos deparamos com a complexidade dos softwares, complexidade das equipes, cronograma apertado para Desenvolvimento, Mercado/Cliente/Usuários mais exigentes, menos tolerância a falhas, menos tolerância aos atrasos das entregas precisamos construir softwares MELHORES e mais BARATOS, de forma mais RÁPIDA

# O custo da não-qualidade

Estatísticas de mercado apontam que para cada R$ 1,00 investido no desenvolvimento e manutenção de software, entre R$ 2,00 e R$ 3,00 são gastos com re-trabalho.

* Não se sabe o percentual de re-trabalho e quanto isso custa
* O software ou parte dele tem de ser refeito
* Não se sabe, através de métricas claras e precisas, quais são as principais áreas de produção que geram re-trabalho
* A empresa não toma iniciativas para corrigir os problemas nas áreas de produção ineficientes

Existem alguns tipos de sistemas onde:

* As falhas causam prejuízos diretos
* As falhas causam a perda de confiança do público (usuários)

## Exemplo 1

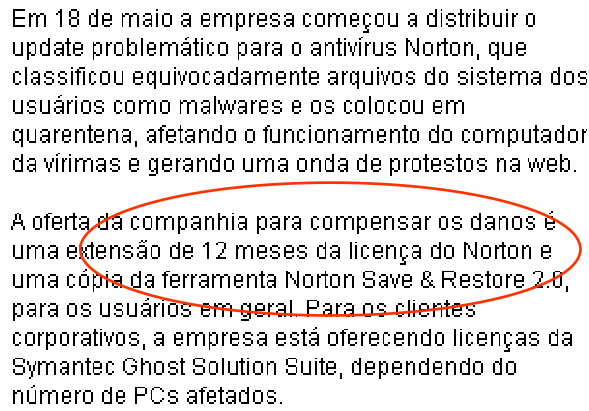
******

Figure 2: Noticia de falha de segurança da Symantec

## Exemplo 2

O custo das falhas AirBus 320

* Desastre
  + US Vicennes derrubou um Airbus 320 em 1988
* Motivo
  + Falha no software de reconhecimento, confundindo o avião com um F-14
* Prejuízo
  + 290 mortes

# O custo da correção dos defeitos

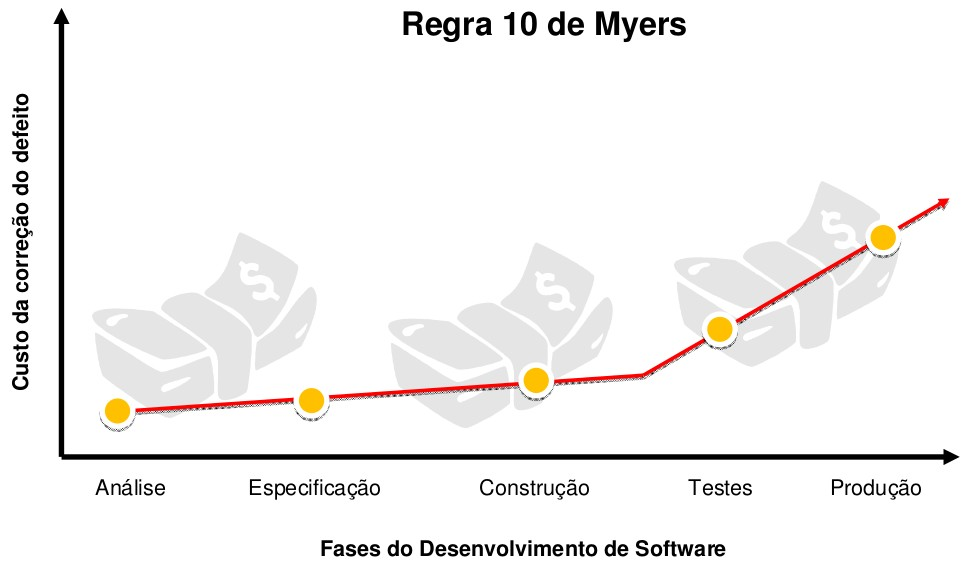
A regra 10 de Myers indica que o custo da correção de um defeito tende a ser cada vez maior quanto mais tarde ele for descoberto. Defeitos encontrados nas fases iniciais da etapa de desenvolvimento do software são mais baratos de serem corrigidos do que aqueles encontrados na produção.

Figure 3: Fases do Desenvolvimento de Software

# Porque testar é necessário?

* porque é provável que o software possua defeitos;
* para descobrir a confiabilidade do software;
* porque falhas podem custar muito caro;
* demonstrar as conformidades com os requisitos;

Para assegurar que as necessidades dos usuários estejam sendo atendidas;

* Para reduzir custos;
* Para avaliar a qualidade do software;

Não devemos testar:

* Para provar que o software não tem defeitos;
* Apenas porque os testes estão incluídos no plano de projeto;

# Qualidade do Software

Figure 4: Qualidade do Software.

# A abordagem tradicional dos testes

Mostrar que o sistema:

1. Faz o que deve fazer
2. Não faz o que não deve fazer

META: Mostrar que o sistema funciona

CRITÉRIO DE SUCESSO: Sistema em funcionamento correto

RESULTADO: Sistema com defeitos

# A melhor abordagem para os testes

Mostrar que o sistema:

1. Faz o que não deve fazer
2. Não faz o que deve fazer

META: encontrar defeitos

CRITÉRIO DE SUCESSO: Sistema com falhas

RESULTADO: Sistema com menos defeitos

# Por que as empresas não testam?

Teste não é uma atividade trivial porque:

Muitas vez o software é complexo, software é algo intangível, o software é altamente modificável, teste lida com pessoas,muitas vezes há desconhecimento sobre a relação custo x benefício, falta de profissionais especializados, dificuldade em implantar um processo de teste, existe o desconhecimento das técnicas adequadas de teste, há desconhecimento de como planejar as atividades, e muitas vezes só se preocupam com testes no final do projeto.

# Processo de teste

## Cenário atual:

1. Testes são realizados como uma etapa do processo de desenvolvimento
2. Pressões sobre prazos fazem com que o esforço de testes seja reduzido ou até mesmo eliminado
3. O objetivo é assegurar que as especificações foram construídas
4. Testes são realizados pelos desenvolvedores ou analistas de sistema
5. Testes são realizados ao final do desenvolvimento
6. Não há garantias de que o software foi testado adequadamente

## Cenário desejável:

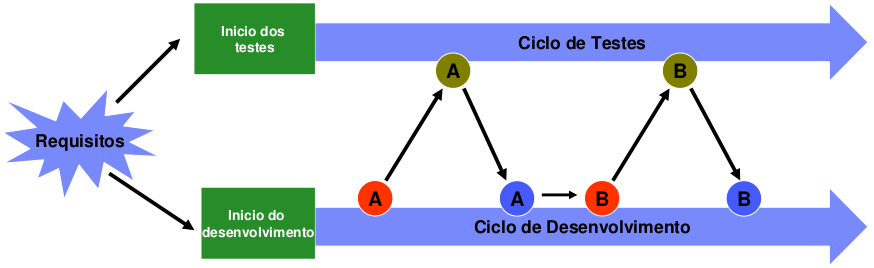
* O teste é tratado como um processo independente, porém altamente integrado ao processo de desenvolvimento
* Os testes são iniciados paralelamente ao processo de desenvolvimento
* Testes realizados por equipes altamente qualificadas e especializadas

Figure 5: Requisitos e ciclos

# Objetivo do processo de teste

Permitir o teste de softwares usando pessoal técnico qualificado, ferramentas adequadas e base em metodologia aderente ao processo de desenvolvimento da organização

Alguns princípios foram sugeridos ao longo dos últimos 40 anos:

Princípio 1: Teste demonstra a presença de defeitos

Princípio 2: Teste exaustivo é impossível

Princípio 3: Teste antecipado (iniciar o quanto antes)

Princípio 4: Agrupamento de defeitos (20/80)

Princípio 5: Paradoxo do Pesticida

Princípio 6: Teste depende do contexto

Princípio 7: A ilusão da ausência de erros (deve atender as necessidades dos usuários)

# O que é estratégia de teste?

Uma estratégia de testes de software descreve a abordagem geral e os objetivos das atividades de teste. Ela deve contemplar os níveis ou fases de teste, os tipos de testes a serem realizados e as técnicas para sua execução. A estratégia de testes de softwares também deve descrever com clareza os critérios para a conclusão dos testes e os critérios de sucesso a serem usados.

# O que deve conter uma estratégia de teste?

A estratégia de teste deve responder as perguntas que seguem:

1. Quando vamos testar?
2. O que vamos testar?
3. Como vamos testar?

# Desenvolvimento da estratégia de teste

Ao elaborar uma estratégia de teste eficiente e eficaz, devemos ter em mente a execução de alguns passos:

1. Avaliar os requisitos do sistema
2. Identificar e analisar os riscos para o negócio
3. Analisar as possibilidades de testes para minimizar os riscos
4. Estabelecer os tipos e técnicas de teste a serem realizadas
5. Definir prioridades para os testes
6. Estabelecer quais produtos serão inspecionados ou testados

# Testes Unitários

O primeiro nível de teste é o Teste Unitário, que envolve assegurar que cada funcionalidade especificada no desenho do componente tenha sido implementada corretamente neste componente.

Características deste nível:

* Testes realizados em uma unidade independente do produto
* Estágio mais baixo da escala de teste
* Aplicado nos menores componentes de código criados, visando garantir que estes atendem as especificações funcionais e de arquitetura
* Geralmente realizado pelo desenvolvedor que construiu a unidade
* Utiliza técnicas de Caixa BrancaTestes de Iteração ou Integração Como os componentes são construídos e testados separadamente, ao serem acoplados deve-se verificar se eles interagem corretamente.

Neste nível os testes não são focados em “o quê” os componentes fazem, mas se eles se comunicam conforme especificado no desenho do sistema.

Características deste nível:

* Acontece após os Testes Unitários
* Realizados dentro de um ambiente controlado
* Geralmente realizado pelo analista de sistema para um módulo ou conjunto de programas
* Normalmente são seguidos dois métodos de interação:
* Abordagem Top-Down
* Abordagem Bottom-Up

# Testes de Sistema

Os testes de sistema têm foco no funcionamento do sistema como um todo, para validar a exatidão e a perfeição na execução das funções requeridas.

Características deste nível:

* Acontece após todos os testes de integração
* Realizado dentro de um ambiente controlado
* Realizado pela equipe de testes
* Envolve testes especializados para validar todos os requisitos funcionais e não-funcionais:
* Desempenho
* Volume
* Documentação
* Robustez

# Tipos de Testes de Sistema

Os testes de sistema têm foco no funcionamento do sistema como um todo, para validar a exatidão e a perfeição na execução das funções requeridas.

Entre outros, são realizados os seguintes testes:

* Teste de Funcionalidade
* Teste de Segurança
* Teste de Estresse
* Teste de Desempenho

# Teste de Aceitação

Último nível de testes antes da implantação do software em ambiente de produção. Seu objetivo é verificar se o sistema tem condições de ser implantado em produção, com base nos critérios de aceitação.

Algumas características:

* É de responsabilidade do Cliente
* Tem a finalidade de validar se o software está apto a entrar em produção e executar as funções ou tarefas a que se propôs
* Geralmente realizado em ambiente de homologação do Cliente
* Testes de aceitação são realizados pelo usuário final para capacitá-lo e para validar todos os requisitos

# Características da qualidade

A Norma ISSO 9126-1 define seis características de qualidade que o software deve atender, a saber:

* Funcionalidade: verifica a capacidade do sistema em prover funcionalidades definidas que atendam as necessidades do usuário, quando usado sob determinadas condições preestabelecidas.
* Confiabilidade: o produto de software é capaz de manter seu nível de desempenho, ao longo do tempo, nas condições estabelecidas de utilização.
* Usabilidade: capacidade do software em ser entendido, aprendido e utilizado sob condições estabelecidas de utilização.
* Eficiência: os recursos e os tempos envolvidos são compatíveis com o nível de desempenho requerido pelo software.
* Manutenibilidade: refere-se ao esforço necessário para a realização de alterações específicas no produto de software.
* Portabilidade: facilidade de o software poder ser transferido de um ambiente para outro.

# Técnicas de teste

## Técnica de teste estrutural

Utilizada quando o objetivo é garantir que o produto desenvolvido está estruturado e funciona corretamente. Pretende determinar se a tecnologia utilizada foi apropriada e se os componentes montados funcionam como uma unidade coesa.

Técnica de teste funcional

Utilizada quando o objetivo é verificar se os requisitos do sistema e as especificações foram atendidas. Envolve a criação de cenários de testes para uso na avaliação das uncionalidades da aplicação, validando se o que foi especificado foi implementado corretamente.

O teste funcional não está preocupado com o processo ocorre em si, e sim com os resultados do processo.

## Teste de estresse

A principal meta do teste de estresse é entender o comportamento do sistema durante condições-limite de execução ou fora da tolerância esperada. Tipicamente envolve a execução do sistema com baixos recursos de hardware e software, ou a concorrência por estes recursos.

Objetivos:

* Determinar a que condições-limite de recursos o software é capaz de ser executado
* Verificar se o sistema é capaz de garantir tempos adequados de resposta sendo executado em condições-limite
* Verificar se há restrições quanto ao ambiente em que o software vai operar
* Determinar que volumes de transação, normais e acima dos normais, podem ser processados num período de tempo esperado
* Quando usar:
* Quando não se sabe quais as condições mínimas de recursos para operacionalização do sistema, sem que haja perdas significativas.

## Teste de contingência

A meta principal do teste de contingência é verificar se, após uma falha, a sua recuperação é adequadamente executada, garantindo a continuidade dos serviços.

Objetivos:

* Manter o backup dos dados
* Armazenar o backup em local seguro
* Documentar os procedimentos de recuperação
* Deixar claro as responsabilidades das pessoas em caso de um desastre
* Quando usar:
* Sempre que a continuidade dos serviços for essencial para o negócio
* Perdas devem ser estimadas (quanto vou perder por hora?)
* A quantidade de perda potencial estipula a quantidade de esforço que irá ser empregada

## Teste de segurança

A principal meta do teste de segurança é garantir que os dados ou funções de um sistema possam ser acessados apenas por atores autorizados a acessá-las. Todas as formas de ataque de acesso indevido devem ser simuladas.

Objetivos:

* Garantir que os dados do sistema estão protegidos adequadamente
* Assegurar que todos os riscos que envolvem os acessos indevidos foram identificados
* Analisar as ameaças e vulnerabilidades, tanto físicas como lógicas
* Assegurar que os mecanismos de controle contra acessos indevidos foram corretamente implementados
* Quando usar:
* Testes básicos de segurança devem ser aplicados a todos os softwares
* Quando a informação que circula através do software for de importância fundamental para a organização

## Teste de performance

O teste de performance, ou de desempenho como também é conhecido, mede e avalia o tempo de resposta, o número de transações e outros requisitos sensíveis ao tempo de resposta do sistema.

Objetivos:

* Determinar o tempo de resposta das transações
* Verificar se os critérios de desempenho estabelecidos estão sendo atendidos
* Identificar pontos de gargalo no sistema
* Verificar o nível de utilização do hardware e software

Quando usar:

* Quando se deseja avaliar o tempo de resposta de uma funcionalidade do sistema ou de todo o sistema
* Devem ser realizados quando ainda há tempo hábil de serem realizadas correções

## Teste de conformidade

Verificar se a aplicação foi desenvolvida seguindo os padrões, procedimentos e guias da área de processos.

As metodologias são usadas para aumentar a probabilidade de sucesso do projeto, e portando devem ser testadas.

Objetivos:

* Verificar se as metodologias de sistema estão sendo seguidas
* Garantir as conformidades aos padrões da empresa
* Avaliar se a documentação do sistema é racional e está completa

Quando usar:

* Quando se deseja que os padrões sejam seguidos
* Quando se deseja determinar o nível de seguimento dos padrões
* Quando se deseja identificar pontos falhos na metodologia

## Teste de funcionalidade

O teste funcional tem por meta verificar se o software executa corretamente suas funções, se a implementação das regras de negócio foi apropriada e se o sistema é capaz de sustentar sua correta execução por um período contínuo de uso.

Objetivos:

* Assegurar a funcionalidade do sistema, incluindo entrada de dados, processamento e resposta
* Verificar se os requisitos dos usuários estão implementados e se atendem os usuários
* Verificar se o sistema funciona corretamente após um período contínuo de utilização

Quando usar:

* Qualquer sistema deve ser suas funcionalidades testadas
* Pode ser usado desde a fase de especificação de requisitos até a fase de operação do sistema

## Teste de regressão

Tem como propósito garantir que os defeitos encontrados foram corrigidos e que as correções ou inserções de novos códigos em determinados locais do software não afetaram outras partes inalteradas do produto.

Objetivos:

* Verificar se as alterações realizadas geraram alguma inconsistência no aplicativo ou em outros sistemas
* Verificar se as funções previamente testadas continuam funcionando após realização de mudanças
* Determinar se a documentação do sistema permanece atual
* Determinar se os dados e as condições de teste permanecem atuais.

Quando usar:

* Quando há o risco de que mudanças em uma parte do sistema afetem outras partes inalteradas do sistema.

## Teste de interconexão

O teste de interconexão, ou teste integrado como também é conhecido, tem como propósito testar as integrações com sistemas externos.

Objetivos:

* Verificar se os dados que são transferidos de um sistema para o outro estão sendo transferidos corretamente
* Garantir que as ações de integração de um software estão alinhadas com as ações de integração do outro software

Quando usar:

* Sempre que uma alteração que possui impacto nas funcionalidades integradas for realizada

## Teste de usabilidade

O teste de usabilidade verifica a facilidade que o software possui de ser claramente entendido e facilmente operado pelos usuários

Objetivos:

* Verificar a facilidade de operação do sistema pelo usuário
* Verificar a facilidade de entendimento das funções do sistema pelo usuário, através da utilização de manuais, on-line help, agentes e assistentes eletrônicos, etc.

Como usar:

* Executar diversas operações do sistema, utilizado a documentação do mesmo.

## Baseando-se nos riscos

Quanto mais crítico for o software, mais testes terão de ser gastos para a garantia da qualidade do produto. Isso significa dizer que quem define a quantidade de testes é a criticidade do negócio.

Falhas ou defeitos estão associados a riscos que podem prejudicar o negócio. Existem diversas técnicas para elaborar Análises de Riscos, e consequentemente escrever uma Estratégia de Testes.

Uma das formas de se fazer uma Análise de Riscos é associar o risco a uma característica de qualidade de software.

# Ambiente de teste

## O que é ambiente de teste?

**Conceito:** o Ambiente de testes é o "local" que o sistema será testado. Ele deve incorporar tudo que envolve o software e seu funcionamento. Segundo Bastos 2007 a definição do ambiente envolve: ambiente físico, pessoas, hardware, documentação, software, suprimentos, rede e interface.

**Preparação:** o Ambiente de testes deve ser planejado na definição da estratégia de testes ou no plano de testes. Quanto maior o nível dos testes a serem aplicados, mais o ambiente de testes deverá ser capaz de reproduzir as características do ambiente de produção.

**Criação:** a Criação, pela equipe de testes, de um ambiente isolado, organizado, representativo e mensurável garante a descoberta de erros reais, ou seja, aqueles que realmente ocorreriam na produção e que não foram descobertos em tempo de desenvolvimento e o mais importante, oferece a garantia de que não houve influência externa.  
Outra vantagem é que a preparação desse ambiente isolado, libera a equipe de desenvolvimento para continuar produzindo novos códigos, sem prejuízo à integridade do ambiente, mesmo durante a execução dos testes e possibilita a realização dos de iteração e sistema, de modo a permitir a integração das diferentes camadas e ou ambientes.

Principais características: ambiente isolado, com processamento independente e com componentes similares ao ambiente de desenvolvimento e produção; Ambiente restrito à equipe de testes; Criação de massa de teste com dados conhecidos e representativos quantitativamente qualitativamente, de modo a atender aos cenários de teste a serem executados e garantir a abertura do código.

## Elementos do ambiente de testes

O ambiente de testes deve ser definido pelo nível de testes a ser executado, ou seja, quanto maior o nível, mas, o ambiente de teste deverá ser capaz de reproduzir as características do ambiente de produção.

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. [4] Logo adiante no item 5 é feita uma abordagem sobre o funcionamento e o ambiente de uma fábrica de testes.

Apresentamos a seguir os elementos do ambiente de testes que devem ser levados em consideração para o planejamento do ambiente de testes.

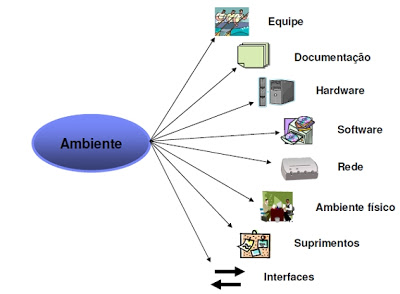


Figure 6: Ambientes de teste de Software

## Maneiras de testar o software

Existem diferentes maneiras de se testar uma aplicação, com cada uma destas levando em conta aspectos como quais profissionais estarão executando o processo de validação ou ainda, a extensão do que estará sendo verificado. Tomando por base estes critérios, os diferentes tipos de testes de software podem ser classificados em:

**Teste de unidade** (também conhecido como teste unitário): é uma modalidade que se concentra na verificação das menores unidades em um projeto de software. O teste é realizado em uma unidade lógica, utilizando dados suficientes apenas para verificar a lógica da estrutura em questão. Unidades em uma linguagem de programação orientada a objetos podem ser identificadas como um método, uma classe ou ainda um objeto;

**Teste de integração**: procura apontar erros verificando os relacionamentos (interfaces) entre as diferentes partes (módulos) que compõem uma aplicação. Testes de integração costumam auxiliar na construção da estrutura de um programa, considerando para isto os requisitos definidos dentro do projeto correspondente;

**Teste de sistema**: conduzido de maneira a simular a operação de uma aplicação por usuários finais, baseando-se para isto num ambiente similar ao do sistema já em produção e na manipulação de dados e informações próximas àquilo que será processado no dia-a-dia;

**Teste de aceitação**: normalmente um grupo de usuários finais do sistema participa desse tipo de teste, visando com isto simular operações cotidianas que determinem se a aplicação está em conformidade com o que se espera para a mesma;

**Teste de regressão**: realizado quando do lançamento de novas versões de um software, de forma a checar se as mudanças introduzidas na aplicação não produziram efeitos colaterais na execução de funcionalidades pré-existentes.

## Preparação do ambiente

Alguns atributos do ambiente de testes precisam ser analisados e planejados para a realização dos testes. São estes:

* Escopo
* Equipe
* Volume de dados
* Origem dos dados
* Interfaces
* Ambiente

## Escopo

O escopo do ambiente sempre dependerá do nível de testes que será executado.

* Testes unitários
  + Unidade Individual
* Testes de integração
  + Agrupamento de unidades
* Testes de sistema
  + Toda a aplicação
* Aceitação
  + Toda a aplicação

## Equipe

* Testes unitários
  + Desenvolvedores
* Testes de integração
  + Desenvolvedores / Analistas de sistema
* Testes de sistema
  + Analistas de sistema / Testadores
* Aceitação
  + Analistas de sistema / Testadores / Usuários

## Volume dos dados

* Testes unitários
  + Volume pequeno
* Testes de Integração
  + Volume pequeno
* Testes de sistema
  + Volume grande
* Aceitação
  + Volume grande

## Origem dos dados

* Testes unitários
  + Criação manual
* Testes de integração
  + Criação manual
* Testes de sistema
  + Criação automática e/ou dados reais
* Aceitação
  + Dados reais

## Interfaces

* Testes unitários
  + Não existem
* Testes de integração
  + Não existem
* Testes de sistema
  + Simuladas / Reais
* Aceitação
  + Reais

## Ambientes virtuais

Infelizmente, na realidade atual, a maioria das empresas não prevê orçamento para preparação dos ambientes de testes na contratação de novos projetos de software.

Uma solução que vem ganhando espaço, por ser mais econômica, é a criação de ambientes virtuais - também são conhecidos como ‘máquinas virtuais’.

Máquina virtual é um software que permite ao arquiteto de testes criar vários ambientes de testes, com diferentes configurações de software, hardware, sistemas operacionais, etc., utilizando na realidade a mesma máquina física.

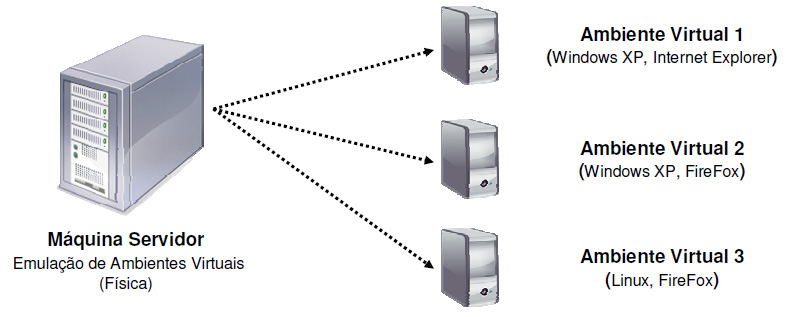


Figure 7: Ambiente de maquinas virtuais

## Preparação do ambiente de teste

O ambiente de testes deve ser planejado na definição da estratégia de testes ou no plano de testes. Quanto maior o nível dos testes a serem aplicados, mais o ambiente de testes deverá ser capaz de reproduzir as características do ambiente de produção.

Nessa fase do projeto, deve-se estimar o esforço gasto para montar o ambiente, ordenar e priorizar os casos de teste e preparar a massa de dados para testes. Dependendo do projeto, essa fase pode ser feita durante a execução, o que não é aconselhável, claro, pois devido a isso um grande atraso para o projeto passaria despercebido. Em grande parte dos projetos que participei e acompanhei, a massa de dados é criada somente durante a execução dos testes (com exceção dos projetos de performance que participei na EDS, que são outros quinhentos. Até projetos de automação, já vi criando esses dados on-the-fly. Portanto, já dá pra ver que o sucesso esperado não foi obtido...  
Todavia, boa parte desse processo de preparação do ambiente de testes eu adaptei do processo da RSI, empresa de consultoria e treinamentos especializada em testes, muito séria e profissional.

1) Objetivo

Oferecer um ambiente com infraestrutura similar ao de produção, com bases de dados reduzidas, descaracterizadas e íntegras, utilizando processos automáticos de carga e validação de informações. O resultado é a realização de testes integrados e com total visualização dos processos de negócio da empresa.

2) Artefatos de entrada

* Plano de testes
* Sistema em desenvolvimento
* Acesso ao Banco de Dados

Quaisquer outras ferramentas periféricas ao sistema que necessitem estar configuradas/preparadas para a execução dos casos de teste

3) Artefatos de saída

* Sistema pronto para ser testado

4) tarefas a serem realizadas

a) preparar a infraestrutura do ambiente de homologação e manter o controle sobre a utilização de seus recursos.

b) estabelecer infraestrutura de hardware, software e testware.

c) definir o tipo de ambiente (sistemas ativos e sistemas por demanda).

d) definir as políticas, normas e padrões do ambiente e dos processos.

e) estabelecer as regras e critérios para criação da Massa de Dados de Teste.

e1) Se os dados vão ser populados manualmente

e2) Se os dados vão ser populados automaticamente (através de uma ferramenta de automação dos dados do ambiente) e3. Se os dados vão ser populados diretamente no Banco de Dados (verificar se estes precisam estar associados à Regras de Negócio interdependentes ou se podem ser simplesmente adicionados sem afetar a estrutura do ambiente).

5) Atores envolvidos na atividade e suas responsabilidades

* Administrador do Banco de Dados – executa
* Analista de Testes – executa
* Desenvolvedor – acompanha
* Gerente – acompanha

6) Padrões

N/A

7) Procedimentos e Orientações

Deve, idealmente, haver um ambiente exclusivo e dedicado para testes.

Definição de requisitos de infraestrutura:

* Hardware: CPU e armazenamento
* Software: básico e produtos
* Testware: Recursos humanos com especialização e foco em testes, e ferramentas de gerenciamento, documentação, automação, produtividade e geração de massa de testes.
* Segurança: Segregação física ou lógica
* Metodologia de teste: Processos estruturados

Tipo de ambiente:

* Ativo: permanentemente disponível; sincronismo com a produção; processamento diário; controle rígido dos acessos, testes integrados.
* Por demanda: gerado pela necessidade; sem sincronismo; execução sob demanda; flexibilidade de acessos; testes integrados.

Normatização:

Definição de normas de controle de configuração e versão, gestão de mudanças e problemas.   
Definição de padrões de Nomenclaturas e Codificação (identificador de ambientes, estruturas de processamento, automação de processamento e scheduler).

Massa de dados de teste:

* Definição da origem dos dados: laboratório, produção ou simulado
* Definição das ferramentas de produtividade

Alguns atributos do ambiente de testes precisam ser analisados e planejados para a realização dos testes. Veja na tabela abaixo a organização destes atributos de acordo com os tipos de teste:

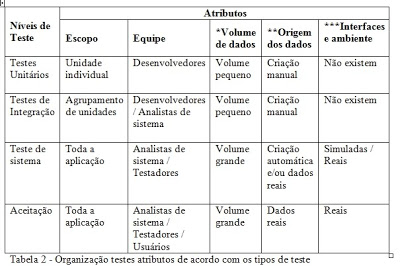


Table 1: Organização testes atributos de acordo com os tipos de testes

# Automação de testes

## Técnica x Ferramenta

A princípio será abordado dois conceitos importantes na automação de testes:

* Técnica: é o procedimento ou o conjunto de procedimentos que têm como objetivo obter um determinado resultado.
* Ferramenta: é um instrumento para aplicar a técnica.

## Mas, afinal, o que é automação de testes?

É a utilização de ferramentas que executam os testes na aplicação sem intervenção humana, por meio da implementação de scripts. Dessa forma, a ferramenta simula uma utilização do software testado e verifica os resultados esperados.

A automação de testes traz diversas vantagens: é mais rápido, diminui a chance de erro humano (mas é preciso atentar para que os scripts não apresentem falhas) e reduz o esforço com tarefas repetitivas, liberando os testadores para atividades que exijam raciocínio humano.

É importante lembrar, também, que para optar pela automação dos testes é necessário ter mão-de-obra especializada para a ferramenta utilizada.

Enquanto pessoas são mais inteligentes, intuitivas e criativas que máquinas, computadores já se dão melhor com cálculos, tarefas repetitivas e que precisam ser feitas num curto espaço de tempo.

## Quando utilizar ferramentas de automação de testes?

Cada vez mais, ferramentas de automação de teste estão sendo lançadas no mercado para automatizar as atividades de teste. Mas antes de pensar em uma ferramenta é preciso definir os processos. Veja alguns fatores que leva a automação de testes:

* Existem fortes pressões para melhorar a qualidade;
* O projeto tem situações que não possam ser testadas adequadamente pelos métodos tradicionais,
* Os perfis dos softwares desenvolvidos forem complexos e com impacto no negócio,
* Estudos de custo X benefício justificar o investimento,
* O tamanho do projeto ou do ambiente de teste justificar.
* A maioria das empresas sempre avaliam a possibilidade do uso de ferramentas abertas e livres.

## Ferramentas de teste

Existem basicamente três categorias de ferramentas de automação de testes, a saber:

* Ferramentas de gerenciamento
* Ferramentas de verificação de código-fonte
* Ferramentas de automatização na execução dos testes

## Ferramentas de gerenciamento

Ferramentas de gerenciamento são as ferramentas gerenciais, normalmente utilizadas para fazer a gestão de testes e defeitos. Por exemplo, uma ferramenta que permite que sejam cadastrados os defeitos encontrados no software durantes os testes.

São ferramentas muito importantes, pois auxiliam o processo de teste como um todo.

Estão divididas em:

* Ferramentas de gerenciamento de defeitos
  + Exemplos de ferramentas: Jira, Mantis, BugZilla
* Ferramentas de controle de versionamento
  + Exemplos de ferramentas: SubVersion, SourceSafe
* Ferramenta de gerenciamento dos testes
  + Exemplos de ferramentas: RTH, TestLink, Mantis

## Ferramentas de verificação de código

São ferramentas utilizadas para:

Verificar se o trabalho foi produzido dentro dos padrões de codificação

* Identificar pedaços de códigos não executados
* Identificar erros mais comuns, como problemas com inicialização e variáveis, estouro de memória, etc.

As ferramentas de verificação de código não têm a obrigação de verificar se o código realiza o que deveria realizar, ou seja, não têm a responsabilidade de testar as funcionalidades. Seu objetivo não é este. Esse tipo de ferramenta é muito utilizada nos testes unitários.

## Ferramentas de execução de testes

* Unitários
  + JUnit
* Integração
* Sistema
  + TestComplet
  + JUnitPerf
* Aceitação
  + JMeter
  + JUnitPerf

# Análise de riscos

## O que é risco?

É um evento ou condição incerta que, se ocorrer, provocará um efeito positivo ou negativo nos objetivos do projeto.

A Análise de Riscos e o Teste de Software estão diretamente ligados, afinal, todos os projetos de software possuem riscos. Peculiares ou não, estes riscos podem envolver desde a tecnologia utilizada até a complexidade do projeto. No entanto, independente da sua particularidade, um risco nada mais é do que um evento que pode ou não ocorrer, e que consequentemente pode causar alguma perda.

Como na maioria das vezes não é possível (nem recomendável) testar exaustivamente um software, a análise de risco voltada para o teste pode revelar uma vasta probabilidade de problemas, e evitá-los ao priorizar os testes.

Existe um consenso de que a atividade de teste é bastante cara, podendo custar até 2/4 do valor estimado para o projeto. Porém, fatores como competitividade, complexidade dos sistemas, e claro, a satisfação dos clientes, vêm sendo levados em consideração e fazem jus ao investimento. Isso sem falar da famosa Regra 10 de Myers, que indica que o custo da correção dos defeitos tende a ser cada vez maior quanto mais tarde ele for descoberto.

A análise de risco no processo de teste de software tem o objetivo de avaliar os fatores cujas ocorrências podem vir a acarretar falhas no sistema e perda para a empresa. É uma avaliação dos recursos de informação de uma organização, dos seus controles e das suas vulnerabilidades. A análise de riscos combina a probabilidade de ocorrência de um problema com a gravidade e o grau de prejuízos que os danos poderão acarretar.

Um risco apenas vira uma ameaça quando existe uma vulnerabilidade. Dentro de um o processo de teste de software, uma análise de risco envolve a identificação das ameaças mais prováveis em conjunto com o exame das vulnerabilidades relacionadas ao software. Os motivos que podem levar ao risco são muitos, como por exemplo, um processo de teste mal definido ou até mesmo não conduzido de forma planejada.

É importante observar que a análise de risco voltada para o teste de software segue basicamente as mesmas regras e metodologias utilizadas em projetos de software em geral, acrescida de algumas características próprias, como demonstrado ao longo do artigo.

## Conceituando riscos - Exemplos

Exemplo de Risco:

* Devido ao uso de uma nova tecnologia para a qual os programadores foram recém treinados, os mesmos poderão precisar de mais tempo para desenvolvimento dos programas, ocasionando um atraso nas atividades do cronograma.

Exemplo de Não-Risco:

* Desconhecimento dos programadores quanto à tecnologia utilizada para a programação deste projeto.

## Categoria de riscos

Quando analisamos riscos em projetos de teste de software, geralmente os categorizamos em:

* Riscos de projeto: são riscos ligados diretamente ao projeto.
* Riscos técnicos: são riscos relacionados à qualidade do software a ser desenvolvido.
* Riscos para o negócio do cliente: são riscos relacionados a defeitos no software, que podem causar prejuízos ao negócio do cliente.

## Gerenciamento de riscos

É um conjunto de técnicas e ferramentas para identificar, estimar, avaliar, monitorar e administrar os acontecimentos que colocam em risco e execução do projeto.

Para Gerentes de Projeto, o Gerenciamento de Riscos é uma disciplina que busca, pró-ativamente, minimizar os efeitos de eventos futuros que possam causar perdas para o projeto, ou maximizar os efeitos de eventos futuros que possam gerar ganhos para o projeto.

É o processo de identificação, avaliação, priorização, desenvolvimento de estratégias de tratamento e acompanhamento dos riscos.

## Identificação dos riscos:

A identificação dos riscos pode ser feita através de um “Checklist” que focaliza subconjuntos de riscos conhecidos:

1) Tamanho do produto

2) Impacto no negócio

3) Características do cliente

4) Definição do processo

5) Tecnologia a ser utilizada

6) Tamanho experiência da equipe

Além disso, é importante fazer analogia com casos conhecidos (dados históricos; relatórios de experiências anteriores; experiências pessoais) e fazer a identificação baseada no julgamento de pessoas e em suas experiências anteriores.

1) Risco Tamanho do Produto

São riscos associados ao tamanho do produto a ser desenvolvido ou modificado. O risco do projeto é diretamente proporcional ao tamanho do produto.

Checklist: formule as seguintes perguntas e compare as respostas com experiências anteriores. Se o percentual de desvio for elevado ou se os números obtidos são similares a experiências anteriores cujos resultados não foram considerados satisfatórios, o risco é elevado.

* Tamanho estimado em LOC ou PF?
* Grau de confiabilidade da estimativa?
* Tamanho do produto em número de transações, programas ou arquivos?
* Percentual de desvio do tamanho estimado observado em projetos anteriores?
* Número de usuários?
* Número de mudanças nos requisitos do produto, antes e depois da entrega?
* Quantidade de reuso?

2) Risco Impacto no Negócio

Conflito entre as considerações de negócios do departamento de marketing e a realidade técnica.

Checklist: formule as seguintes perguntas e compare as respostas com experiências anteriores. Se o percentual de desvio for elevado ou se os números obtidos são similares a experiências anteriores cujos resultados não foram considerados satisfatórios, o risco é elevado

* Efeito do produto sobre o lucro da empresa?
* Visibilidade do produto para a gerência sênior?
* Número de clientes que irão utilizar o produto e consistência entre o produto e suas necessidades?
* Número de produtos com o qual irá interagir?
* Sofisticação dos usuários finais?
* Quantidade e qualidade da documentação que deverá ser entregue ao cliente?
* Restrições governamentais quanto à construção do produto?
* Custos associados a entrega em atraso?
* Custos associados a entrega com defeitos?

3) Risco Características do Cliente

Um mau cliente provoca fortes impactos negativos na capacidade da equipe entregar o produto no prazo e dentro do orçamento. Clientes geralmente possuem diferentes necessidades (uns sabem bem o que querem; outros não. Uns são detalhistas e outros se contentam com promessas vagas), personalidades (uns gostam de ser clientes – a tensão, as negociações e as recompensas de um bom produto; outros aceitam qualquer produto e acham excelente um produto pobre. Uns irão reclamar quando a qualidade for baixa e elogiar quando for alta. Outros não irão se importar), relações com seus fornecedores (alguns conhecem bem o produto e o fornecedor. Outros são reservados e formais, comunicando-se apenas por correspondência escrita e curtas e escassas ligações telefônicas)

Além disso, clientes frequentemente são contraditórios, representando ameaças ao plano do projeto e riscos substanciais à gerência do projeto.

Checklist: Se a maioria das respostas a essas perguntas for “não”, investigações futuras devem ser realizadas, para avaliar o risco em potencial

* Você já trabalhou com o cliente no passado?
* O cliente possui uma sólida ideia sobre suas necessidades? Ele reserva algum tempo para descrevê-las?
* Os clientes concordam em reservar tempo para participar de sessões de elicitação de requisitos?
* Os clientes estão dispostos a estabelecer vínculos de comunicação com a equipe de desenvolvimento?
* Os clientes concordam em participar de revisões?
* Os clientes são tecnicamente sofisticados na área do produto?
* Os clientes entendem os processos de software?
* Os clientes resistirão a participar de trabalhos técnicos detalhados?

4) Risco do Processo

Se os processos de software não estão definidos, ou estão definidos, mas não são utilizados, o risco do projeto é elevado.

Checklist: Se a maioria das respostas a essas perguntas for “não”, o risco é elevado

* A gerência sênior apoia a definição de políticas formais que enfatizem a importância de padrões de processos de software?
* A organização desenvolveu um processo de software por escrito para ser utilizado no projeto?
* A equipe concorda com o processo como está documentado e está disposta a utilizá-lo?
* Processos de software foram utilizados em projetos anteriores?
* A organização promove treinamentos técnicos e gerenciais em engenharia de software?
* A equipe técnica e gerencial dispõe de padrões de engenharia de software publicados?
* Os produtos intermediários associados ao processo estão definidos e existem exemplos disponíveis?
* Revisões técnicas formais são realizadas regularmente em todas as fases do processo?
* Casos de testes e procedimentos de testes formais são realizados regularmente?
* Os resultados das revisões técnicas são documentados, incluindo erros e recursos utilizados?
* Existe algum mecanismo estabelecido para garantir que os padrões de engenharia de software estão sendo observados?
* A gerência de configuração está sendo utilizada para manter consistência entre os requisitos, projeto, código e casos de testes?
* Existem mecanismos para controlar mudanças nos requisitos dos usuários?
* O trabalho subcontratado está formalmente documentado, os requisitos estão especificados e existe um plano para cada subcontratado?
* Existem procedimentos definidos para acompanhar e rever os trabalhos subcontratados?
* Estão sendo utilizadas técnicas para facilitar a comunicação entre a equipe e usuários?
* Estão sendo utilizados métodos específicos de análise?
* Estão sendo utilizados métodos específicos de projeto de dados e de arquitetura?
* Está sendo utilizada linguagem de alto nível em mais de 90% do código?
* Existem convenções para código documentadas e são utilizadas?
* Métodos específicos de teste são utilizados?
* Estão sendo utilizadas ferramentas para apoiar o planejamento e acompanhamento do projeto?
* Estão sendo utilizadas ferramentas de gerência de configuração?
* Estão sendo utilizadas ferramentas CASE?
* Estão sendo utilizadas ferramentas para auxiliar na criação de protótipos?
* Estão sendo utilizadas ferramentas de teste?
* Estão sendo utilizadas ferramentas para auxiliar na produção da documentação?
* Métricas de qualidade estão sendo coletadas e analisadas?
* Métricas de produtividade estão sendo coletadas e analisadas?

5) Risco da Tecnologia

Atuar no limite da tecnologia é excitante e desafiador, mas oferece muitos riscos. Além de ser muito difícil prever esses riscos e mais ainda elaborar um planejamento para mitigá-los.

Checklist: Se a maioria das respostas a essas perguntas for “sim”, investigações futuras devem ser realizadas, para avaliar o risco em potencial

* A tecnologia é nova para a organização?
* Os requisitos exigem o desenvolvimento de novos algoritmos ou novas tecnologias de entrada e saída?
* Existem interfaces com hardware novo ou não testado?
* O software irá fazer interface com produtos de software de outros fornecedores, ainda não testados?
* O software irá fazer interface com sistemas de bancos de dados ainda testados na área do software?
* Os requisitos demandam interfaces com usuários especializadas?
* Os requisitos exigem a construção de componentes que nunca foram desenvolvidos pela organização?
* Os requisitos exigem novos métodos de análise, projeto e teste?
* Os requisitos demandam a utilização de novos métodos de desenvolvimento como os baseados em inteligência artificial, métodos formais ou redes neurais?
* Os requisitos estabelecem restrições excessivas de desempenho?

6) Risco tamanho e experiência da equipe

Checklist: Se a maioria das respostas a essas perguntas for “não”, investigações futuras devem ser realizadas, para avaliar o risco em potencial.

* Boas pessoas estão disponíveis?
* As pessoas possuem o perfil adequado?
* A equipe pode comprometer-se durante toda a duração do projeto?
* Parte da equipe está atuando em “part time”?
* A equipe possui expectativas adequadas sobre o projeto?
* A equipe recebeu o treinamento necessário?
* A rotatividade de pessoal deverá ser baixa?

## Analisar os riscos

Uma vez identificados os riscos, precisamos analisá-los a fim de encontrar o que os causam, e detalhar o seu impacto e a probabilidade da sua ocorrência.

Cada risco deve ser analisado sob a perspectiva do seu impacto e probabilidade de ocorrência Impacto

Impacto:

* Prejuízo que um risco pode causar ao projeto.
* Modelo de classificação: variação de 1 a 9.

Probabilidade:

* Chance de um risco se concretizar.
* Modelo de classificação: variação de 1 a 9.

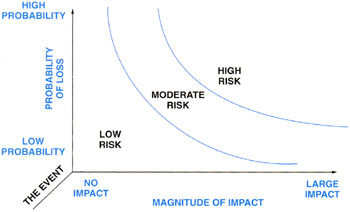


Figure 8: Impacto vs. Probabilidade

## Priorizar os riscos

O objetivo desta atividade é comparar os riscos e identificar quais são os mais importantes e que consequentemente merecem maior atenção. A priorização é necessária porque normalmente os recursos do projeto são escassos, e devemos concentrar esforços primeiro nos riscos mais importantes.



Figure 9: Matriz de priorização de riscos

## Planejar resolução dos riscos

Uma vez priorizados e selecionados os riscos que serão atacados, para cada um deles deveremos preparar:

* Um plano de mitigação
* Um plano de contingência

## Plano de mitigação

O plano de mitigação pode ser considerado uma atividade proativa, pois através dele tentamos minimizar o impacto e a probabilidade de um risco acontecer.

Características:

* Proativo
* Contempla atividades para minimizar impacto e probabilidade de ocorrência
* Deve ser executado antes que o risco ocorra
* Evita que o risco se transforme em um problema

## Plano de contingência

Ao contrário do plano de mitigação, o plano de contingência é reativo, e deve ser acionado quando o risco virar um problema.

Características:

* Reatividade
* Sem chance de evitar que o risco ocorra

# Planejamento de testes

## O que é um projeto?

Segundo a PMBOK, PMI Um projeto é um empreendimento **único** e **temporário** com o objetivo de criar um produto ou serviço **exclusivo**.

Teste é um projeto, e como tal deve ser planejado.

Todo projeto deve cumprir um ciclo básico de vida:

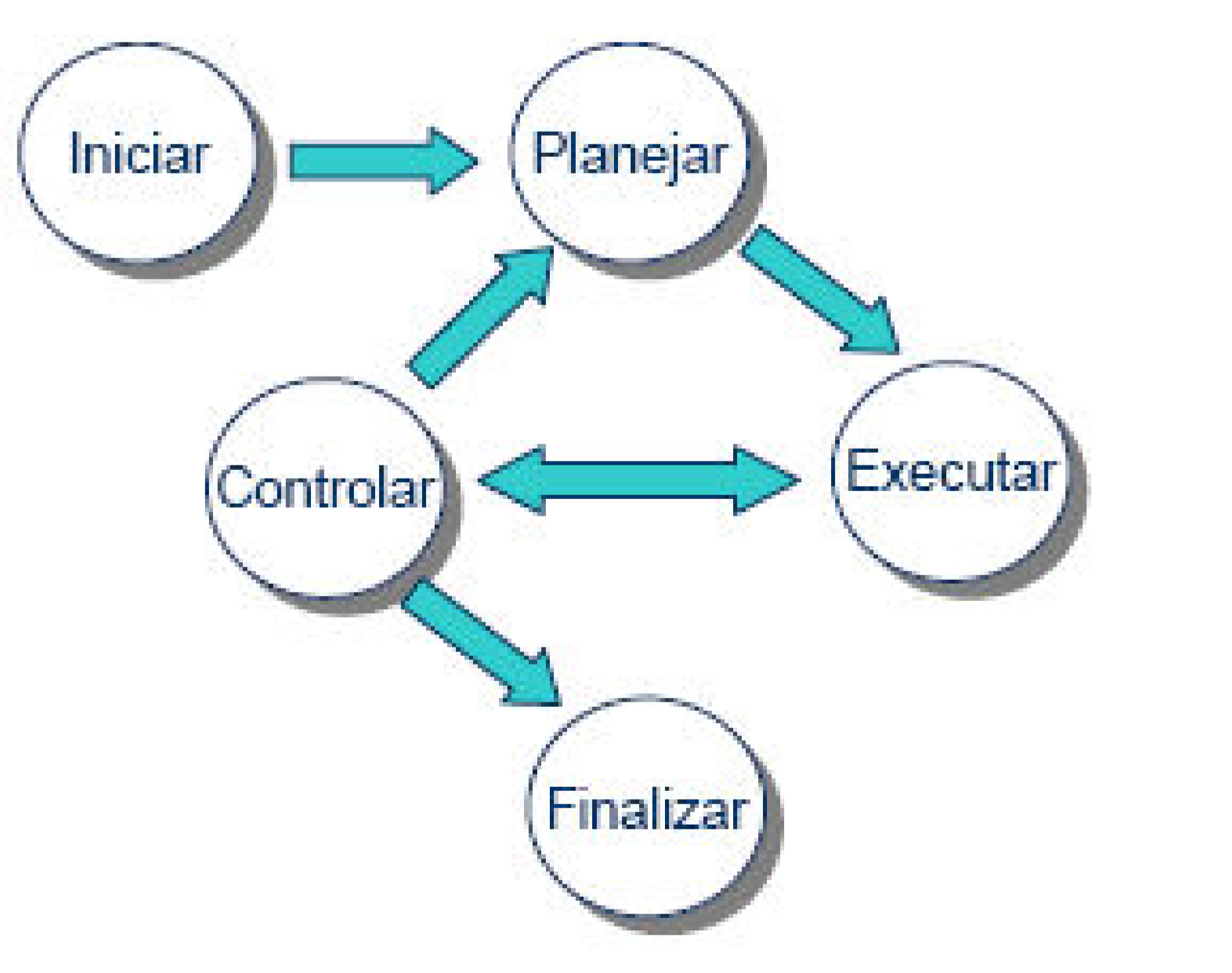


Figure 10: Ciclo básico de vida de um projeto.

## Porque investir em planejamento?

* Ajuda a garantir que o produto final irá atender aos usuários;
* Ajuda a evitar problemas e a minimizar o re-trabalho;
* Garante que você está na direção certa antes de iniciar os trabalhos;
* Garante que você vai conseguir o comprometimento das gerências envolvidas; para o seu plano;

Diminui o tempo total do projeto;

Considerações:

Nunca comece um projeto sem um documento de compromisso (project charter)

Identifique claramente as necessidades do usuário, crie o plano de projeto junto com a sua equipe.

# Estratégia de Testes – Método Rex Black

O método Rex Black consiste em:

Avaliar os riscos

* Identificar as características de qualidade do software
* Determinar a importância relativa das características de qualidade do software
* Detalhar a importância dos testes de cada subsistema e de cada característica de qualidade, e determinar a técnica de testes

# Plano de testes

Descreve o planejamento para execução do teste, incluindo a estratégia de testes, abrangência, abordagem, recursos e cronograma das atividades de teste. Identifica os itens e as funcionalidades a serem testadas, as tarefas a serem realizadas e os riscos associados com a atividade de teste.

# Padrões e planos de testes

Temos basicamente três padrões de plano de testes:

1. Padrão PMI
2. Padrão QAI
3. Padrão IEEE

## Plano de Testes – Padrão PMI

O padrão PMI propõe que para cada projeto as seguintes áreas de conhecimento sejam cobertas:

Escopo

Custo

Tempo

Qualidade

Integração

Recursos Humanos

Comunicação

Riscos

Suprimentos

Pode-se criar um plano para cada uma das áreas de conhecimento, ou apenas um plano contemplando todas as áreas de conhecimento.

## Plano de Testes – Padrão QAI

O padrão QAI apresenta a seguinte estrutura para o plano de testes:

Escopo de Teste

Objetivos de Teste

Premissas

Análise de Risco

Estrutura do Teste

Funções e Responsabilidades

Recursos e Cronograma de Testes

Gerência de Dados de Teste

Ambiente de Teste

Comunicação

Ferramentas de Teste

## Plano de Testes – Padrão IEEE 829

O padrão IEEE 829 apresenta a seguinte estrutura para o plano de testes:

Identificação do Plano de Testes

Referências

Introdução

Funcionalidades a serem testadas

Riscos do processo de teste

Funções a serem testadas da perspectiva do usuário

Funções que não serão testadas da perspectiva do usuário

Abordagem (estratégia de testes)

Critérios de conclusão dos testes

Critérios para interrupção e retomada de testes

Entregáveis

Ambiente de Testes

Pessoal e Responsabilidades

Cronograma

Plano de riscos

Aprovação do teste

# Construindo um Plano de Testes

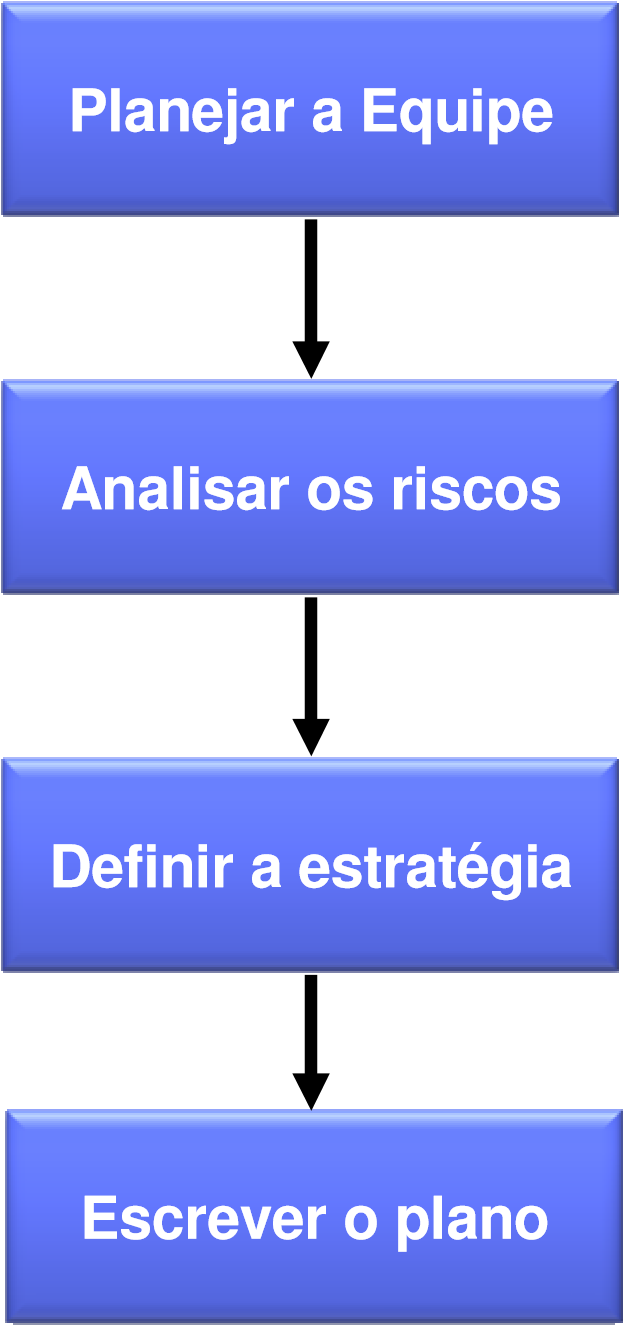


Figure 11: Plano de teste

Dificuldades

* Falta de envolvimento dos usuários e dos clientes
* Falta de treinamento
* Desenvolvedores x Testadores
* Falta de ferramentas de teste
* Falta de apoio da gerência
* Falta de tempo para testar
* Quem testa é a equipe de testes
* Alterações rápidas
* Os testadores são sempre os culpados
* Testadores precisam aprender a dizer ‘não’

# Documentação da Elaboração dos Testes

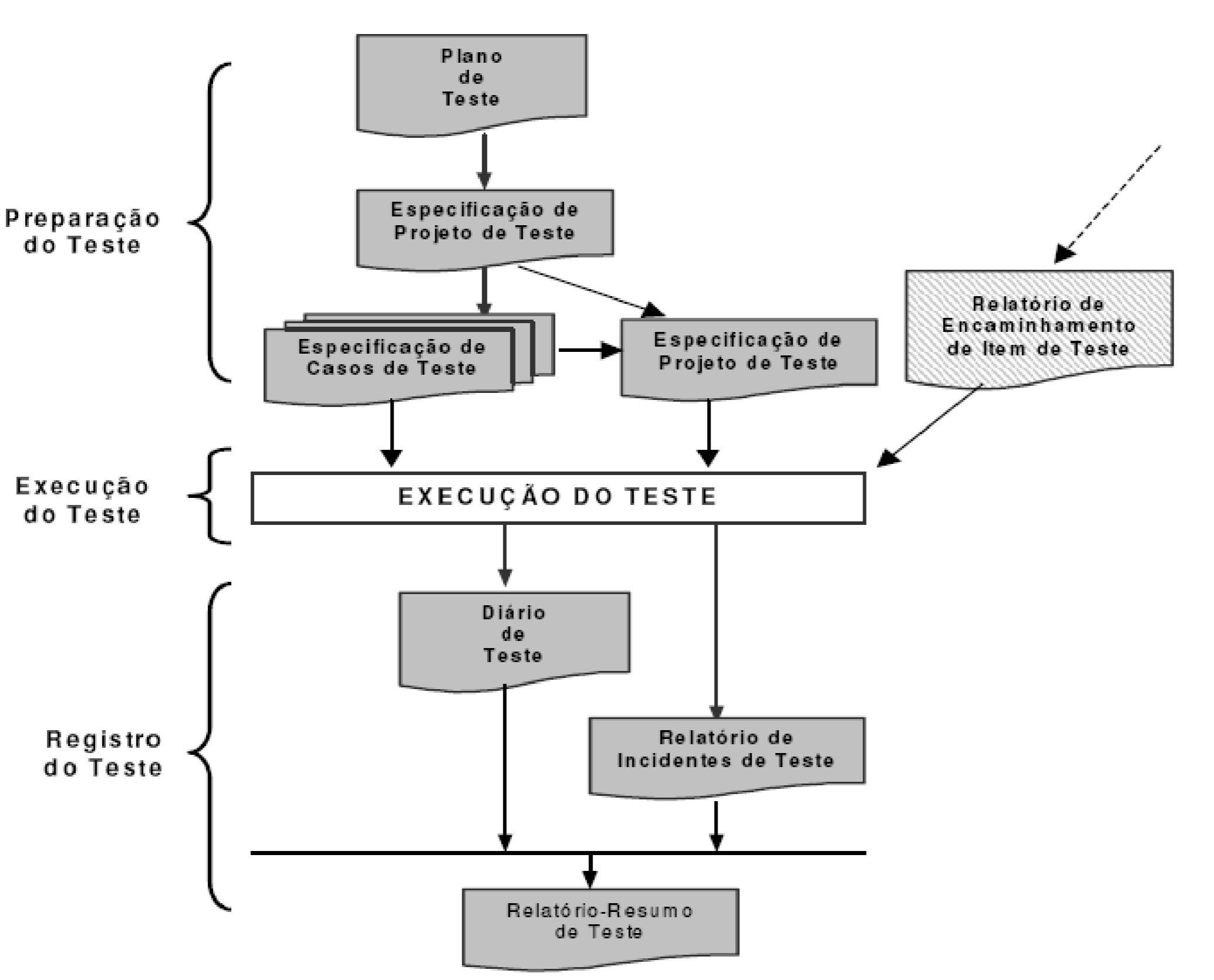


Figure 12: Documentação da Elaboração dos Testes

## Estrutura básica de um Caso de Teste

Segundo a norma **IEEE 829**, um Caso de Teste deve possuir:

* Identificador do Caso de Teste
* Itens de teste
* Especificações de entrada
* Especificações de saída
* Ambiente necessário
* Exigências especiais
* Interdependências
* Itens de Qualidade

Para atender aos requisitos de qualidade, um caso de uso deve ser:

* Efetivo
* Econômico
* Reutilizável
* Rastreável
* Auto-explicativo
* Elaborando Casos de Teste

Os Casos de Teste são especificados a partir dos requisitos do sistema, e para a sua elaboração devemos:

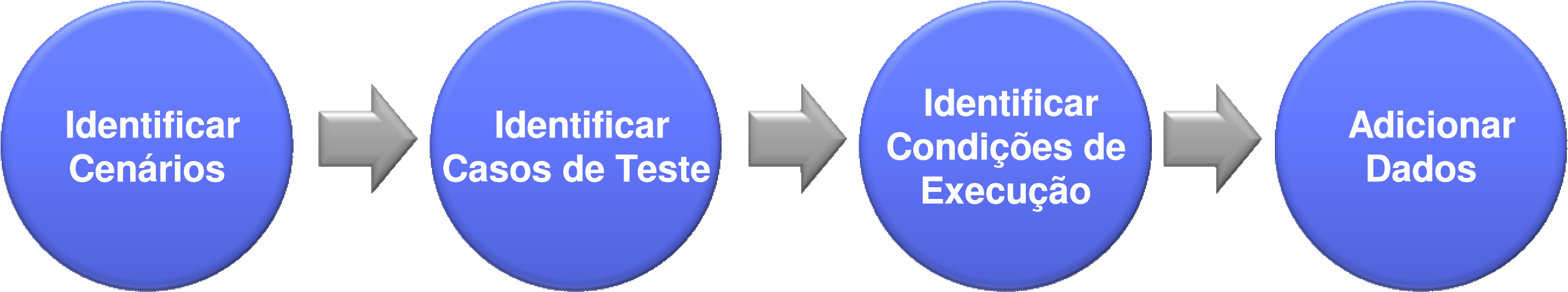


Figure 13: Casos de teste

## Composição de um Caso de Teste

Um Caso de Teste é composto de:

* Nome do Caso de Teste: “CT001”, por exemplo.
* Pré-condições
* Descrição ou procedimentos
* Resultado esperado
* Massa de dados
* Critérios especiais
* Ambiente
* Implementação
* Iteração

## Exemplo de Caso de Teste

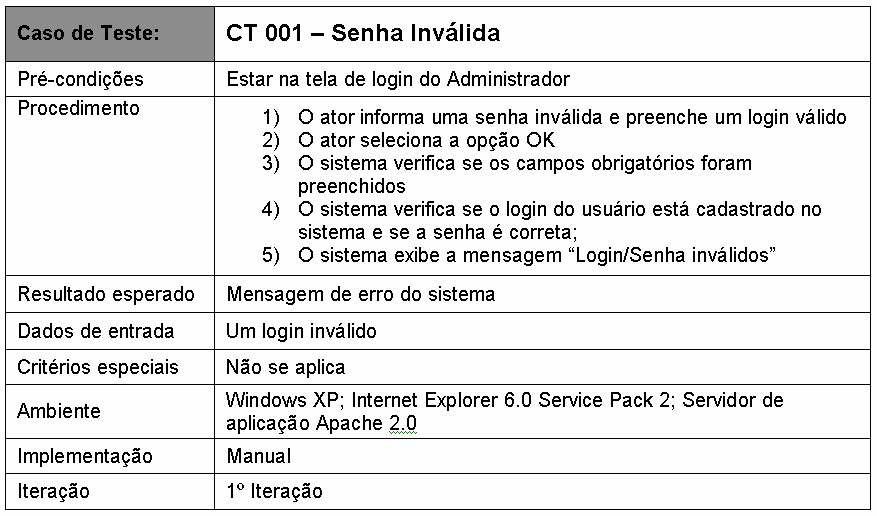


Figure 14: Exemplo de caso de teste

## Papéis e Responsabilidades

O Analista de Teste é basicamente responsável por escrever os Casos de Teste. As responsabilidades incluem:

* Identificação e definição de cada Caso de Teste
* Garantir que as mudanças serão comunicadas
* Garantir que tenham sido identificados Casos de Teste suficientes
* Garantir que há detalhes suficientes para implementar e conduzir o teste.
* Gerenciamento e manutenção de relacionamentos de rastreabilidade apropriados entre requisitos e Casos de Teste
* Gerenciamento do escopo apropriado dos Casos de Teste em uma iteração específica

## Testes exaustivos

Considere a seguinte situação:

Um sistema web com os seguintes requisitos não-funcionais:

* Deve operar em diferentes browsers
* Deve poder usar diferentes plug-ins
* Deve rodar em diferentes sistemas operacionais nas máquinas-clientes
* Deve receber páginas por diferentes servidores
* Deve rodar em diferentes servidores

|  |  |
| --- | --- |
| **1) Deve operar em diferentes browsers** | |
| Internet Explorer 5.0 | Netscape 6.0 |
| Internet Explorer 5.5 | Netscape 7.0 |
| Internet Explorer 6.0 | Mozilla 1.1 |
| Firefox | Opera 7 |
| **2) Deve poder usar diferentes plug-ins** | |
| Real Player | Media Player |
| Não usar nenhum |  |
| **3) Deve rodar em diferentes sistemas operacionais nas máquinasclientes** | |
| Windows 95 | Windows NT |
| Windows 98 | Windows XP |
| Windows ME | Windows 2000 |
| **4) Deve receber páginas por diferentes servidores** | |
| IIS | Apache |
| Tomcat |  |
| **5) Deve rodar em diferentes servidores** | |
| **Windows 8, Linux, Mac osX** | |

# Métodos para refinamento de Casos de Teste

Alguns métodos mais utilizados:

* Classes de Equivalência
* Análise de Valor-Limite
* Refinamento por probabilidade de erro

## Classe de Equivalência

Usada para reduzir o número de Casos de Teste a um nível controlável, mantendo uma cobertura razoável.

Dividir todas as combinações possíveis em classes.

Exemplo: Em um sistema de gestão de contratos, a idade dos clientes varia de 18 a 120 anos.

|  |  |  |  |
| --- | --- | --- | --- |
| **Entrada** | **Valores**  **Permitidos** | **Classes** | **Casos de Teste** |
| Idade | Idade entre 18 e 120 | 18 a 120 | Idade = 20 |
| < 18 | Idade = 10 |
| > 120 | Idade = 150 |

## Análise de Valor-Limite

Análise de Valor-Limite é uma técnica de teste que explora os limites dos valores para preparar Casos de Teste

Um grande número de erros tende a ocorrer nos limites do domínio

Esta técnica complementa o particionamento em classes de equivalência

|  |  |  |  |
| --- | --- | --- | --- |
| **Entrada** | **Valores**  **Permitidos** | **Classes** | **Casos de Teste** |
| Idade | Idade entre 18 e 120 | 18 a 120 | Idade = 18  Idade = 19  Idade = 119  Idade = 120 |
| < 18 | Idade = 17 |
| > 120 | Idade = 121 |

## Refinamento por Probabilidade de Erro

Baseado na intuição e experiência de testar condições que normalmente provocam erros:

|  |  |  |  |
| --- | --- | --- | --- |
| **Entrada** | **Valores**  **Permitidos** | **Classes** | **Casos de Teste** |
| Idade | Idade entre 18 e 120 | 18 a 120 | Idade = 18  Idade = 19  Idade = 119  Idade = 120 |
| < 18 | Idade = 17 |
| > 120 | Idade = 121 |
| Zero | Idade = Zero |
| Negativo | Idade = -18 |
| Branco | Idade = Nula (data não digitada) |
|  |  | Inválida | Idade = Inválida (data incorreta) |

## Desafios

Alguns desafios ao construir os Casos de Teste:

* Rapidez para os prazos
* O mais difícil do teste é escrevê-lo, e não executá-lo
* Mudança de requisitos
* Mudança de cronograma

# Execução dos testes

Executar o software não é testa-lo, a execução é apenas uma das etapas do teste, que este na verdade se subdivide 4 partes que se caracterizam como etapas: Iniciação, especificação (que é o que se espera que o programa faça), a própria execução e a entrega.

Existem duas formas de executar testes, a primeira é a forma estática, na qual o código é examinado e a segunda é a dinâmica, nesta o código é executado.

## Forma estática:

Estes se subdividem em três tipos: Revisão Individual, Pair Programming e Revisão Formal.

Na revisão individual apenas uma pessoa revisa o documento e depende totalmente do conhecimento desta durante sua execução.

Em Pair Programming dois testers trabalham com o software fazendo uma inversão periódica. Nele podem-se trabalhar o levantamento de requisitos, análise, design e outras atividades de processo.

A revisão formal pode envolver pares ou grupos, a revisão pode envolver usuários, programadores, analistas, projetistas, operadores, analistas de testes e testadores, desde que eles estejam envolvidos com o projeto. O objetivo desse envolvimento é apontar melhorias.

## Forma dinâmica:

Nessa forma existem 4 níveis, o de teste unitário, integrado, de sistema e aceitação.

No teste unitário, uma unidade de programa é testada por aqueles que a desenvolveram, em seguida, no integrado, há junção de unidades que já passaram por testes unitários, formando componentes, esses então são testados pelos desenvolvedores e analistas de sistemas. Depois segue-se com os testes de sistema, em que os analistas de testes trabalham com novas partes derivadas de um conjunto mínimo de componentes. Após isso, uma última fase é alcançada, o processo de teste de aceitação, feito pelos usuários e analistas do projeto.

*Algumas Considerações:* Caso um dos níveis não seja feito de forma eficiente, provavelmente o software apresentará erros que já poderiam ser concertados em etapas anteriores, desestabilizando o cronograma.

## Como os tipos de testes devem ser feitos?

Existem 4 fases para realização, a de preparo de dados, a de execução, solução de problemas e elaboração de resultado final, em que todas possuem um relatório de defeitos e uma estratégia são adotadas. Essas 4 fases serão explicadas a seguir:

Preparo de dados: O que é investido?

* Casos de Teste
* Roteiros de Teste
* Documentação do Sistema

**Produtos gerados:** Base de teste, arquivos ou dados disponíveis

Execução de testes: Quais são os insumos?

* Casos de testes
* -Roteiros de teste
* Resultados

**Produtos gerados:** Resultado dos testes, relatório do que há para resolver

Solução de ocorrências: Quais são as partes investidas?

-Relatório com informações do que há para resolver

**Produto gerado:** Relatório com informações do que foi resolvido

Assim após essas etapas, há formação de um relatório final contendo todas as ocorrências de um processo de teste.

## E o que acontece após a execução dos testes?

Três documentos são gerados: o relatório de ocorrências, o resumo dos testes e o log de teste. O relatório de ocorrências consiste em informar resultados não esperados, estes não necessariamente são defeitos, podendo ser mesmo problema do hardware ou a configuração do ambiente. O resumo de testes compara o resultado com o cronograma previsto, sendo um registro formal das atividades realizadas. O log de teste visa detalhes relevantes da execução dos testes, como a duração de tempo destes.

## Quando os testes terminam?

Quando o custo destes não está muito elevado e há boa redução dos defeitos encontrados e seus custos

## O que acontece se o software for entregado muito cedo?

Muitos defeitos podem não ter sido vistos, gerando insatisfação do cliente

## O que acontece se o software for entregado muito tarde?

Ele terá sido bem testado e terá qualidade, porém pode perder em oportunidades de negócio.

Conclusão: Para sucessos dos testes, a documentação e execução devem estar bem aplicadas.

# Gestão de defeitos

O principal objetivo dessa gestão é evitar os defeitos. Essa gestão utiliza um processo de melhora continua, as falhas são identificadas e solucionadas, depois armazenadas em um relatório de gestão, assim na próxima vez que o relatório for acessado, ele já estará contendo a solução dos defeitos.

Prevenir defeitos é uma das fases mais importantes dentro do desenvolvimento de ciclo de vida de um software, pois preveni-los custa muito menos do que o custo de retrabalho. Para essa prevenção, é preciso ter pessoas capacitadas, adequação de processo e de organização, adoção de padrões, treinamento e educação.

## Aonde as falhas podem ser identificadas?

As falhas geralmente são identificadas durante os testes estáticos e dinâmicos, quando isso acontece elas são passadas para o relatório de gestão, após isso ocorre uma discussão entre a equipe se o erro encontrado é válido ou não para o projeto, e se tem impactos na entrega para o cliente.

## O que acontece depois que o defeito é reconhecido?

Após a decisão de que se o defeito é válido ou não, a falha deve ser reportada e junto com a documentação dos testes, enviada para o desenvolvedor, caso ocorra divergências na equipe sobre a validação do defeito, o gerenciador do projeto deve decidir o que se deve ser feito com o erro.

## Como se soluciona os defeitos?

Deve-se decidir se a correção é de um nível alto, médio ou baixo, programa-lá e executá-la, reportando quando tempo ele demorará para acontecer. É importante que essas revisões sejam feitas, para nada de errado acontecer de novo. Um outro processo importante é a comunicação com o desenvolvedor e o registro no relatório de gestão, pois há controle da quantidade de defeitos corrigidos e que se a remoção destes não causam outros defeitos.

Taxonomia dos defeitos:

Taxonomia dos defeitos é uma espécie de categorização das falhas, quanto melhor ela for, melhor é a gestão de defeitos. A seguir há uma figura mostrando essa taxonomia:

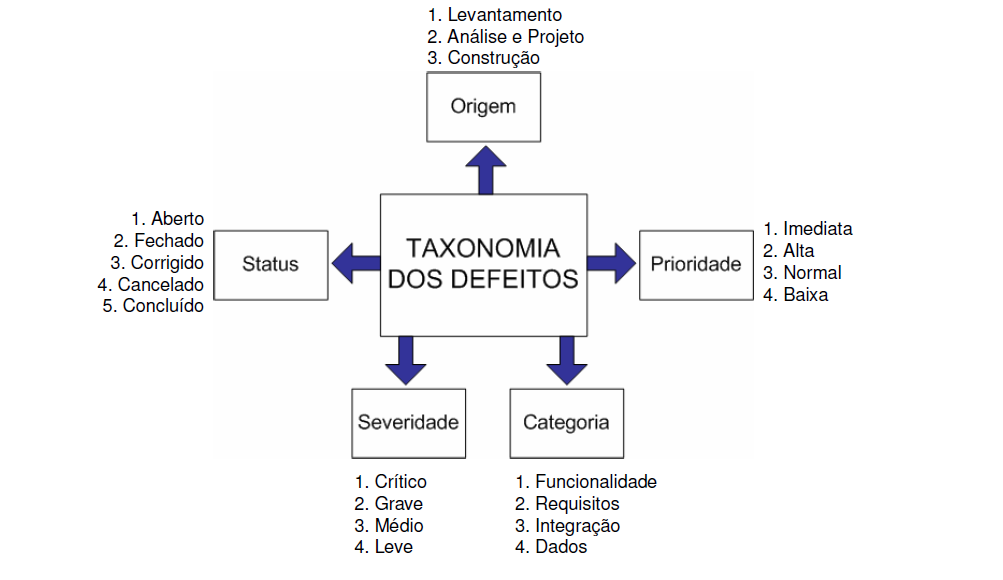


Figure 15: Taxonomia

*Considerações:* Evitar defeitos é fundamental para que não ocorra a multiplicação de falhas no projeto, a gestão de defeitos deve estar ligada ao desenvolvimento, minimizando os riscos.

# Relatórios de teste

São relatórios que documentam os testes, eles visam cumprir três objetivos: definir o escopo do teste, apresentar seus resultados, conclusões e recomendações.

Se o relatório for feito de imediato, ele serve para prover informações sobre o sistema, definindo se ele está pronto para entrar em produção e minimizar futuras consequências, e se for feito a longo prazo, ele serve para rastrear problemas e analisar a produção.

Existem três categorias de relatórios de teste: a de situação de projeto, que tem a função de obter informações para a gerência, a de intermediários de teste, que fornecem informações sobre os testes, e os relatórios finais, que apoiam as decisões relativas na implantação do software.

Durante a elaboração dos relatórios deve ser permitido que a equipe comente o rascunho antes da sua finalização, não se deve incluir nomes ou estabelecer culpados, além de limitar as informações aos itens importantes.

Quando terminados, os relatórios devem ser entregues ao gerente de projeto, o qual deve envolver as equipes do projeto, para se apontar recomendações.

# Gerência de comunicação

Uma gerência de comunicação deve incluir os processos necessários para a geração, coleta, distribuição, armazenamento e descarte de informações. Esse gerenciamento define para quem e quais informações serão distribuídas, trabalhando com um relatório de desempenho da comunicação a cada distribuição. Essa área trabalha com 4 estágios, explicados a seguir:

**Planejamento de comunicações:**

Entradas: Requisitos de comunicação, tecnologia de informações e restrições

Saída: Plano de gerência das comunicações.

**Distribuição de informações**

Entradas: Resultados do trabalho, plano de gerência das comunicações e plano de testes

Saídas: Registro do projeto, apresentações do projeto

**Relatório de desempenho:**

Entradas: Plano de testes e registro do projeto

Saídas: Relatório de progresso, log de teste, relatório de ocorrência e de situação

**Encerramento**

Entradas: Documentos, registros e outras informações

Saídas: Plano de gerências de comunicações e relatório de sumário de teste.

Nesse trabalho nós leitores podemos aprender e assimilar vários conceitos sobre a informática. E viu também como um sistema operacional funcionar, assim como, um processador trabalha para executar um cálculo em conjunto com a memória RAM. Foi possível mostrar o básico da informática abordando alguns assuntos relacionado ao enorme universo que a informático nos proporciona.

# Bibliografias

**[4]** **Base de Conhecimento em Teste de Software.** 2ª Edição. Rios, Emerson; Cristalli, Ricardo; Moreira, Trayahú & Bastos, Aderson. – S. Paulo, Martins Fontes, 2007.

Almeida, Carla. **Introdução ao Teste de Software**<<http://www.linhadecodigo.com.br/artigo/2775/introducao-ao-teste-de-software.aspx#ixzz3khJtHuvx>> acessado em: 11/09/2015

Eliza, Renata e Lagares, Vivian. [**Análise de Riscos e o Teste de Software - Revista Java Magazine 97**](Análise%20de%20Riscos%20e%20o%20Teste%20de%20Software%20-%20Revista%20Java%20Magazine%2097) <<http://www.devmedia.com.br/analise-de-riscos-e-o-teste-de-software-revista-java-magazine-97/22831#ixzz3lOX467Km>> acessado em: 11/09/2015

Gomes De Souza, Jesiane. **Ambiente de Testes de Software** <<http://www.webartigos.com/artigos/ambiente-de-testes-de-software/74582/#ixzz3khF0ruka>> acessado em: 11/09/2015

Groffe, Renato Jose. **Testes Unitários no Visual Studio 2012**   
<<http://www.devmedia.com.br/testes-unitarios-no-visual-studio-2012/27215#ixzz3khKnptAo>> acessado em: 11/09/2015

Schroeder Viera, Luiz Gustavo. **Preparar Ambiente de Testes**<<http://www.testavo.com.br/2010/06/preparar-ambiente-de-testes.html>> acessado em: 11/09/2015