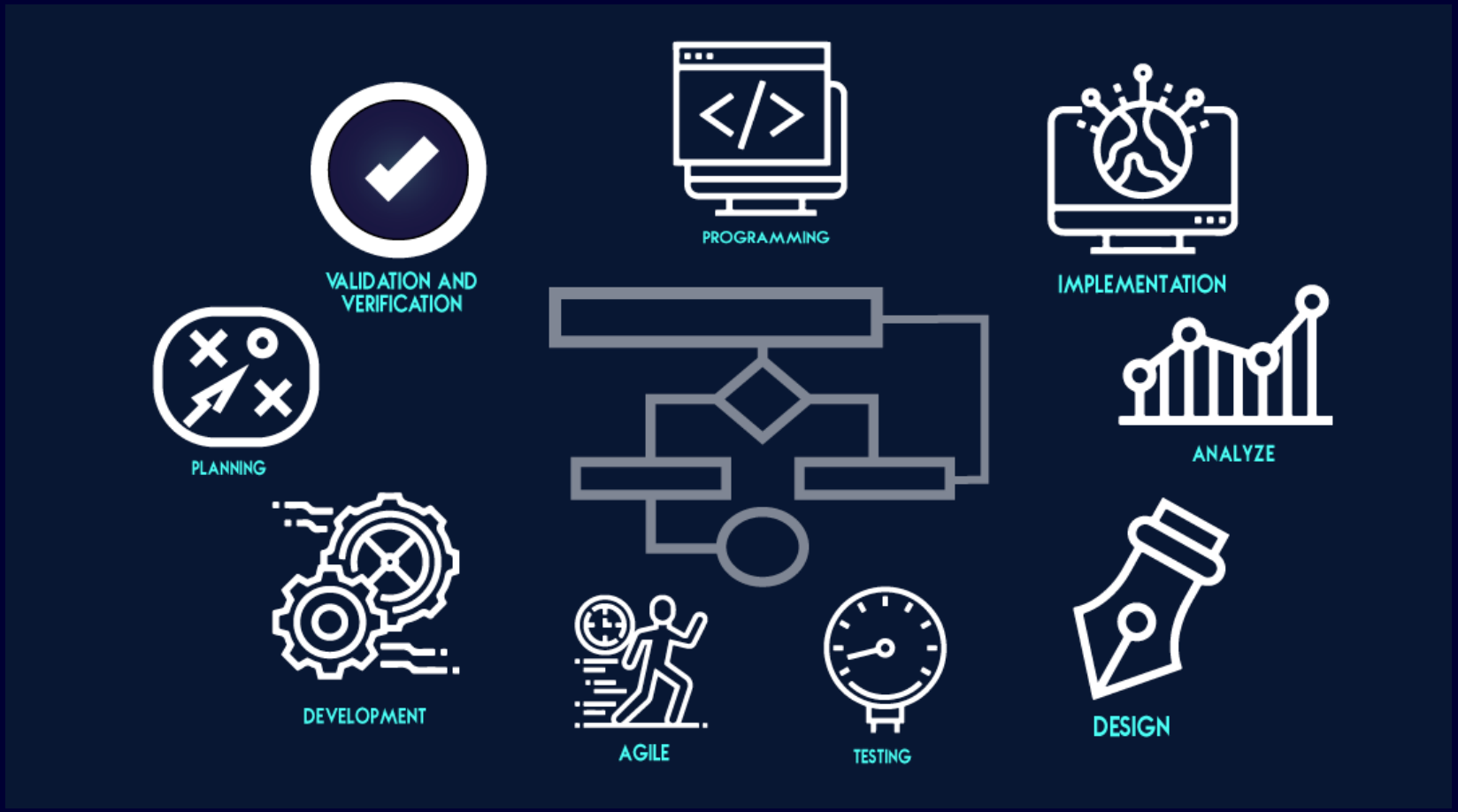
Unichristus
Centro Universitário Christus



Engenharia de Software: Codificação e Testes de Softwares

Lattes - linkedin
euristenhojr@gmail.com
http://www.unichristus.edu.br/

Euristenho Queiroz de Oliveira **Júnior**
Especialista em Engenharia de Software
MSc em Engenharia de Software

AGENDA

1. Apresentação
2. Livros
3. Acordo de Convivência
4. - Engenharia de Software: Testes
5. - Definição de Testes
6. - Tipos de Testes
7. - Ferramentas
8. - Exercícios

Unichristus
Centro Universitário Christus

Apresentação

FORMAÇÃO ACADÊMICA

- Graduado em Telemática/Telecomunicações - IFCE (2002 - 2008)
- Especialista em Engenharia de Software - FA7 (2011 - 2013)
- MSc em Engenharia de Software - UFPE (2011 - 2015)

CURRÍCULO PROFISSIONAL

- Atuei 4 anos na empresa privada
- 10 anos no ambiente Público
- Atualmente Líder Técnico de 45 Projetos de Tecnologia na SEPOG/PMF

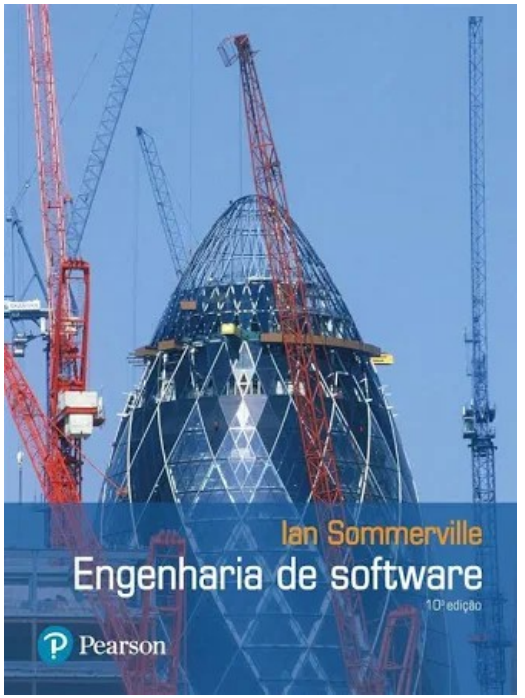
Apresentação

DOCÊNCIA

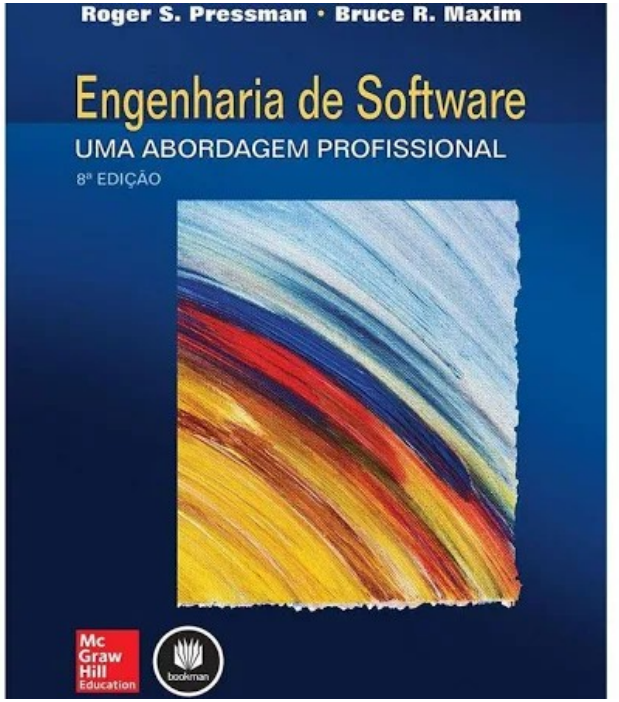
- Professor Substituto das Disciplinas de Sistemas de Informação – FA7 (2011 - 2012)
- Professor da Especialização em Sistemas WEB – FJN (2011 - 2012)
- Professor de Bancas de graduação em Sistemas de Informações – FA7 (2012)
- Professor dos Cursos de Tecnologia da Informação da Unifanor (2015 – 2018)
- Professor do Curso de Sistemas de Informação Unichristus (2018 - Atual)

Livros

- **Engenharia de Software** - 10ª Ed – Lan Sommerville - Pearson



- **Engenharia de Software – Uma abordagem profissional**- 8ª Ed. AMGH



Dicas de Convivência

- Horários
- Conversas
- Dúvidas
- Celular
- Avaliações

Questionamentos

Testes de Software

“O objetivo do Teste de Software é encontrar bugs, encontrá-los o mais cedo possível e garantir que os bugs sejam corrigidos” [Patton, 2005]

O que é um bug?

Como e quando um bug Pode ser encontrado?

O que é um bug?

Várias definições existem em cada empresa

- **Um bug é: um defeito, falta, problema, incidente, anomalia, CR, etc.**

Empresas usualmente perdem um bom tempo discutindo isso

- **Mas normalmente a conclusão é que um bug é apenas um bug**

Uma definição mais formal de bug pode ser feita se pensando quando um bug ocorre [Patton, 2005]

Um bug ocorre quando uma ou mais das opções abaixo for verdadeira:

- **O software NÃO faz algo que a especificação diz que ele deveria fazer**
- **O software FAZ algo que a especificação diz que ele NÃO deveria fazer**
- **O software faz algo que a especificação não menciona**
- **O software NÃO faz algo que a especificação NÃO menciona, mas deveria mencionar**
- **O software é difícil de usar, entender, ou na visão do testador pode ser visto pelo usuário final como não estando correto**

Alguns bugs Clássicos

CD-ROM Rei Leão da Disney

- Lançado em 1994
- Primeiro CD-ROM da Disney (muita propaganda e grandes vendas)
- Lançado antes no natal
- Não foi testado em diferentes configurações de PCs e não informava a configuração mínima

Bug do ano 2000

- Computadores não estavam preparados para o ano 2000

Sistema de Telefones de AT&T nos EUA

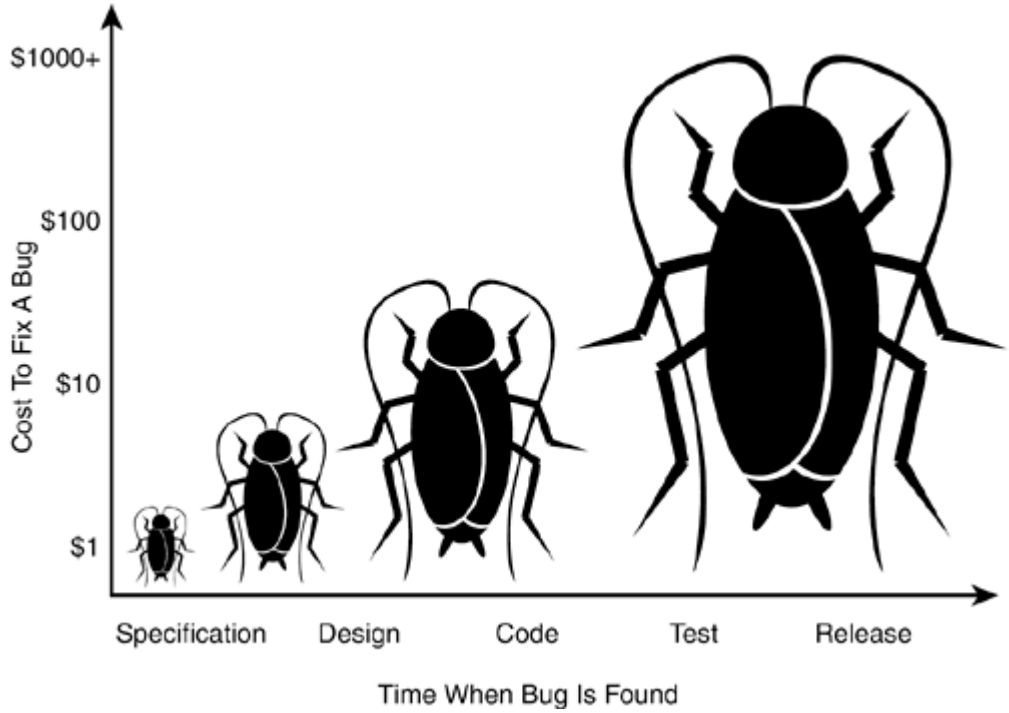
- Toda a rede caiu por causa de um switch case errado

Sistema de Foguetes em Nara e Agência Européia

- O foguete explodiu no lançamento pois o código feito na Nasa estavam no sistema métrico inglês e o da agência Européia no sistema internacional

Qual o custo de um Bug?

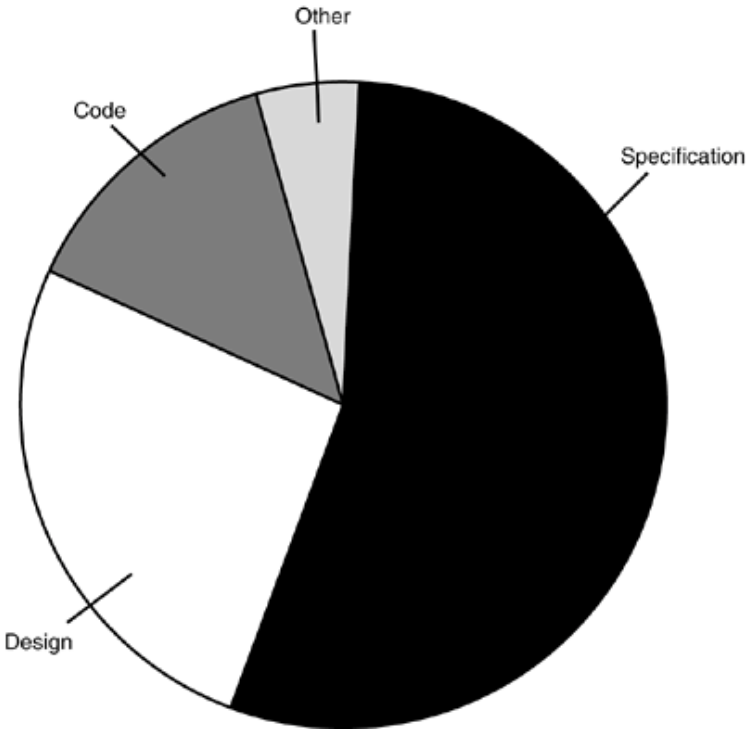
O custo de um bug está diretamente associado a fase do ciclo de desenvolvimento em que o bug é encontrado



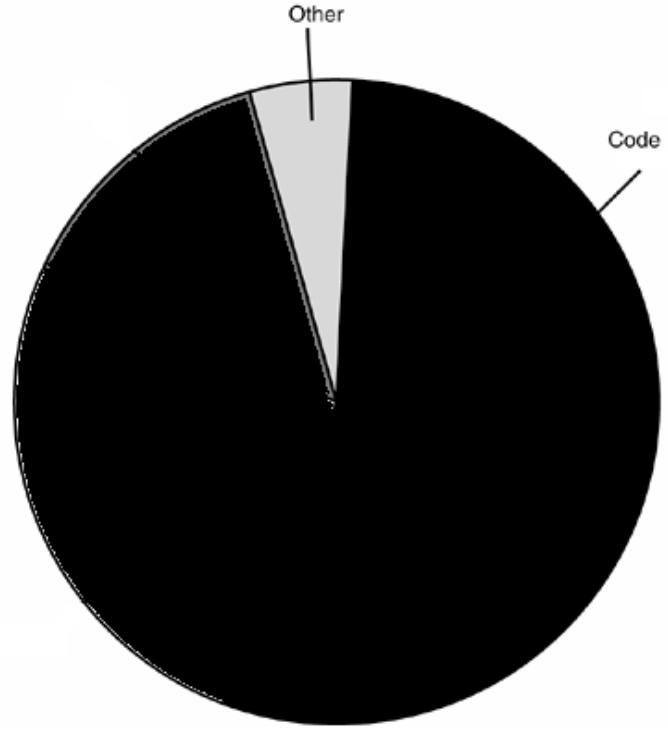
Um bug encontrado durante a especificação pode custar 1 dolar. O mesmo bug encontrado no release pode custar 100x mais

Por que Bugs ocorrem?

Usualmente a razão de um bug ocorrer, depende do tamanho e do tipo do sistema



Sistemas de Média e Grande Porte



Sistemas de Pequeno Porte

Quem é o testador?

O testador está diretamente associado a definição de teste de software

O testador

- **Encontra o bug**
- **Busca encontrá-lo o mais cedo possível**
- **Garante que o bug está corrigido**

Um bug corrigido não necessariamente implementa em mudança no código

- **Correção de um manual**
- **Correção de treinamento**
- **Etc.**

O testador deve ser responsável por verificar realmente se o bug foi corrigido

Quem é o Testador?

Por que o testador é necessário?

- **Os testes feitos por desenvolvedores**
  - Tendem a verificar apenas o “caminho feliz”
  - Normalmente são otimistas
  - Não são sofisticados
- **Organizações iniciais tem usualmente 5 testes de “caminho feliz” para cada teste de “caminho alternativo”**
  - Organizações maduras usualmente tem o oposto - 5 de caminho alternativo para cada um de caminho feliz
- **Desenvolvedores normalmente só conseguem enxergar 50-60% dos casos de teste do seu código**

O Testador tem um perfil diferente do desenvolvedor

- **São exploradores**
- **Gostam de encontrar problemas**
- **Criativos no uso do software**
- **Visão das diferentes situações em que o software pode ser usado**

Por que testes precisam se realizados?

Existem várias técnicas de encontrar bugs

- **Nenhuma técnica consegue garantir que TODOS os bugs sejam encontrado**
- **Testes de software é uma das possíveis técnicas que podem ser aplicadas**

Nenhuma técnica consegue encontrar mais de 85% dos bugs

Testes unitários + testes de componente + testes de sistemas atingem no má

Removal Step	Lowest Rate	Modal Rate	Highest Rate
Formal code inspections	45%	60%	70%
Modeling or prototyping	35%	65%	80%
Personal desk-checking of code	20%	40%	60%
Unit test	15%	30%	50%
New function (component) test	20%	30%	35%
Integration test	25%	35%	40%
Regression test	15%	25%	30%
System test	25%	40%	55%
Low-volume beta test (<10 sites)	25%	35%	40%
High-volume beta test (>1,000 sites)	60%	75%	85%

Testes vs Inspeções

Inspeções dos artefatos buscam validar os artefatos antes que a próxima fase do desenvolvimento seja iniciada

Vários estudos indicam que o custo de encontrar um erro com inspeção é menor que o custo de encontrar com testes [Kaplan 1995]

- **3,5 horas de esforço para encontrar um erro com inspeção**
- **15-25 horas de esforço para encontrar um erro com testes**

Isso não significa que testes não são necessários

- **Mas que outras técnicas também podem ser aplicadas**

Outras Técnicas de Encontrar Bugs

Inspeção de Software

- Processo formal de verificação do software
- Pode ser aplicado para praticamente todos os artefatos gerados durante o ciclo de desenvolvimento
- Tem o custo mais alto de aplicar, mas também é o que produz o melhor resultado
- Combinação de inspeções na especificação, design e código podem encontrar até 85% dos bugs de um software
- Como o processo é formal, ele é fácil de mensurar e de acompanhar

Prototipação

- Ajuda no levantamento e definição dos requisitos
- Dá uma boa visão para o cliente do que vai ser o software final
- Reduz bastante as duvidas de usabilidade e de como as funções devem ser implementadas

Outras Técnicas de Encontrar Bugs

Programação por pares

- **Técnica definida em XP (Extreme Programming)**
- **2 desenvolvedores por computador**
  - 1 programador digita
  - 1 programador procura por erros
- **Possui várias vantagens**
  - Facilita mentoring de novos desenvolvedores
  - Dissemina a cultura da empresa
  - Pode chegar a encontrar a mesma quantidade de problemas da inspeção
- **É mais aplicável para a fase de codificação**

Walkthroughs e Code Reading

- **Similar a inspeção, mas sem o formalismo de um processo**

Enquete

Marca V ou F nas razões pelas quais a especificação do software é a principal causa de bugs

V F

( ) ( ) A grande maioria dos software não tem uma especificação detalhada

( ) ( ) Mudanças na especificação não são informadas para toda a equipe

( ) ( ) A equipe de testes não tem capacidade de entender a especificação

( ) ( ) O cliente não tem obrigação de fornecer e ajudar no detalhamento da especificação

( ) ( ) A equipe que faz a especificação não precisa se preocupar com os testes

( ) ( ) A equipe de testes só é alocada no final do projeto

Por que testes precisam se realizados?

É possível que exista um software cuja a especificação seja totalmente fechada antes no início do desenvolvimento

a) Sim, os requisitos de qualquer software podem ser capturados e totalmente documentados antes do início da construção

b) Não, o software por natureza é complexo e seus requisitos não são claros

c) Sim, mas eles podem e vão mudar durante o desenvolvimento

Testes nos Ciclos de Vida

As atividades de testes são inseridas no contexto das atividades de engenharia de software

TODO software segue um ciclo de vida

- **Existem vários modelos de ciclo de vida que podem ser adotados durante o desenvolvimento**

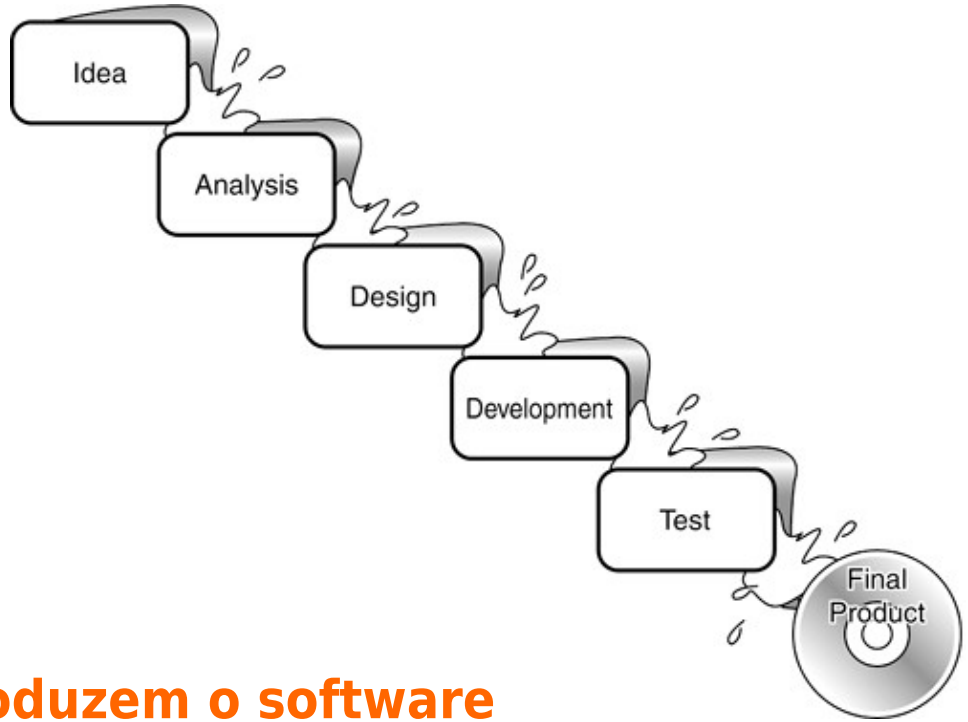
O modelo de ciclo de vida usualmente direciona

- **Quais testes serão rodados**
- **Quando os testes serão rodados**

Os seguintes modelos serão discutidos

- **Cascata**
- **V**
- **Incremental**
- **Codificar e testar**

Modelo Cascata



Fases seqüências que produzem o software

Ao final de cada fase uma revisão interna deve validar se é possível passar a fase seguinte

A princípio só se deve passar a fase seguinte com a anterior finalizada

Foco principal é garantir que a especificação é bem feita antes de iniciar as fases seguintes

Por que testes precisam se realizados?

O modelo em cascata é ideal para softwares bem conhecidos

- Sem grandes surpresas durante a especificação
- Tecnologia bem conhecida da equipe

A fase de testes é bem natural e é executada após toda a codificação

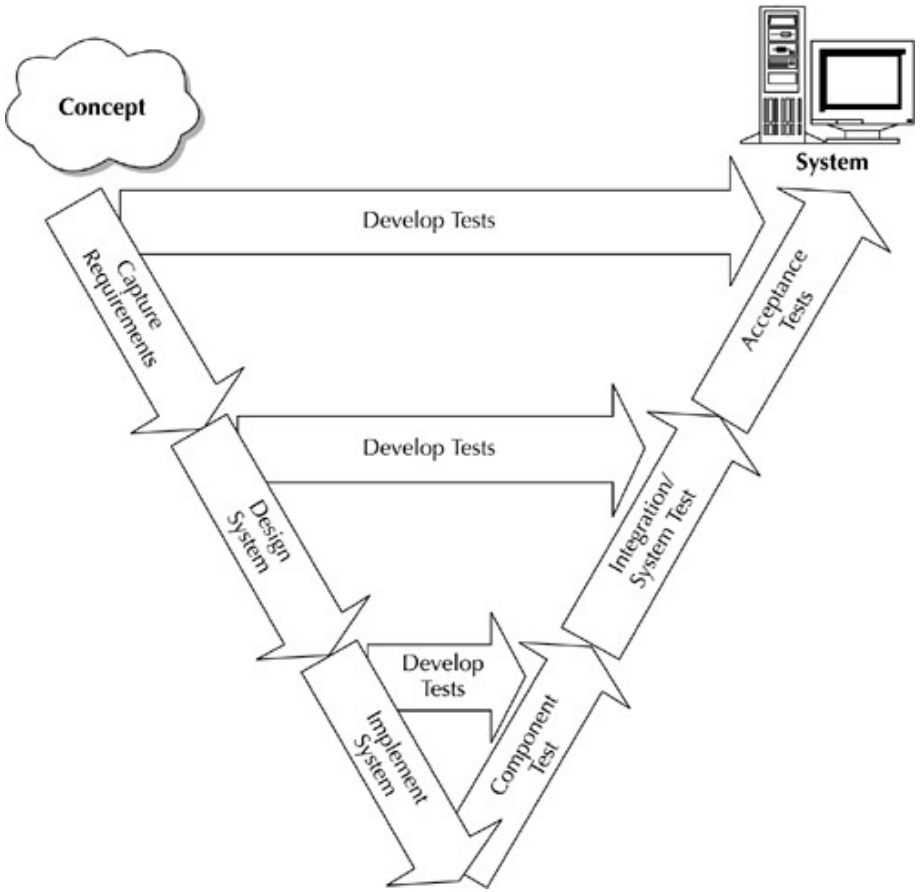
Principal vantagem para os testes

- A especificação normalmente é bem feita e detalhada

Principal problema

- Os testes só são executados no final
- O custo de corrigir um problema é mais alto

Modelo V



Variação do modelo em cascata

Associa diferentes tipos de teste de acordo com a fase do ciclo de vida

- **Testes de aceitação usualmente estão associados aos requisitos**
- **Testes de Integração usualmente estão associados ao design**
- **Testes de componente usualmente estão associados ao código**

Modelo V

O modelo V define diferentes estágios de testes para serem executados

Pode ser visto como uma extensão do modelo em cascata

- Ainda existe a ênfase na especificação
- Cada fase precisa ser finalizada antes do início da fase seguinte
- Também é ideal para softwares com requisitos conhecidos

A grande vantagem do modelo é:

- Identificar os diferentes estágios de teste que validam aspectos do ciclo de vida do software
- Permite um melhor planejamento dos testes que precisam ser executados
- “Quebra” os testes em diferentes focos

A principal desvantagem é:

- Os testes ainda são rodados apenas após o código estar pronto

Modelo Incremental

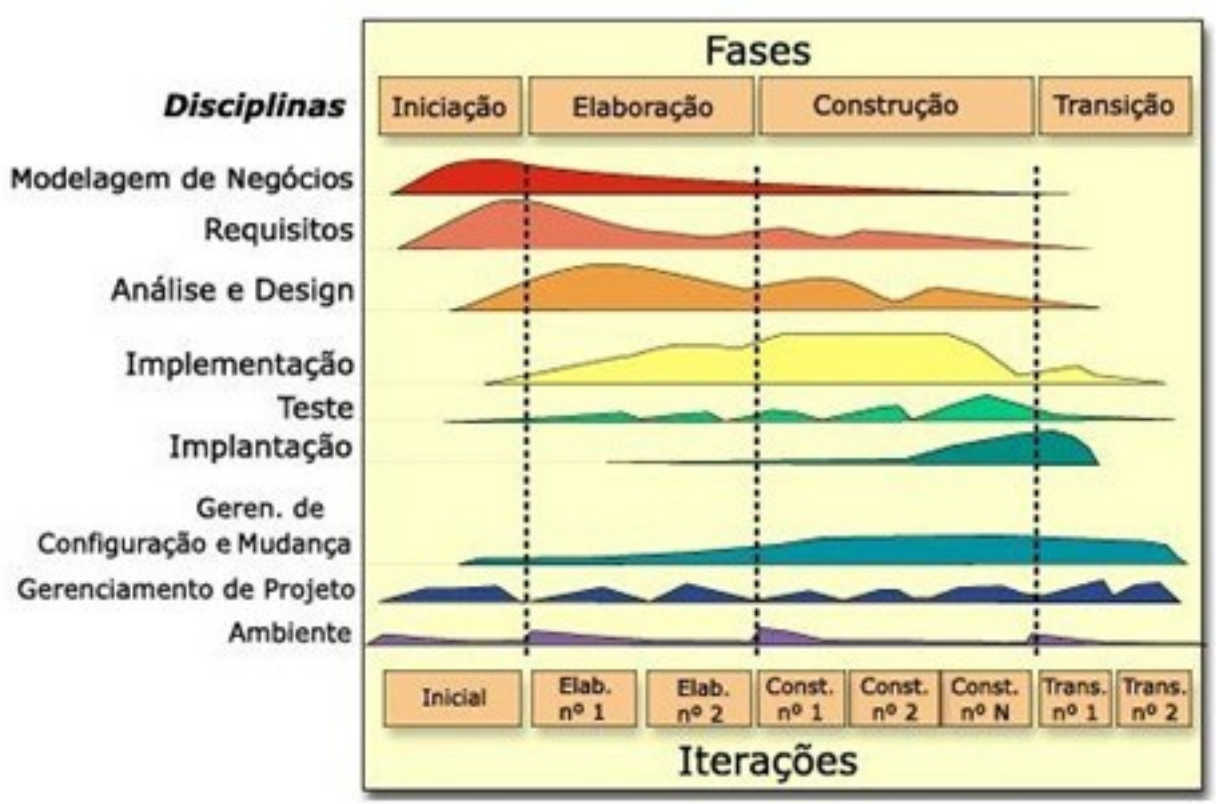
Desenvolvimento é “quebrado” em várias iterações

Cada iteração pode ser vista como uma pequena cascata

Cada iteração passa por todas as fases do ciclo

- **Em maior ou menor escala**

Requisitos são adicionados ao software em cada iteração



Modelo Incremental

O modelo incremental busca usar as vantagens do cascata

- Focar em ter uma boa especificação

Mas sendo realista

- Nem toda a especificação pode estar pronta no início do desenvolvimento

As iterações podem ser repetidas até que o software esteja estável

Ao final cada iteração é necessário

- Determinar os objetivos da próxima iteração
- Avaliar os riscos
- Avaliar os testes
- Avaliar os requisitos que já estão implementados
- Planejar a iteração seguinte

Modelo Incremental

A principal vantagem para os testes é

- **Os testes são rodados várias vezes (ao final de cada iteração)**
- **Os bugs podem ser antecipados**
- **Os testadores usualmente podem (e devem ser) envolvidos no projeto desde o início**

A principal desvantagem para os testes é

- **Algumas vezes não fica claro que testes devem ser rodados ao final de cada iteração**
- **O esforço de planejamento de testes é maior**
  - Mas esse esforço normalmente é compensado
- **É necessária uma equipe de testes mais experiente para planejar e executar testes neste modelo**

Modelo Codificar e Testar

Parte de uma especificação informal

Ênfase em codificação

- Não existe ênfase em outras fases do desenvolvimento

Como a especificação não é clara os testes também não são

- Mas os testes são executados várias vezes

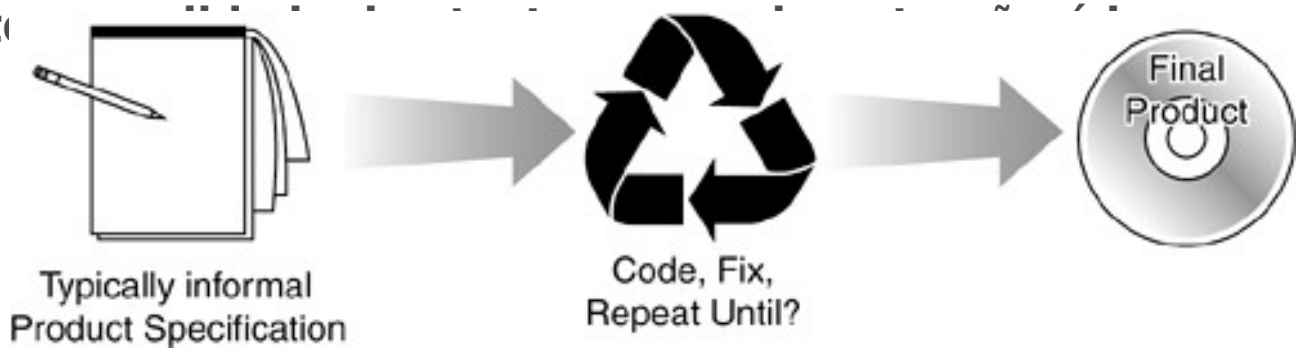
É ideal para projetos pequenos e com equipe pequena

Principal vantagem para os testes

- Vários ciclos de testes
- Custo de correção não é alto

Principal desvantagem para os testes

- Planejament



Modelos Ágeis

Muito parecido com o codificar e testar

Foco em fechar várias versões do software

- **Integração continua**

Especificação de requisitos informal

- **Mas o cliente final DEVE ficar próximo a equipe de desenvolvimento para validar os requisitos**

Mas os testes

- **São mais formais**
- **Devem ser automatizados**
  - Garantir que sempre serão rodados
- **São executados a cada integração**
- **Cliente pode participar da validação**

Principal vantagem para os testes

- **Automatização**
  - Sai a figura do testador e entra a do desenvolvedor de testes

Principal desvantagem

- **É necessário o uso de uma ferramenta para automatização**

Modelos Ágeis: TDD

Test Driven Development

- **Modelo de Desenvolvimento Orientado a Testes**

É uma das principais técnicas associadas a modelos ágeis

A idéia é que os testes sejam desenvolvidos antes do código

- **TFD: Test First Design**

O passos para usar TDD são:

- **Escreve um caso de teste e executa um caso de teste**
- **Falhando, implementa o código para passar no teste**
- **Passando, continua o desenvolvimento e escreve um novo caso de testes**

TODA a fase de implementação de um modelo ágil pode ser feita com TDD

Enquete

Seleciona as tarefas que precisam ser executadas antes que qualquer atividade de testes seja realizada (mais de uma resposta verdadeira)

a) Implementar a primeira versão da aplicação
b) Definir o processo de testes
c) A especificação inicial dos requisitos está fechada
d) A arquitetura do software está definida
e) Existe um cronograma que determina as atividades e atribuições do projeto

Conceitos em Testes de Software

Alguns conceitos são usados na literatura de testes de software

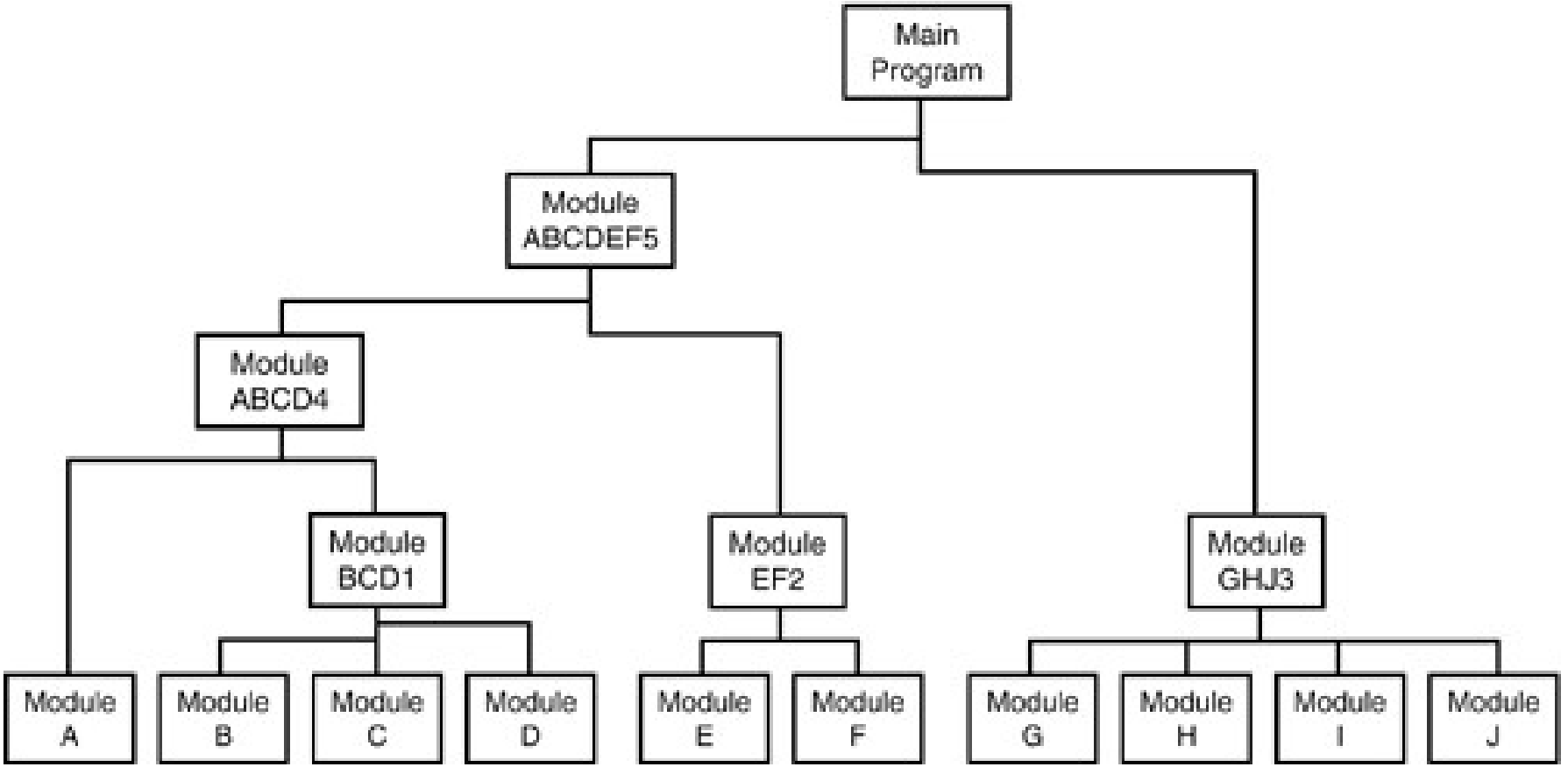
- Unidade, Componente e Integração
- Test Driver e Stub de Testes
- Processo de Testes
- Estágios de Teste
- Tipos de Teste
- Testes de Regressão
- Abordagem de Teste
- Procedimento de Teste
- Resultado de Testes
- Ferramenta de Testes
- Testes Positivos e Testes Negativos

Estes conceitos serão usados durante o restante do módulo

Conceitos em Testes de Software

- Um software em geral é quebrado em diferentes componentes (ou módulos principais)
- Cada componente é quebrado em unidades (sub-componentes)
- Estes unidades precisam ser integradas em componentes
- Os componentes precisam ser integrados no software como um todo

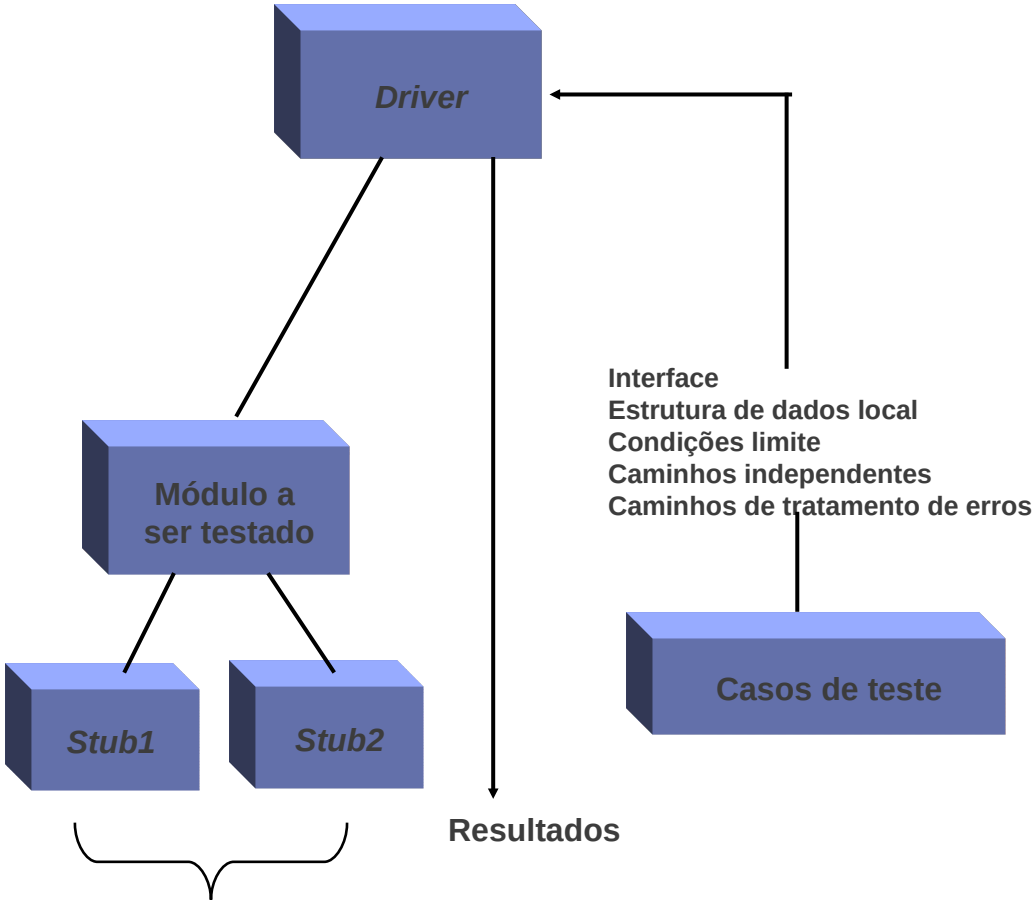
Conceitos em Testes de Software



Unidade, Componente e Integração

- Testes precisam ser executados
  - Na unidade para garantir que esta funciona separadamente
  - No componente para garantir que quando as unidades foram integradas, eles continuam funcionando
  - No sistema como um todo para garantir que os componentes integrados funcionam corretamente
- O escopo de cada um desses testes é bem diferente
  - Todos buscam encontrar bugs
  - Mas tipos diferentes de busca usualmente são encontrados
  - Um teste não substitui o outro

Unidade, Componente e Integração



Futuros módulos que serão implementados em próximas iterações

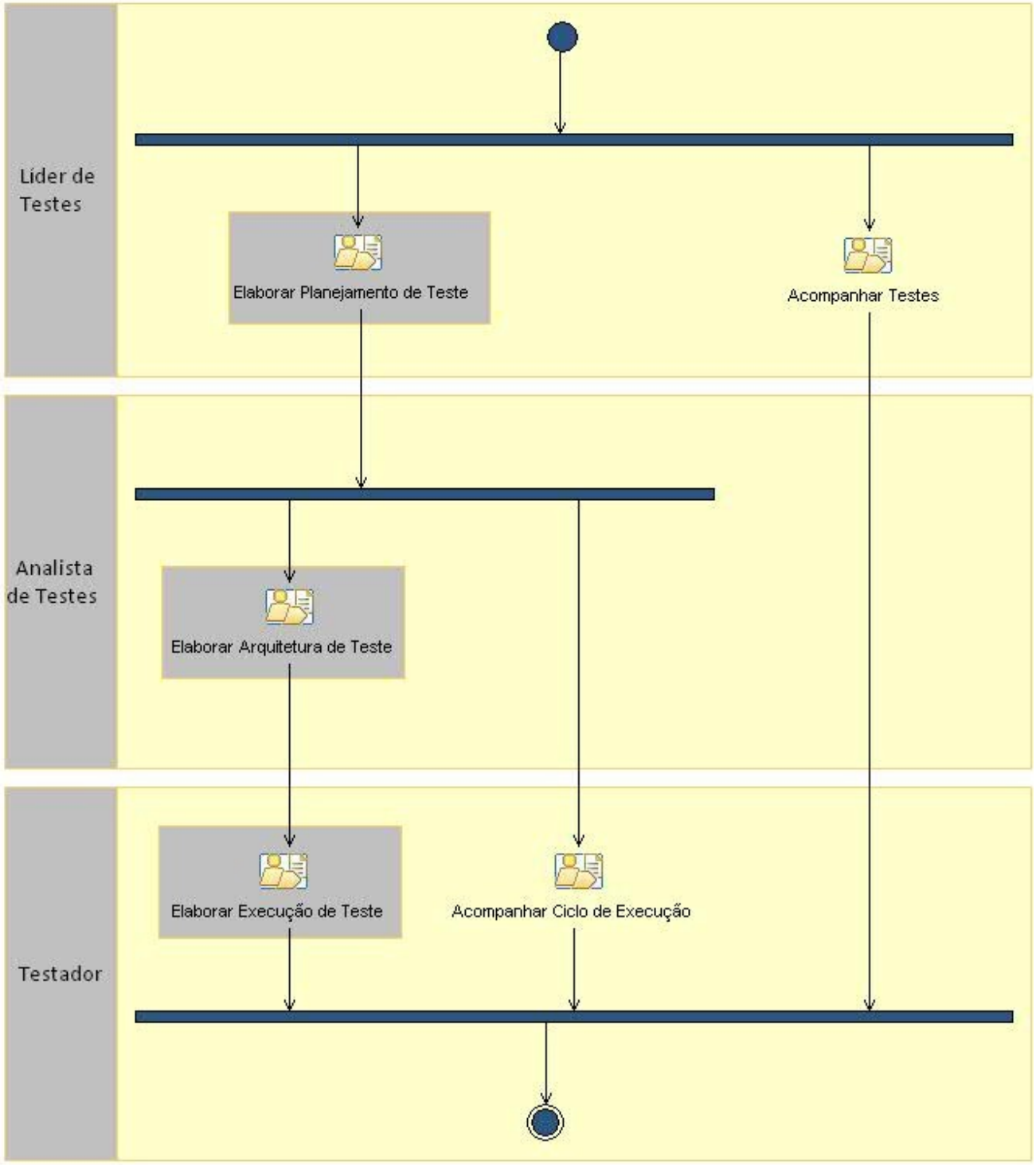
Driver **– programa principal, que aceita dados do caso de teste, passa estes dados para o módulo a ser testado e visualiza os dados relevantes.**

Stubs **– módulos que substituem (dublês) outros módulos subordinados. Utiliza a interface do módulo subordinado, manipula os dados e retorna um resultado esperado.**

Processo de Testes

- Os processos de teste são seqüências de ações, operações que são executadas com o objetivo de
  - Encontrar problemas no software
  - Encontrá-los o mais cedo possível
  - Aumentar a percepção de qualidade geral do software
  - Garantir que o usuário final tenha um software que atende as suas necessidades
- De forma geral, o processo de teste pode ser separado em 4 grandes ações
  - Planejar → entender o que precisa ser testado e definir como
  - Preparar → selecionar e montar todo o ambiente para execução
  - Executar → executar os testes e coletar o resultado
  - Avaliar → verificar os resultados e métricas para melhorar os testes

Processo de Testes



Prof. Euristenho Júnior

Estágios de Teste

- O termos “Estágio de Teste” é usualmente aplicado ao teste que está associado a uma determinada fase do ciclo de desenvolvimento
- Relembrado o Modelo V, tem-se:
  - Testes Unitários
  - Testes de Integração
  - Testes de Sistema
  - Testes de Aceitação

Estágios de Teste

- Testes Unitários
  - Executados pelo desenvolvedor para validar a unidade que está sendo implementada
- Testes de Integração
  - Executado pelo testador para garantir que vários componentes funcionam corretamente juntos
- Testes de Sistema
  - Executados pelo testador para garantir que as funcionalidades do software estão de acordo com a especificação
- Testes de Aceitação
  - Executados pelo usuário para garantir que o software faz o que foi inicialmente requisitos pelo usuário
- Cada um destes estágios será melhor detalhado posteriormente

Enquete

Marca V ou F para o que é processo de testes

V F

( ) ( ) O processo de testes determina os estágios de teste

( ) ( ) O processo determina as ações relacionadas a atividade de testes

( ) ( ) O processo de testes defini que recurso do projeto vai executar os testes

( ) ( ) O processo de testes defini quanto tempo dura a execução dos testes

Enquete

Marca V ou F para as razões que vão determinar quais os estágios de teste que serão realizados durante o desenvolvimento

V F

( ) ( ) Os estágios são determinados pelo ciclo de vida

( ) ( ) Os estágios estão associados a equipe de desenvolvimento. Caso a equipe tenha condições de executar um estágio especifico isso será feito

( ) ( ) Os estágios estão associados ao domínio da aplicação e a tecnologia do projeto

( ) ( ) Os estágios são determinados pelo processo de testes

Enquete

Marca V ou F para as afirmações relacionadas a estágios de testes

V F

( ) ( ) Testes unitários são obrigatoriamente executados por uma equipe de testes separada

( ) ( ) Testes unitários são obrigatoriamente executados pelo próprio desenvolvedor

( ) ( ) Teste de sistema são obrigatoriamente executados por uma equipe de testes separada

( ) ( ) Testes de componente sempre são automatizados

Tipos de Teste

- Frequentemente confundidos com os estágios de teste
- “Tipos de Testes” identificam os diferentes testes que podem ser realizados em cada um dos estágios
  - Cada tipo pode estar associado a um estágio específico, ou pode ser realizado a partir dos testes de componente
- Tipos também estão associados com cada domínio de aplicação
  - No desenvolvimento de jogos são realidades testes de jogabilidade (só aplicável a jogo)
  - Em aplicações que usam banco de dados são realizados testes de volume de carga no banco
- Alguns tipos de teste comuns
  - Testes de carga
  - Testes de configuração
  - Testes de compatibilidade
  - Testes de performance
  - Testes de documentação
  - Testes de internacionalização
  - Testes de usabilidade

Tipos de Teste

- Testes de carga
  - Foco em garantir um número específico de transações no banco de dados (alto volume de dados e alto número de transações)
  - É aplicável a partir dos testes de componente
- Testes de configuração
  - Verifica se o software funciona com diferentes configurações de hardware e software
  - Está associado aos requisitos não-funcionais / restrições do software
  - Usualmente é aplicável a partir dos testes de integração
- Testes de compatibilidade
  - Verifica se um determinado software está aderente a um padrão que ele deveria implementar
  - Usualmente é executado durante o teste de sistema

Tipos de Teste

- Testes de performance
  - Garantir que a velocidade de execução do sistema é adequada (tempo para carregar uma página web por exemplo)
  - Associado aos requisitos não funcionais
  - Pode ser executado a partir dos testes de integração
- Testes de documentação
  - Garantir que a documentação do software está do acordo com o que foi realmente implementado
  - Tem mais sentido ser executado durante os testes de sistema
- Testes de internacionalização
  - Garantir que o software funciona em diferentes línguas e com configurações dependentes de pais (moeda, timezone, etc.)
- Testes de usabilidade
  - Verifica se as funções do sistema realmente são “fáceis” de serem realizadas
  - Deve ser executado durante os testes de sistema

Testes de Regressão

- Considere o seguinte cenário
  - Durante a execução dos testes de sistema de um software encontra-se um erro após a execução de 70% dos testes
  - Este erro é corrigido e uma nova versão é gerada
- Que testes devem ser executados após isso?
  - Os 30% restantes
  - E também pode-se regredir e executar todos os outros testes que já foram executados
  - A execução dos testes que já foram rodados é chamado de Testes de Regressão
- Um testes de regressão completo significa que TODOS os testes serão rodados novamente

Abordagem de Teste

- Existem 2 abordagens principais na execução de testes
  - Caixa Preta
  - Caixa Branca
- Testes caixa preta são aqueles nos quais o testador não vê o “interior” do objeto que está sendo testado
  - Usualmente executados por um testador que é diferente do desenvolvedor
  - Avalia todas as diferentes entradas e suas respectivas saídas para garantir que o software se comporta como especificado
  - Associado a estágios de teste como sistema ou componente

Abordagem de Teste

- Testes caixa branca são aqueles nos quais o testador sabe como funciona o “interior” do objeto sendo testado
  - Usualmente são executados pelo próprio desenvolvedor
  - São fortemente associados aos testes unitários
  - Avalia TODOS os caminhos lógicos do código para garantir que TODOS os fluxos são executados corretamente
  - Busca encontrar bugs, mas bugs no código e não bugs na especificação, por exemplo
- A grande maioria dos software realiza apenas testes caixa preta
- Quando testes caixa branca são executados, eles não são formalizados
  - Não existe procedimento bem definido
  - Não existe resultado de testes documentado
- Caso um procedimento de testes caixa branca seja escrito, existe grande chance dele ser descartado após a execução dos testes

Procedimento de Teste

- O procedimento de testes está diretamente relacionado ao conceito de caso de teste
- Cada caso de teste está associado a um diferente “cenário” a ser testado
  - Para cada requisito existem diferentes cenários a serem testados
- Para validar um caso de testes é necessário
  - Definir o ambiente no qual o teste será realizado
  - Definir a entrada deste caso de teste
  - Definir a saída esperada para cada entrada
  - Definir os passos a serem realizados para executar os testes
- O conjunto de passos acima determina o que um procedimento de testes

Resultado de Testes

- Quando cada caso de teste é executado, o seu resultado deve ser coletado
- Existem diferentes abordagens para definir o resultado de um caso de teste especifico
- A mais comum define as seguintes opções
  - Passou → todos os passos do caso de testes foram executados com sucesso para todas as entradas
  - Falhou → nem todos os passos foram executados com sucesso para uma ou mais entradas
  - Bloqueado → o teste não pode ser executado, pois o seu ambiente não pode ser configurado, ou pode alguma dependência externa

Ferramenta de Testes

- Existem 2 tipos principais de ferramentas
  - Ferramentas de apóio a execução dos testes
  - Ferramentas apóio ao planejamento e documentação de Testes
- As ferramentas de execução focam em
  - Automatização do procedimento de testes através de scripts de testes
  - Automatização da execução
  - Automatização do resultado de testes
- As ferramentas de planejamento focam em
  - Gerenciamento dos casos de teste e dos ciclos de execução de testes
  - Atribuição de recursos para execução dos testes
  - Coleta de métricas
- Vários outros requisitos podem ser considerados em outras diferentes ferramentas

Testes Positivos e Testes Negativos

- Testes positivos verificam se o caminho feliz é executado corretamente
- Testes negativos verificam as exceções
- Testes positivos
  - Garantem que o software funciona minimamente como esperado
  - Garantem que o software não quebra para as suas operações principais
  - Não levam o software aos seus limites para identificar possíveis falhas
- Testes negativos
  - Buscam realmente quebrar o software
  - Focam em identificar problemas
  - Garantem que o software funciona em diversas situações diferentes
- Ambos os tipos de testes devem ser realizados

ENQUETE

O que é testes de regressão e quando eles são executados

a) São testes que validam se o software atinge a expectativa do usuário, eles são executados no final do projeto
b) São testes realizados pelo desenvolvedor toda vez que ele faz uma modificação no código para garantir que o código está correto
c) São testes executados quando uma primeira versão do software é fechada para verificar se o software está indo na direção correta
d) São testes executados após uma manutenção do software para garantir que as funcionalidades principais ainda funcionam

ENQUETE

Marca V ou F para as afirmações relacionadas a tipos de testes

V F

( ) ( ) Testes de carga só são aplicados para sistema de banco de dados

( ) ( ) Não existe um número de tipos de testes fixo e as suas definições também podem variar

( ) ( ) Teste de usabilidade são obrigatoriamente executados por uma equipe de testes separada

( ) ( ) Testes de performance sempre são automatizados

ENQUETE

Marca V ou F para as afirmações relacionadas a abordagem de testes

V F

( ) ( ) Testes caixa preta são sempre realizados pelo desenvolvedor

( ) ( ) Testes caixa preta só podem ser testes de sistema

( ) ( ) Testes unitários são sempre caixa branca

( ) ( ) Testes caixa preta são mais adequados para encontrar problemas de especificação que testes caixa branca

Tipos de Testes

Anteriormente foi listado brevemente alguns e diferentes tipos de testes

Estágios

- Testes Unitários / Integração
- Testes de Sistema
- Testes de Aceitação

Tipos de Teste

- Testes de Carga
- Testes de Performance
- Testes de Uso de Memória
- Testes de Usabilidade
- Teste de Configuração
- Testes de Compatibilidade
- Testes de Documentação

Estágios – Testes Unitários

- Testes unitários são testes isolados executados sobre uma rotina, classe ou pequena arte do programa
  - Esses testes são executados pelo próprio desenvolvedor para garantir que a sua unidade pode ser integrada com o resto do software
- Dois aspectos fundamentais
  - Os testes são isolados
  - Os testes são implementados e executados pelo próprio desenvolvedor
- Testes isolados
  - Stubs devem ser implementados para suprir alguma parte da funcionalidade ou gerar valores errados para validar a unidade
  - Os componentes externos a unidade em testes devem ser simuladas
- Testes executados pelo desenvolvedor
  - Os testes são normalmente rodados no ambiente de desenvolvimento
  - Os casos de testes focam apenas na unidade em testes sem se preocupar com a interação com outras unidades

Estágios – Testes Integração

Unidades ou aplicações que foram testadas em separado são testadas de forma integrada

A interface entre as unidades integradas é testada

O teste de integração deve ser feito de forma incremental, ou seja, as unidades devem ser integradas em pequenos segmentos

Este teste é executado por um testador de integração (geralmente o desenvolvedor)

Teste de Integração

- A integração dos módulos pode ser feitas através das abordagens *top-down* ou *bottom-up*:

- *Top-down* – Os módulos são integrados de cima para baixo. O teste usa *driver* e *stubs.*

  - O *driver* é utilizado como módulo de controle principal, e os módulos reais são substituídos por *stubs*.
  - À medida que os testes vão sendo realizados os *stubs* são substituídos pelos módulos reais, um de cada vez.

Teste de Integração

- *Bottom-up* – a integração é feita a partir do nível mais básico da hierarquia. Os *stubs* nem sempre são necessários.
  - Os módulos do nível inferior são combinados.
  - Para cada combinação é criado um *driver* que coordena a entrada e a saída dos casos de teste.
  - O módulo é testado.
  - O *driver* é substituído pela combinação de módulos correspondentes, que passam a interagir com os módulos do nível superior.

Teste de Integração

- *Top-down*
  - Vantagem: os testes das funções principais são realizados no início.
  - Desvantagem: criação de *stubs* complexos.
- Bottom-up
  - Vantagem: podem não precisar de *stubs*.
  - Desvantagem: o módulo principal só é testado quando todos os módulos dos níveis inferiores estiverem sido testados.

Estágios – Testes Sistema

- **Verifica se a aplicação está funcionando como um todo**
- **A integração dos componentes de software com o ambiente operacional similar ao de produção (hardware, software, pessoas e outros sistemas) é testada**
- **Geralmente é um teste “caixa-preta”, executado por um testador de sistemas (idealmente membro de um grupo independente de testes)**

Estágios – Testes de Aceitação

- São os testes realizados pelo usuário que contratou o desenvolvimento do sistema
- Valida que a necessidade inicial do cliente quando o desenvolvimento do software foi contratada foi atendida
  - Foco não é na especificação, mas na necessidade do cliente
- Por mais que especificações sejam escritas e aprovadas pelo cliente
  - É comum que alguma necessidade não seja totalmente atendida
  - Algumas vezes a necessidade original foi implementada de outra forma
- Outros problemas como
  - Usabilidade, performance ou estabilidade do software também podem ser identificados
- A organização que desenvolveu o software não tem muito controle sobre a aceitação
  - Um processo de aceitação pode ser definido, mas o cliente também pode ter seu processo interno

Tipos – Testes de Carga

- Associado principalmente a softwares que usam bancos de dados
- É um tipo de testes
  - Caixa preta
  - Principal objetivo é verificar como o software se comporta com diferentes cargas de banco de dados
  - Precisa de diferentes scripts para popular o banco
  - Está associado aos requisitos, mas também ao modelo de dados do software
- Algumas vezes a implementação assume alguns comportamentos do modelo de dados
  - Dados dependentes da implementação do código e vice-versa
  - Isso pode causar bugs na aplicação, pois os dados do banco deveria estar associados e serem dependentes apenas do modelo de dados
- Esses testes podem ser automatizados, mas o ideal é que sejam manuais para identificar os comportamentos diferentes do software

Tipos – Testes de Performance

- Verificar a performance da aplicação em diferentes cenários
- Esta diretamente associado aos requisitos não funcionais do software
  - Os requisitos devem especificar a performance esperada do software
  - Caso isso não seja feito pode gerar frustração durante a aceitação do software
- A performance também está associada ao número de usuário utilizando o software
  - Em sistemas Web, a performance tem que ser garantida até um número X de usuários
- Existem ferramentas que ajudam a implementar scripts para validar a performance
  - Simular vários usuários conectados a aplicação
- Estes testes também estão associados ao ambiente de deploy do software
  - Este ambiente influencia diretamente no resultado dos testes

Tipos – Testes de Performance: Exemplo

- Considerando o caso de uso listar contas de pessoas.
- O software pode ter um requisito não funcional
  - Mostrar as contas de um pessoa específica em no máximo 5 segundos
  - Com uma base de até 10mil pessoas e 100mil contas
  - Com até 100 usuários logados simultaneamente
- Os seguintes casos de teste podem ser identificados:
  - Montar uma base com este número de usuários e buscar listar as contas destes usuários repetidamente. Calcular a média de tempo para mostrar as contas

Tipos – Testes de Usabilidade

- Que atributos definem uma boa interface com o usuário?
  - Grande investimento é feito pelas empresas para responder essa pergunta
- Com os atributos definidos, a UI do software pode ser definida para atingi-los
- Algumas características de um boa UI
  - Segue um padrão determinado
  - Flexível
  - Intuitiva
  - Consistente
  - Confortável
  - Útil

Tipos – Testes de Configuração

- Atualmente existem diferentes plataformas de hardware e software nas quais um determinado software pode executar
- Além disso, o software desenvolvido também possui diferentes configurações internas
- Resumindo, os testes de configuração buscam
  - Identificar o comportamento do software em diferentes configurações de hardware
  - Identificar o comportamento do software em diferentes configurações de software
  - Identificar o comportamento do software em diferentes configurações internas do software
- As possíveis plataformas de hardware e software nas quais o software deve funcionar
  - DEVEM ser mapeadas durante a especificação do software
  - DEVEM ser aprovadas pelo cliente
- Partindo destas especificações os casos de teste precisam ser levantados

Tipos – Testes de Configuração

- O software pode ter como restrição utilizar banco de dados Oracle ou Postgresql
- O caso de teste então deve
  - Configurar 2 ambientes de testes: 1 com cada banco de dados
  - Executar as funcionalidades básicas do software em ambos os ambientes
    - Caso alguma não funciona corretamente, o teste é falho
  - Em algumas situações TODOS os testes precisarão ser executados em ambos os ambientes

Tipos – Testes de Compatibilidade

- É aplicável para software que
  - Implementam algum padrão específico e que precisam estar compatíveis com este
  - Interagem com alguma aplicação que em a sua interface padronizada
- No primeiro caso, o software precisa ser valido quando ao padrão
  - Usualmente os organismos de padronização definem conjuntos de testes que validam as implementações
  - A execução destes testes precisam então ser planejadas para validar a implementação do software
- No segundo caso, o software precisa
  - Garantir que está usando a interface padronizada corretamente (por exemplo, usar um porta USB)
  - Testes precisam ser especificados para as diferentes opções definidas no padrão

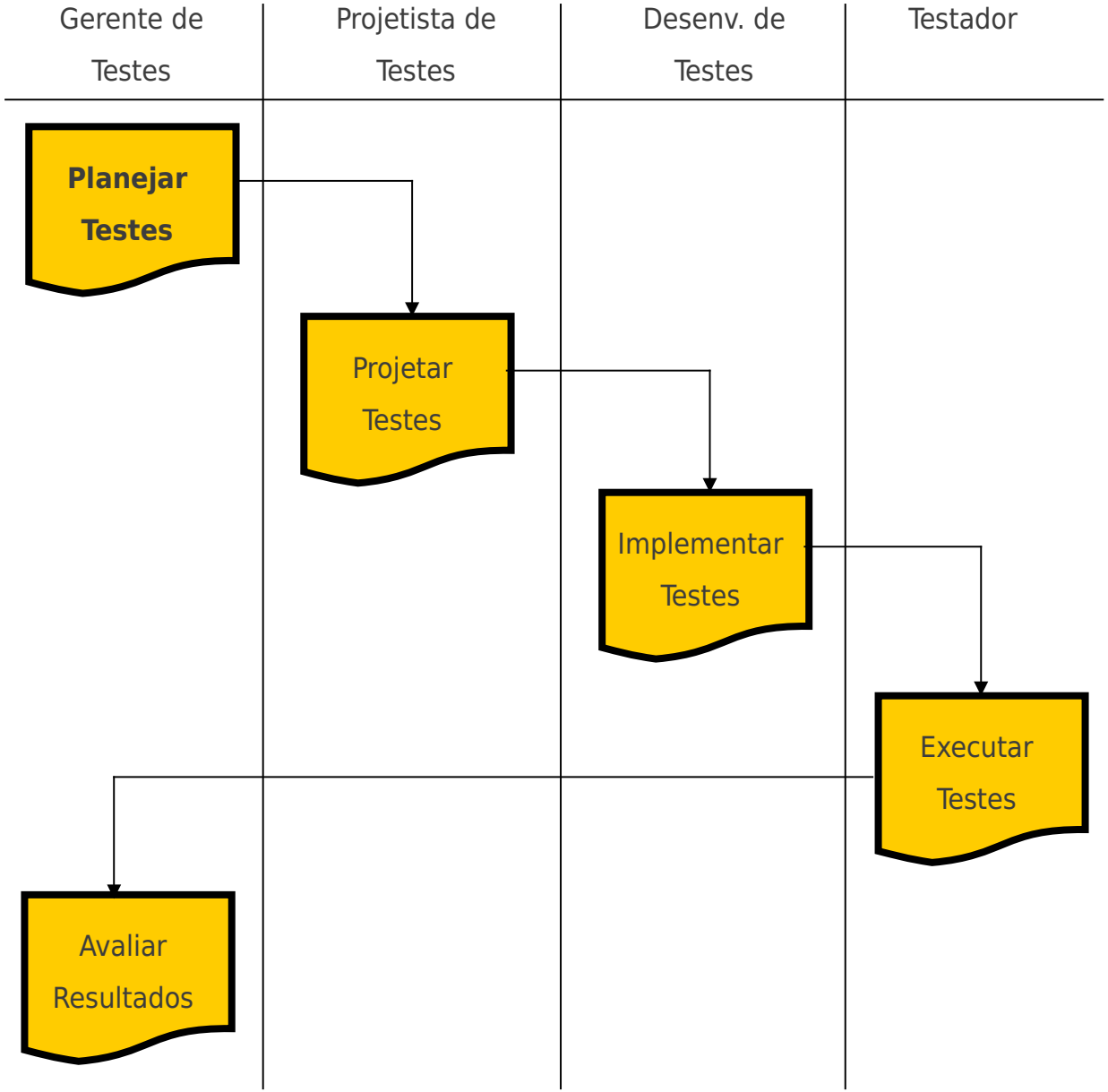
Processo de Testes

- O primeiro passo para realizar as atividades relacionadas a testes em uma organização é definir um processo de testes
- Este processo deve:
  - Definir as principais atividades que precisam ser seguidas durante os testes
  - Definir papéis e responsabilidades das atividades de teste
  - Definir as ações associadas a cada atividade
  - Definir as entradas e saída de cada um das atividades
  - Associar possíveis templates de documento a cada uma das atividades
- As principais atividades que podem ser identificadas em um processo de testes são:
  - Planejar Testes
  - Projetar Testes
  - Implementar Testes Automáticos
  - Executar Testes
  - Avaliar Resultados

Principais papeis em Testes

- Pode-se identificar 4 papéis principais para as atividades de testes
  - Gerente de Testes → Monta o plano de testes e faz o acompanhamento de todas as atividades relativas aos testes
  - Projetista de Testes → Identifica os casos de teste e escreve os procedimentos para cada um deles
  - Desenvolvedor de Testes → Implementa os casos de teste automático que foram projetados pelo projetista
  - Testador → Executa os casos de teste e preenche o resultado de testes
- Estes papéis podem ou não ser realizados pela mesma pessoa
  - Isso depende de cada organização ou do tamanho / escopo do projeto
- Existem outros termos que podem ser usados para descrever os papéis em atividades de teste
  - Durante o restante do módulo serão usados os termos definidos acima

Processo de Testes



Projetar Testes

- Objetivo da Atividade
  - Planejar as atividades de testes que serão realizadas durante o ciclo de vida do software
- Responsável pela atividades
  - Gerente de Testes com apóio do gerente de projetos
- Principais Passo
  - Definir estágios de testes que serão executados
  - Definir tipos de teste que serão necessários em cada estágio
  - Entender os requisitos que precisarão ser testados
  - Definir se algum teste será ou não automatizado
  - Estimar o esforço necessário para projetar e implementar os testes
  - Estimar o esforço necessário para executar os testes
  - Definir que recursos de hardware e software serão necessário para a execução dos testes (definir ambiente de testes)
  - Definir riscos, dependências e premissas dos testes
  - Montar cronograma das atividades de testes
  - Associar este cronograma com o cronograma principal do projeto
- Artefato Gerado
  - Plano de Testes

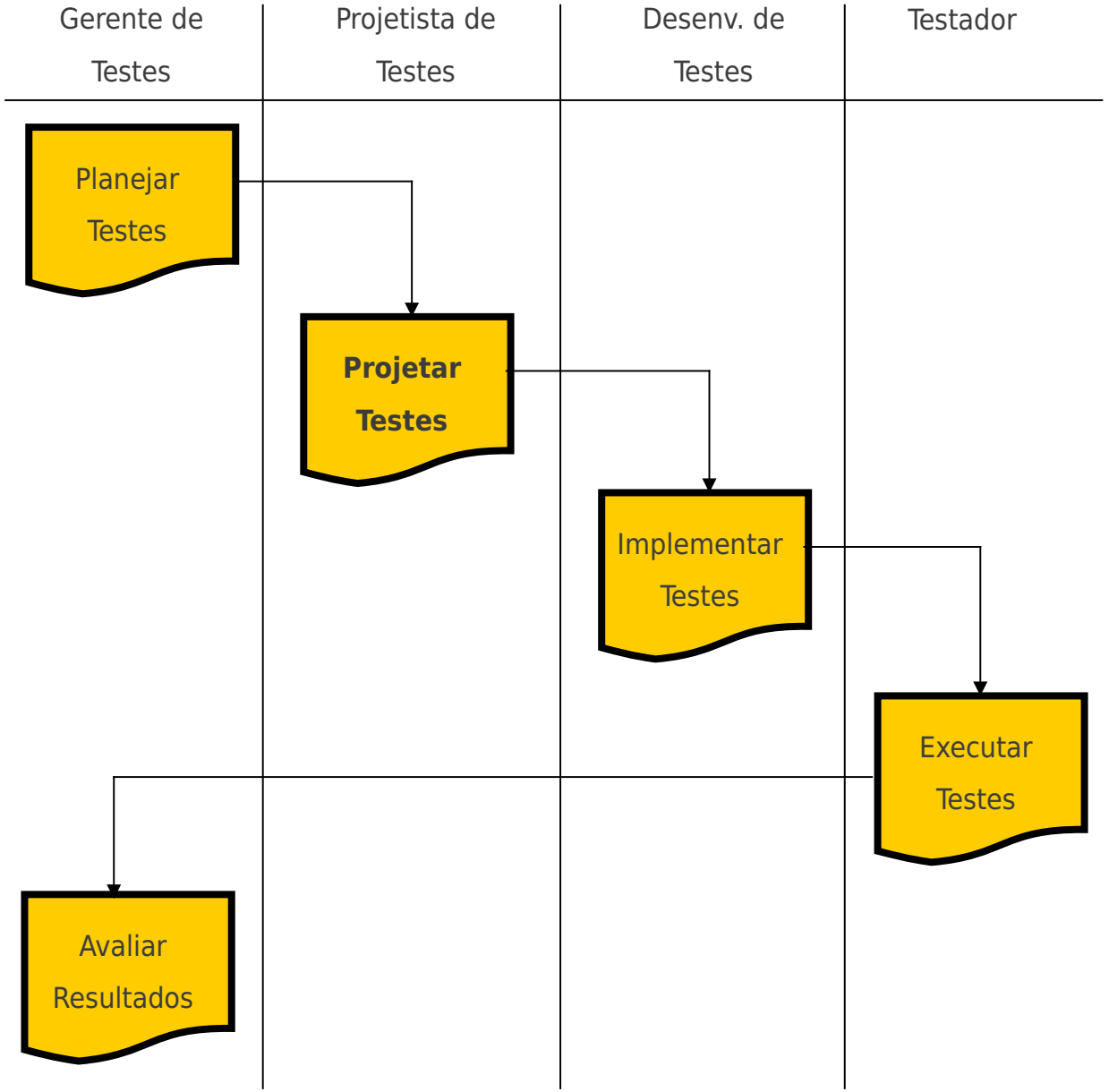
Planejar Testes

- É importante notar que as atividades podem mudar de acordo com o tipo de projeto
- O planejamento de testes DEVE ser executado
  - Após ou em paralelo com as atividades de requisitos
- O gerente de testes deveria ser alocado junto com o gerente de projetos
  - Assim como o projetista de testes DEVE ser alocado no projeto junto com o analista de requisitos
- Determinar que estágios e tipos de teste serão realizados é uma das principais atividades
- A atividade mais difícil de executar é a estimativa
- Durante o módulo seguinte a atividade de planejamento de testes será mais detalhada

Planejar Testes: Possível Estrutura de um Plano de Testes

- Introdução
- Estágios de Testes
  - Critérios de Entrada
  - Critérios de Saída
- Tipos de Testes
  - Critérios de Entrada
  - Critérios de Saída
- Ambiente de Testes
  - Recursos de Hardware
  - Recursos de Software
- Riscos, Dependências e Premissas
- Cronograma

Processo de Testes



Projetar Testes de Sistema

- Objetivo
  - Identificar os casos de teste e escrever o procedimento de testes para cada um deles
  - Definir qual dos casos de teste vai ser automatizado
- Responsável pela atividade
  - Projetista de Testes
- Principais Passos
  - Avaliar os requisitos funcionais e não funcionais
  - Verificar quais os tipos de testes precisarão ser executados
    - De acordo com isso outros passos precisarão ser realizados
  - Para cada requisito definir os cenários de testes associados a eles
  - Para cada cenário definir um caso de teste
  - Para cada caso de testes definir
    - Entradas e saídas
    - Pré e pós condições
    - Passos para execução do caso de teste
- Artefato de Saída
  - Projeto de Testes

Projetar Testes de Sistema: Avaliar Especificação

- Primeiro passo para projetar os testes de sistema é avaliar a especificação do software
- Essa avaliação pode ser feita
  - Durante uma revisão formal da especificação
  - Após os requisitos serem fechados e antes de se projetar os testes
- O ideal é que o projetista de teste participe da revisão dos testes
  - Requisitos NÃO devem ser fechados sem a avaliação pela equipe de testes
- Cada requisito funcional precisa ser verificado para garantir
  - Completude
  - Corretude
  - Precisão
  - Consistência
  - Relevância
  - Viabilidade
  - Testabilidade

Projetar Testes de Sistema: Avaliar Especificação

- Completude
  - Existe algo faltando no requisito? Ele contém todas as informações necessários para o seu entendimento
- Corretude
  - O requisito apresentado realmente resolve o problema que ele se propõe?
- Precisão
  - A descrição do requisito está clara? Existe alguma parte que pode interpretada erradamente?
- Consistência
  - Os requisitos são consistentes entre si? Alguma requisito é contraditório com outro?
- Relevância
  - O requisito realmente é um requisito? Ou é algum aspecto de design que deveria ser tratado posteriormente?

Projetar Testes de Sistema: Avaliar Especificação

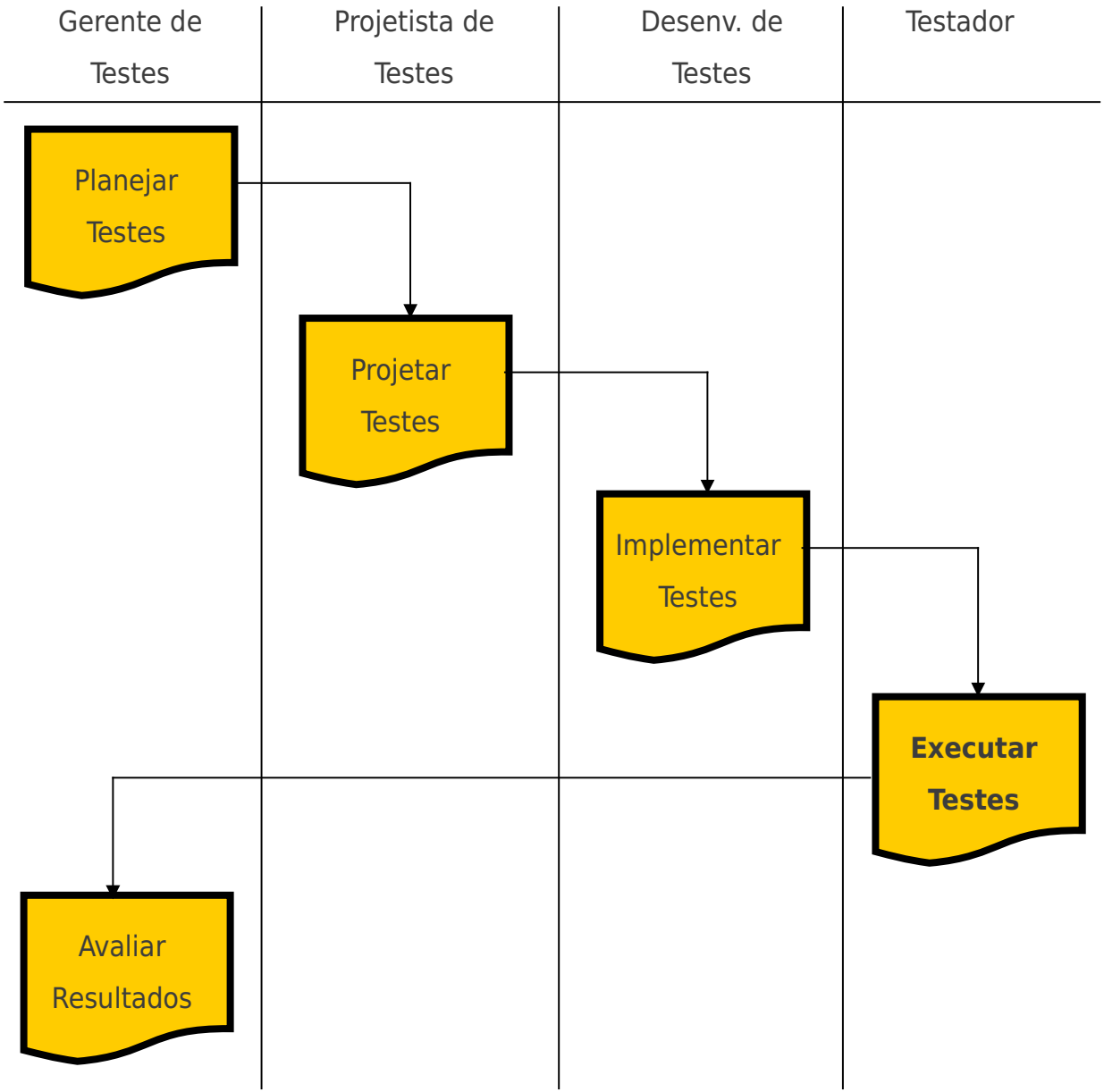
- Viabilidade
  - É possível implementar esse requisito ? A tecnologia atual permite que ele seja implementado?
- Testabilidade
  - O requisito pode ser testado ? É possível gerar um procedimento de testes que valide se o requisito está implementado corretamente?

- Os requisitos não funcionais NÃO devem ser esquecidos
- É necessário garantir que estes estão claramente especificados
  - Qual o requisito de segurança?
  - Existe requisito de performance? Qual é?
  - Existe alguma restrição de ambiente? Qual o banco de dados e o servidor de aplicação que deve usado?
  - Existem alguma restrição de tecnologia?

Projetar Testes de Sistema

- Os casos de teste de sistema são caixa preta e são executados sobre todo o sistema
- A avaliação dos requisitos é a atividade principal
- O ideal é que o projetista de testes participe da revisão dos requisitos para garantir que
  - Cada requisito está claramente especificado
  - Cada requisito é testável
  - Os cenários de cada requisito são claramente definidos
  - Os requisitos funcionais estão corretamente especificados
- Sem as informações acima não é possível projetar os casos de teste corretamente
  - A baixa qualidade dos testes está diretamente associada a baixa qualidade dos requisitos
- A atividade de projetar testes será melhor detalhada no módulo seguinte

Processo de Testes



Executar Testes

- Objetivo
  - Executar os testes planejados e coletar os resultados
- Responsável pela atividade
  - Testador
- Principais passos
  - Os testes a executar devem ser definidos pelo projetista / gerente de testes
  - Para testes manuais, o testador deve seguir o procedimento de testes aplicando todas as entradas e verificando as saídas
  - Para testes automáticos os testadores devem configurar a ferramenta de testes e colocar os scripts de teste para rodar
  - Os resultados de testes devem ser coletados
- Artefato de Saída
  - Resultado de testes

Executar Testes

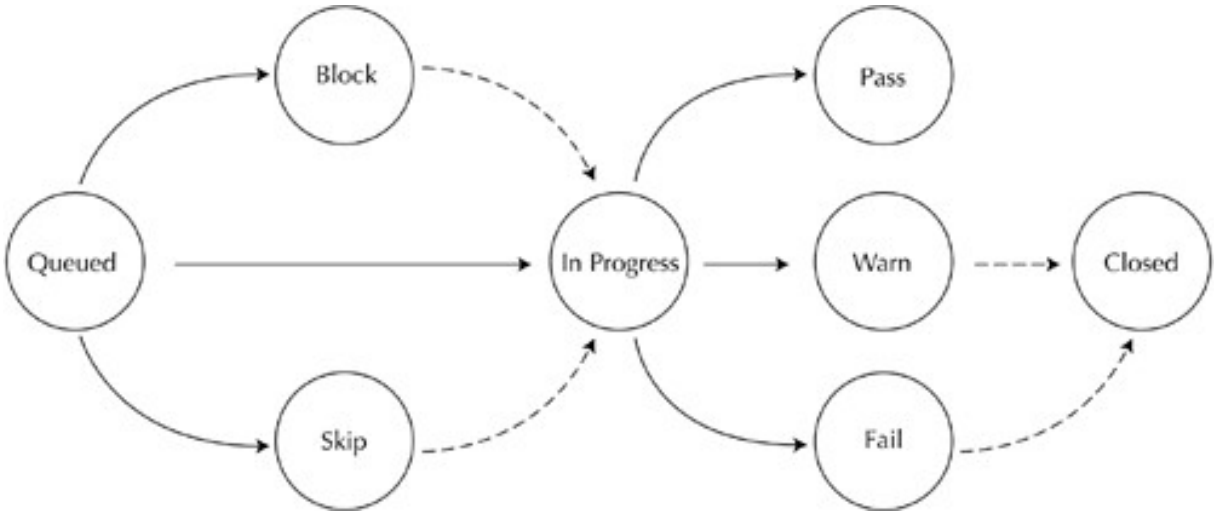
- O setup para a execução de testes automáticos pode ser bem demorado
  - Essa atividade NÃO deve ser sub-estimada
- Os testes manuais devem ser realizados com muito cuidado pelo testador
  - Alguns testes podem ser exploratórios
  - O testador deve se prender ao procedimento
  - Mas caso algum problema fora do procedimento seja encontrado, ele tem obrigação de reportar o problema
  - O testador também deve validar se o procedimento de testes está correto
    - Problemas no procedimento também devem ser reportados
  - Na dúvida, o testador DEVE reportar como erro
    - É melhor ter um falso positivo que deixar um bug passar
- O testador deve na medida do possível dominar os requisitos que estão sendo testados

Executar Testes: passo-a-passo para executar testes

- Selecionar que casos de testes serão executados de acordo com o planejamento
- Atribuir os casos de testes para os testadores
- Executar o caso de teste e reportar os bugs. Capturar informações continuamente levando em conta as execuções anteriores
  - Fazer o setup do ambiente de testes de acordo com o procedimento
  - Entrar os dados especificados e esperar pelas saídas determinadas
  - Avaliar os resultados, comportamentos e estado final do software. Pesquisar qualquer diferença com o resultado esperado
  - Caso ocorra um problema no software, reportar
  - Caso um problema seja encontrado no caso de teste, reportar
  - Capturar as informações sobre o teste que foi executado
- Resolver possíveis issues que estejam bloqueando os testes
- Reportar os resultados periodicamente

Executar Testes: passo-a-passo para executar testes

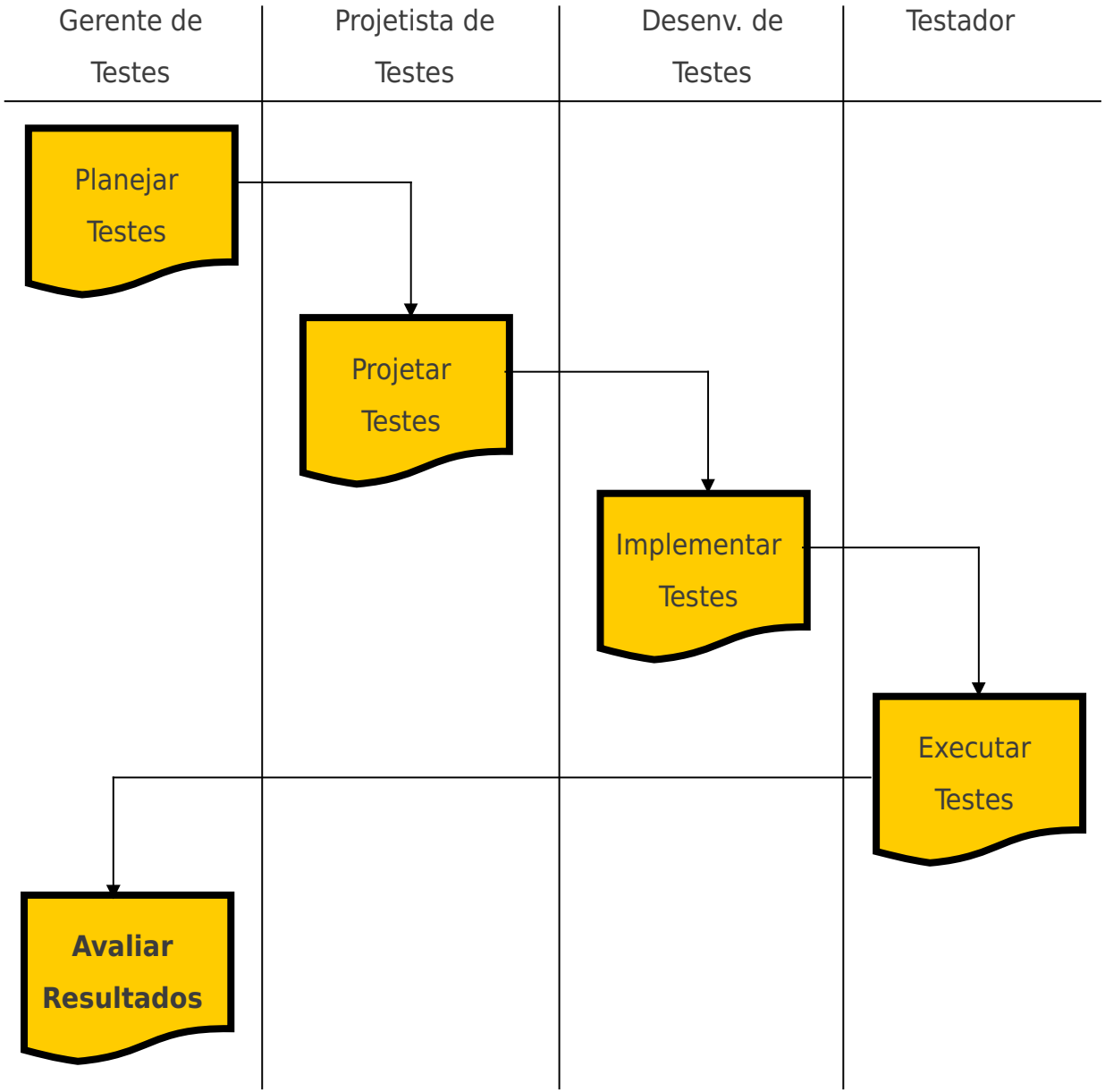
- Durante a execução de um caso de teste, este pode está em diferentes estados
  - Em alguns software é importante identificar estes possíveis estados
  - Os estados algumas vezes estão associados aos resultados dos testes
- Alguns possíveis estados genéricos
  - Queued
  - Block
  - Skip
  - In Progress
  - Pass
  - Warn
  - Fail
  - Closed



Executar Testes: Possível Estrutura do Resultado de Teste

- Caso de Teste N
  - ID do caso de Testes
  - Status: Passado, Falho, Bloqueado, etc.
  - Comentário
  - Número de Bug
  - Tempo de Execução

Processo de Teste



Avaliar Resultados

- Objetivo
  - Consolidar os dados gerados pela execução dos testes e avaliar os resultados para poder tomar decisões em relação ao projeto
- Responsável pela atividade
  - Gerente de Testes
- Principais passos
  - Coletar toda a informação sobre os resultados dos diferentes estágios
  - Avaliar se todos os testes que estavam planejados foram realmente executados
  - Caso algum problema tenha ocorrido durante a execução, isso deve ser avaliado
  - Reportar os resultados para o restante do projeto do software
  - Avaliar se os critérios esperados de resultados de testes foram atingidos. Caso não tenha sido, buscar identificar se ocorreu algum problema durante os testes
- Artefato de Saída
  - Resultado dos testes

Avaliar Resultados

- A avaliação dos resultados é fundamental pois é a conclusão de cada uma das execuções de teste
- Ela é a base para tomadas de decisões gerenciais de todo o projeto
  - Caso os testes estejam falhando mais que o esperado, alguma ação precisará ser tomada
  - Caso os testadores estejam com problema para executar os testes alguma ação
  - Caso o procedimento de testes esteja com muitos problema, ele provavelmente deverá ser revisto

Avaliar Resultados: Informações Importantes de consolidar

Várias informações podem ser consolidadas na avaliação da execução dos testes:

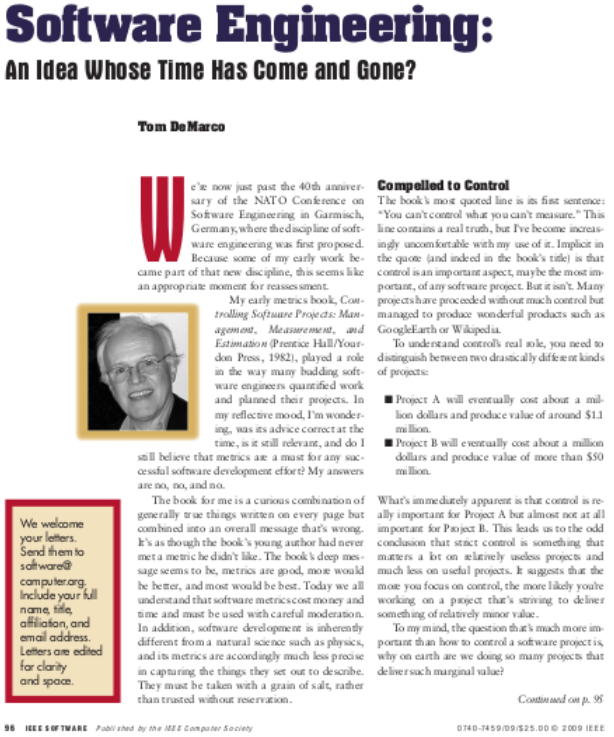
- Quantidade de Testes Planejados
  - Número total de casos de testes que foram planejados para execução
- Quantidade de Testes Executados
  - Número total de casos de testes que foram realmente executados
- Quantidade de Testes Passados
  - Número total de casos de testes que passaram
- Quantidade de Testes Falhos
  - Número total de casos de testes que falharam
- Quantidade de Testes Bloqueados
  - Número total de casos de testes bloqueados
- Quantidade de Bugs Encontrados
  - Número total de bugs encontrados durante os testes
- Tempo total de execução dos testes
  - Tempo total de execução dos testes

Avaliar Resultados: Informações Importantes de consolidar

- As informações listadas anteriormente podem ser combinadas em diferentes métricas
- Estas métricas são usadas para
  - Acompanhar o processo de testes
  - Avaliar a eficácia dos testes
  - Avaliar a eficiência dos testes
  - Avaliar o custo dos testes
- O módulo seguinte vai detalhar as possíveis métricas em testes de software

Exercícios

X



Software Engineering:
An Idea Whose Time Has Come and Gone?

Tom DeMarco

We're now just past the 40th anniversary of the NATO Conference on Software Engineering in Garmisch, Germany, where the discipline of software engineering was first proposed. Because some of my early work became part of that new discipline, this seems like an appropriate moment for reassessment.

My early metrics book, *Controlling Software Projects: Management, Measurement, and Estimation* (Prentice Hall/Yourdon Press, 1982), played a role in the way many budding software engineers quantified work and planned their projects. In my reflective mood, I'm wondering, was its advice correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort? My answers are no, no, and no.

The book for me is a curious combination of generally true things written on every page but combined into an overall message that's wrong. It's as though the book's young author had never met a metric he didn't like. The book's deep message seems to be, metrics are good, more would be better, and most would be best. Today we all understand that software metrics cost money and time and must be used with careful moderation. In addition, software development is inherently different from a natural science such as physics, and its metrics are accordingly much less precise in capturing the things they set out to describe. They must be taken with a grain of salt, rather than trusted without reservation.

Compelled to Control

The book's most quoted line is its first sentence: "You can't control what you can't measure." This line contains a real truth, but I've become increasingly uncomfortable with my use of it. Implicit in the quote (and indeed in the book's title) is that control is an important aspect, maybe the most important, of any software project. But it isn't. Many projects have proceeded without much control but managed to produce wonderful products such as GoogleEarth or Wikipedia.

To understand control's real role, you need to distinguish between two drastically different kinds of projects:

- Project A will eventually cost about a million dollars and produce value of around $1.1 million.
- Project B will eventually cost about a million dollars and produce value of more than $50 million.

What's immediately apparent is that control is really important for Project A but almost not at all important for Project B. This leads us to the odd conclusion that strict control is something that matters a lot on relatively useless projects and much less on useful projects. It suggests that the more you focus on control, the more likely you're working on a project that's striving to deliver something of relatively minor value.

To my mind, the question that's much more important than how to control a software project is, why on earth are we doing so many projects that deliver such marginal value?

Continued on p. 95

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.

0740-7459/09/$25.00 © 2009 IEEE

Tom Demarco

1- Contradição de Tom Demarco

Apresentar na próxima aula...

Valendo !!!

(1ª -Insígnia da Invisibilidade)