Engenharia de Software II / Qualidade e Teste de Software

Aula 02: Motivação e Conceitos Básicos de Teste de Software

Breno Lisi Romano

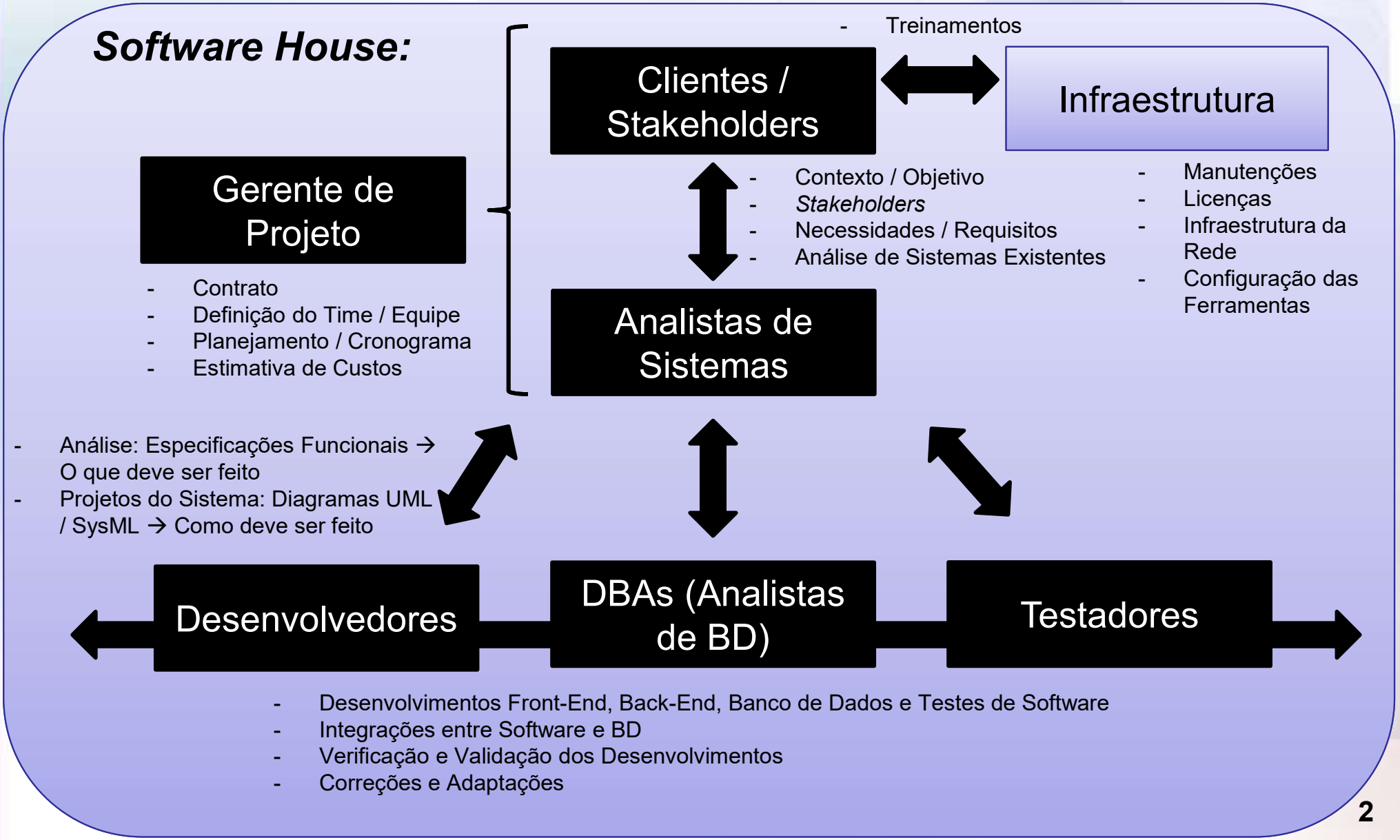
http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)
Tecnologia em Sistemas para Internet – TSI (QTSI6)

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Revisão de Engenharia de Software: Principais Papéis de uma Equipe de Desenvolvimento de Software

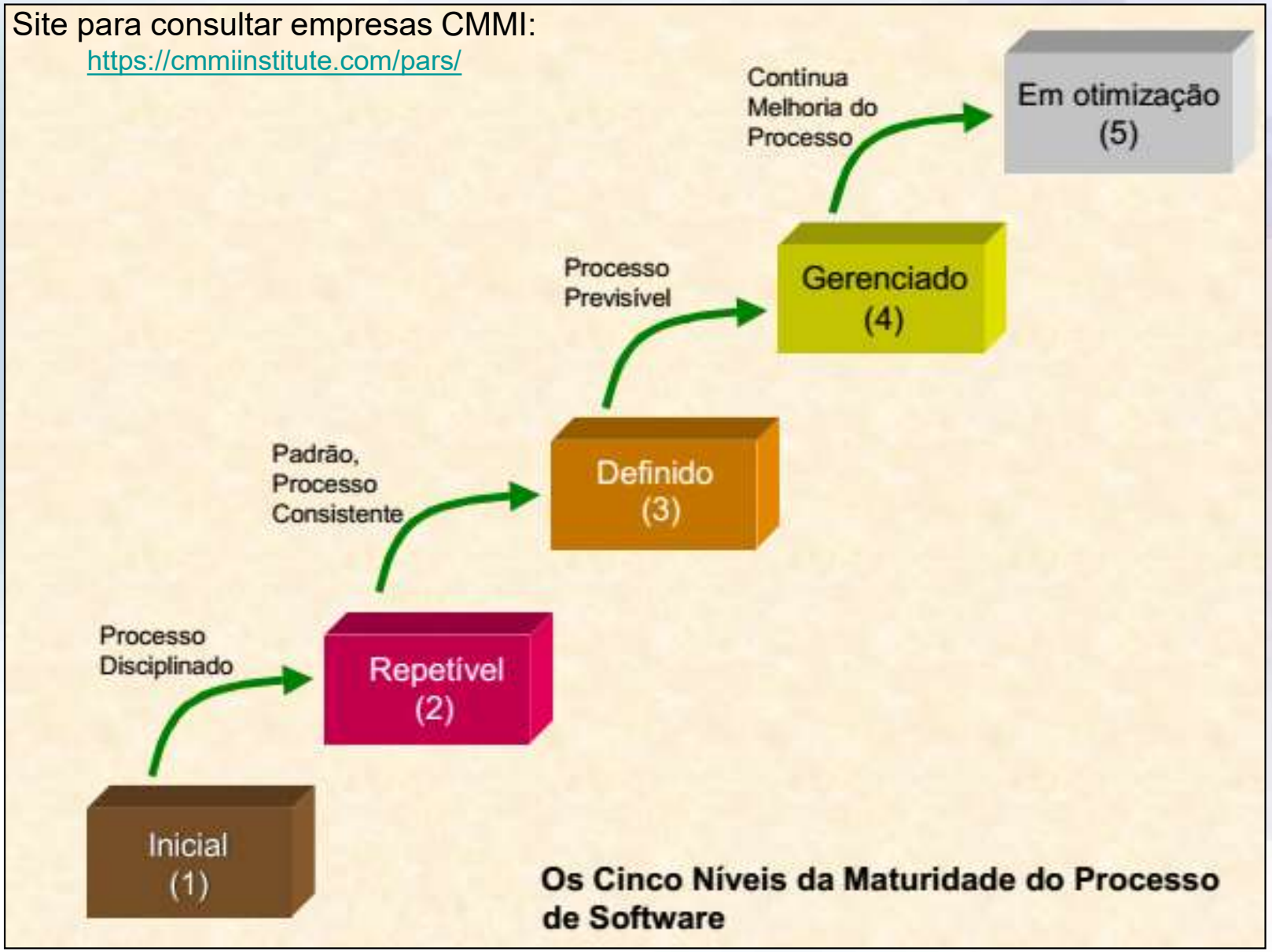


Revisão de Engenharia de Software: *Capability Maturity Model* – CMM

- **Definição:**
  - CMM **fornece** às **organizações** de **software** um **guia** de como **obter controle em seus processos para desenvolver e manter software** e **como evoluir em direção a uma cultura de Engenharia de Software e excelência de gestão**
  - Objetivo de **auxiliar** na **seleção** das **estratégias** de **melhoria**, determinando a **maturidade atual** do processo e **identificando** as **questões** mais **críticas** para a **qualidade e melhoria do processo** de software
- **CMM não é um processo de desenvolvimento de software**

Revisão de Engenharia de Software: Níveis de Maturidade do CMM

Site para consultar empresas CMMI:
https://cmmiinstitute.com/pars/

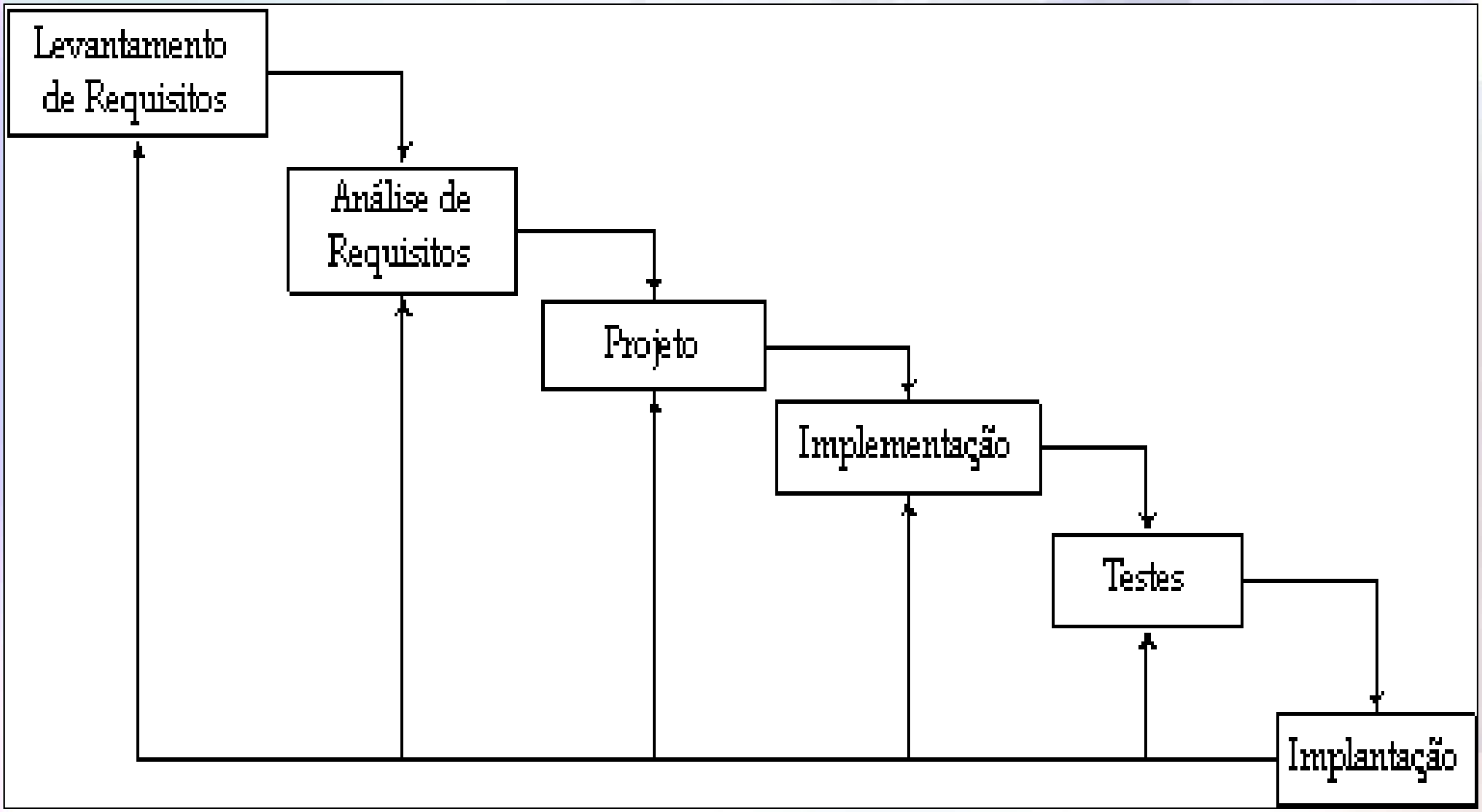


Os Cinco Níveis da Maturidade do Processo de Software

Revisão de Engenharia de Software: Tipos de Processos de Desenvolvimento de Software

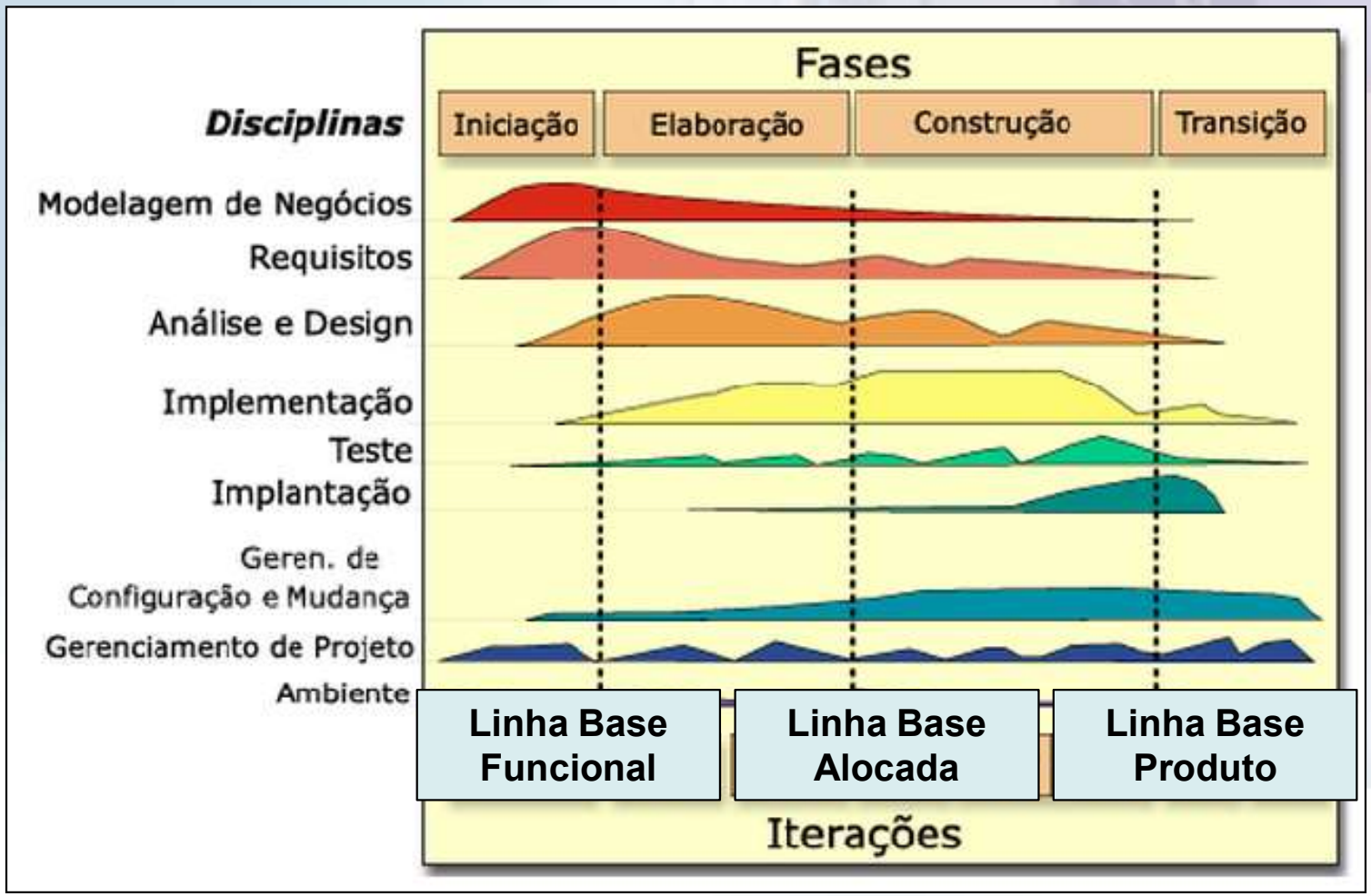
- Existem **duas grandes** Escolas:
  - **Modelos Prescritivos / Processos Tradicionais:**
    - São modelos que definem um **conjunto distinto de atividades**, **tarefas**, **marcos** e **produtos** de trabalho que são **necessários** para fazer **Engenharia de Software** com **alta qualidade**
    - **Processos** que se baseiam em uma **descrição de como as atividades são feitas**
  - **Metodologias Ágeis (Próximas Aulas):**
    - Modelos que definem um **conjunto de valores, princípios e práticas** que **auxiliam** a **equipe de projeto** a **entregar produtos** ou **serviços** de **valor** em um ambiente
    - Valor central: As **respostas às mudanças** são mais importantes que o cumprimento de um plano

Revisão de Engenharia de Software: Modelo Cascata com Realimentação



Revisão de Engenharia de Software: RUP (*Rational Unified Process*)

- É um processo configurável de Engenharia de Software
- O RUP é um guia para como usar efetivamente a UML



Revisão de Engenharia de Software: O que é agilidade?

- Agilidade??
  - Rapidez, desembaraço
  - Qualidade de quem é veloz
- Capacidade de responder rapidamente a **mudanças**
  - Mudanças de tecnologias, de equipe, de requisitos...
- Entregar valor ao cliente quando se lida com imprevisibilidade e dinamismo dos projetos
- Problema
  - Aparenta ser indisciplinado
  - Em muitos casos não é só aparência
  - Por quê?
    - Pois confundem agilidade com ausência de documentação

Revisão de Engenharia de Software: Manifesto Ágil

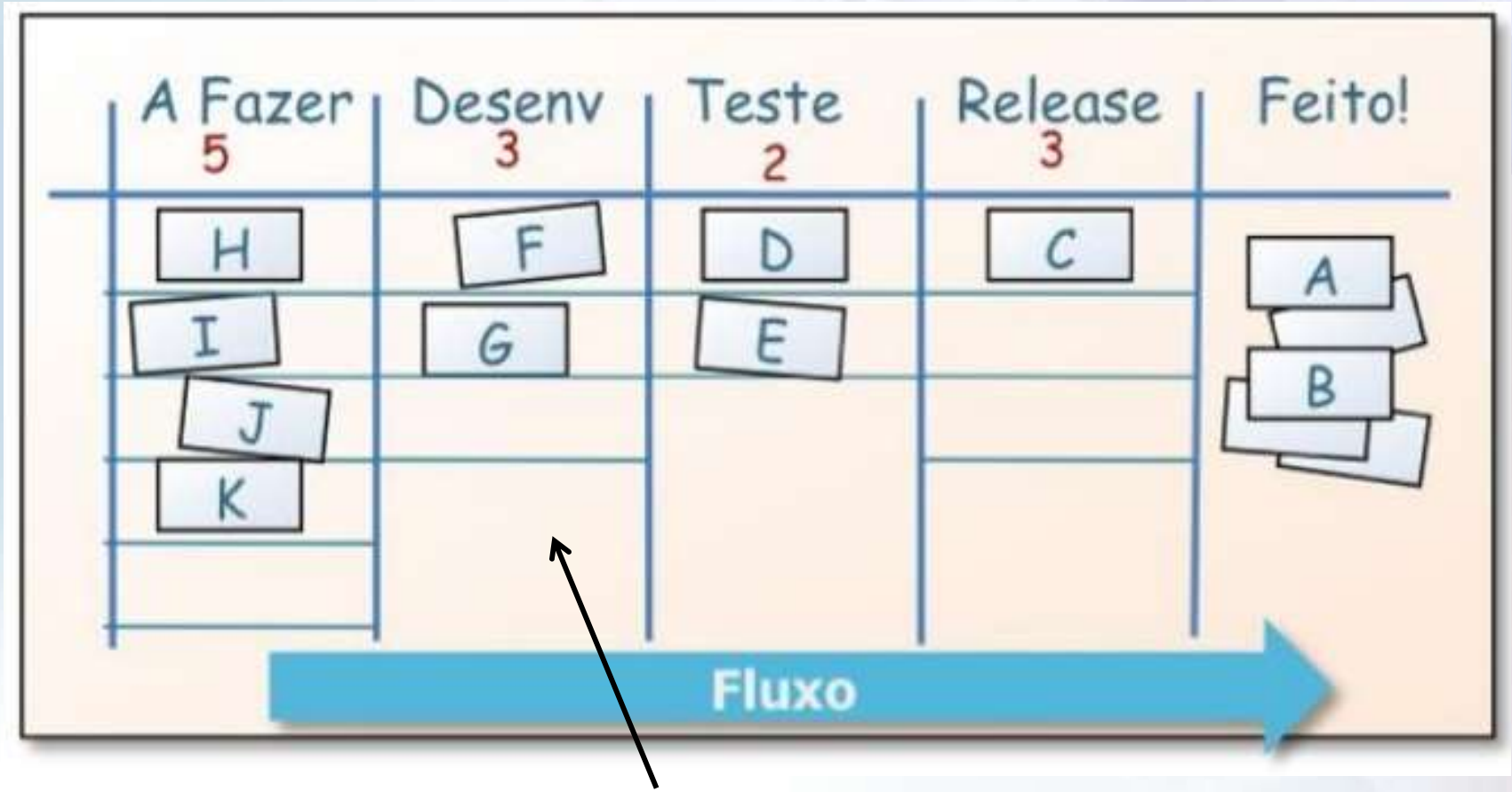
- Estamos descobrindo melhores formas de **desenvolver software** através da nossa própria prática e auxiliando outros.

Valores

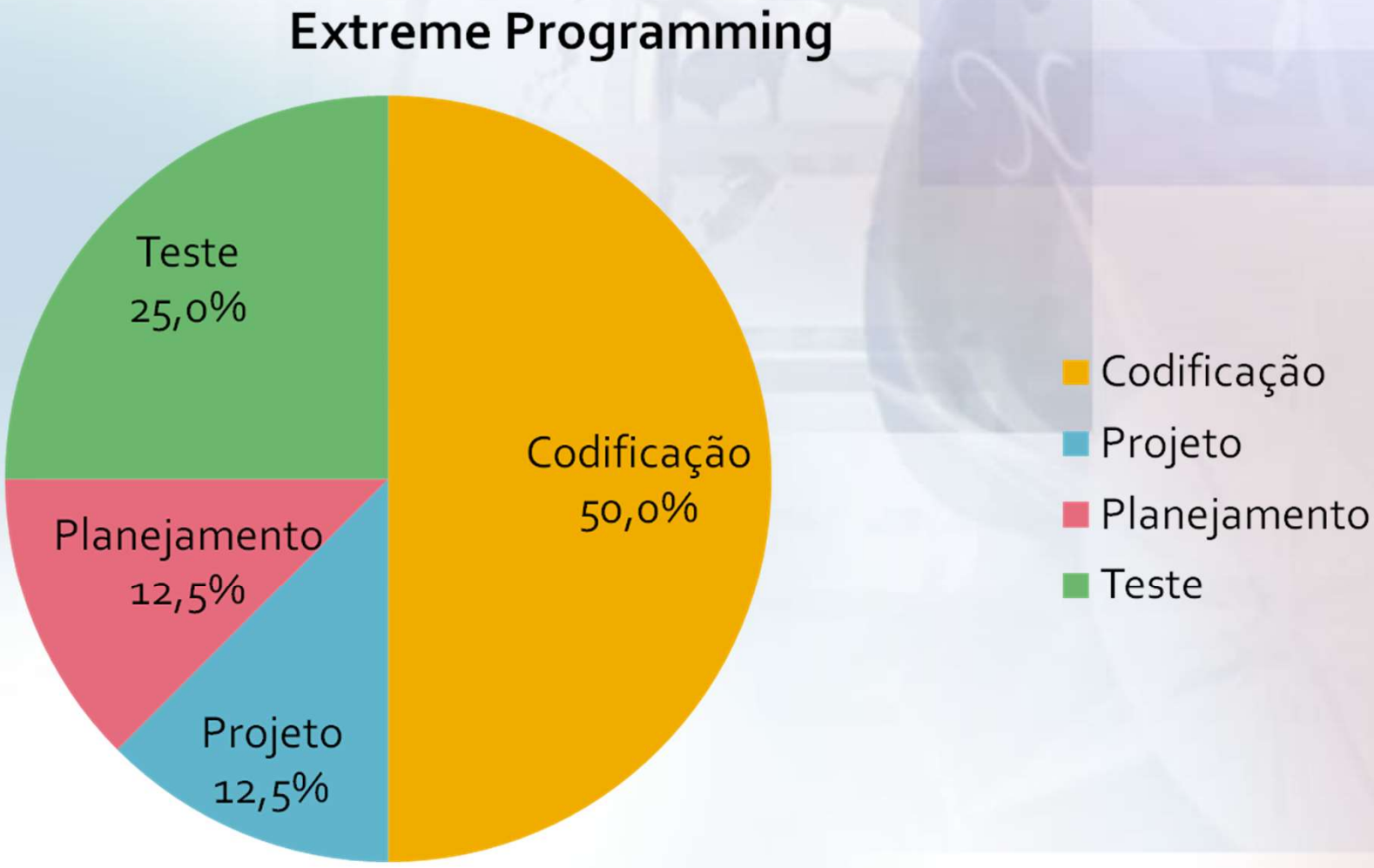
Mais Importante → Menos Importante

Indivíduos e Iterações	**X**	Processos e Ferramentas
Software funcionando		Documentação detalhada
Colaboração com cliente		Negociação de contratos
Responder a mudanças		Seguir um plano

Revisão de Engenharia de Software: Kanban

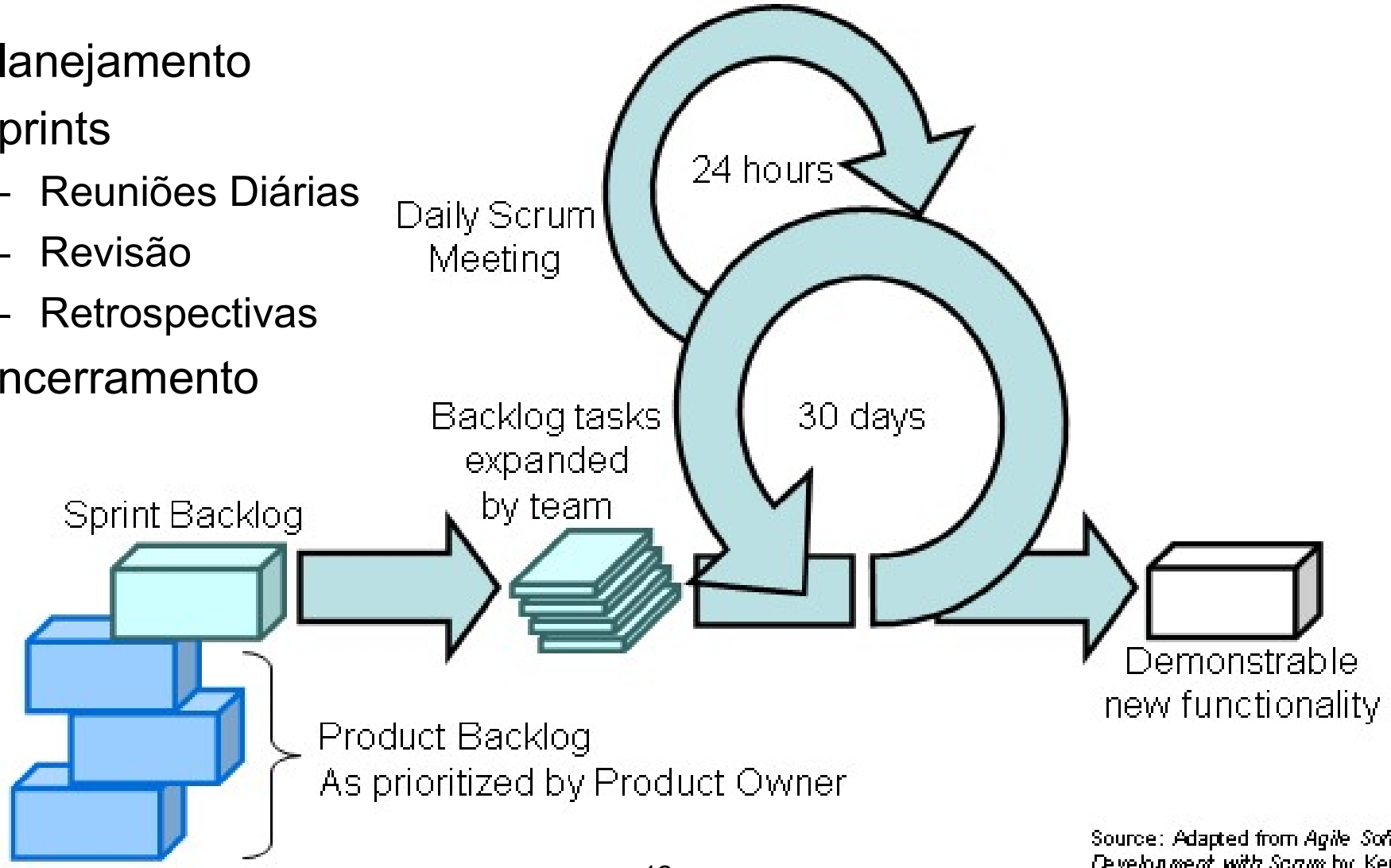


Revisão de Engenharia de Software: XP (*Extreme Programming*)



Revisão de Engenharia de Software: SCRUM

- Planejamento
- Sprints
  - Reuniões Diárias
  - Revisão
  - Retrospectivas
- Encerramento



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

Motivações para realizar Teste de Software

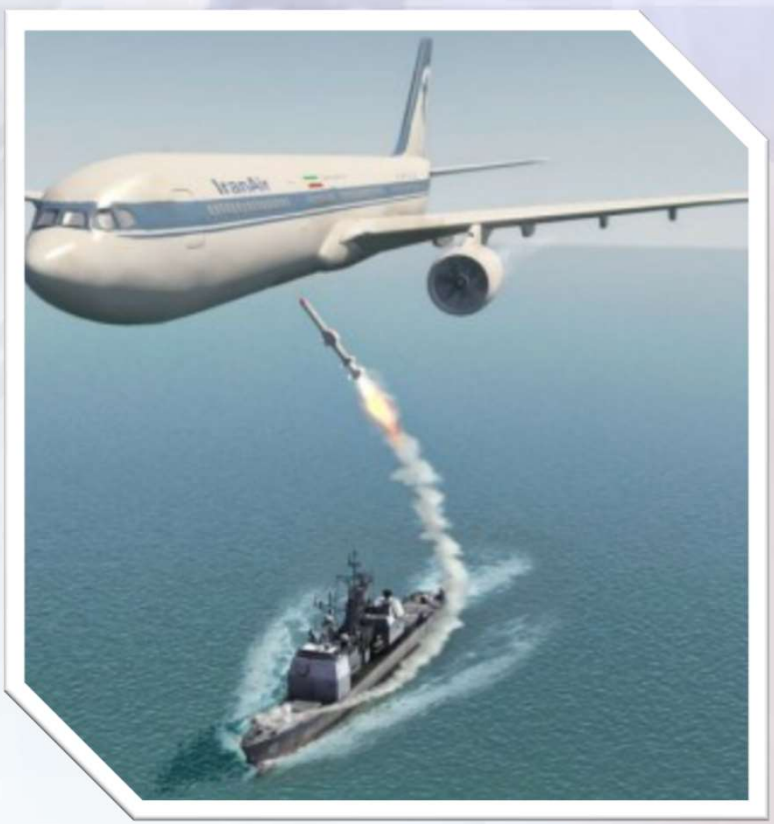
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Motivação para Testar (1)

- **Mars Climate Orbiter (1998):**
  - Objetivo:
    - Enviar sinais a partir de marte, após seu pouso no planeta
  - Desastre:
    - Chocou-se com o Planeta
  - Motivo:
    - Bug no software responsável pela conversão de medidas para aproximação
      - A nave espacial deveria efetuar sua inserção na órbita de Marte a uma altitude de 140 a 150 km da superfície
      - Devido a um erro de cálculo, a manobra de inserção orbital foi feita a uma altitude de 57 km → causou a destruição da nave espacial pela sua fricção com a atmosfera de Marte
      - A equipe fez o uso de medidas inglesas (jardas, pé, milhas) para calcular os parâmetros para a manobra de inserção orbital, enviando-os à nave, cujos sistemas realizavam cálculos no sistema métrico (quilômetros, metros, centímetros)
  - Prejuízo:
    - **U$ 165 milhões**

Motivação para Testar (2)

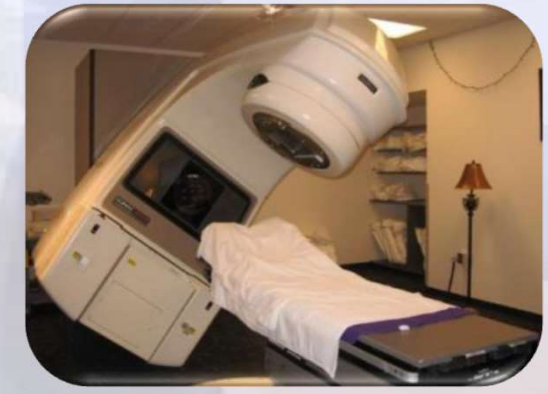
- **Airbus 320 (1988):**
  - Desastre:
    - USS Vicennes derrubou um Airbus 320 em 1988
  - Motivo:
    - Bug no software de reconhecimento do USS Vicennes, confundindo o avião com um F-14
  - Prejuízo:
    - **290 mortes**

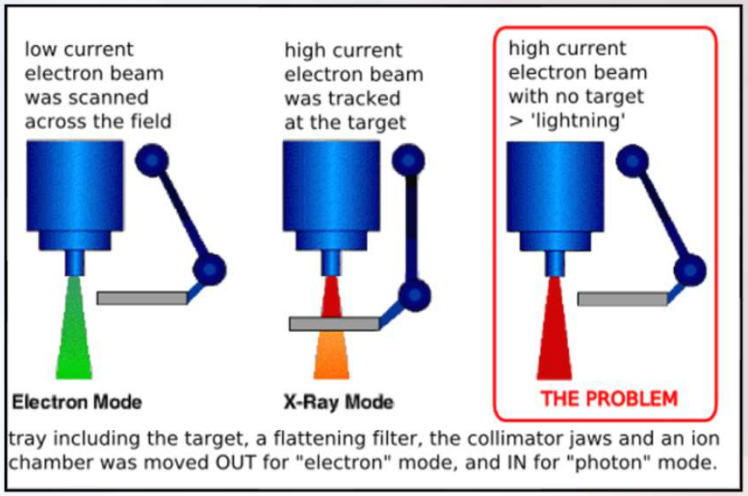


Link: https://www.naval.com.br/blog/2018/05/18/ha-30-anos-o-cruzador-uss-vincennes-derrubava-um-airbus-matando-290-civis/

Motivação para Testar (3)

- **Therac-25: Máquina de Terapia Radioativa (1985 – 1987):**
  - Desastre:
    - Overdose em pacientes sob tratamento entre 1985 e 1987
  - Motivo:
    - Inabilidade em gerenciar certas condições de disputa
  - Prejuízo:
    - **Morte de 2 pessoas**
    - **06 pessoas gravemente feridas**
  - Artigo que detalha sobre o ocorrido: Link





Motivação para Testar (4)

- **London Ambulance Service (1992):**
  - Desastre:
    - Serviço auxiliado por computador falhou nos dias 26 e 27 de Novembro de 1992, gerando várias falhas, como o envio de 02 ambulâncias para o mesmo destino, envio de uma ambulância para um local estando outras mais próximas, etc...
  - Motivo:
    - Tudo indica que o problema estava relacionado a alta carga de emergências durante o período
  - Prejuízo:
    - **Morte de 20 pessoas**
  - Artigo que detalha sobre o ocorrido: Link

A Comedy of Errors: the London Ambulance Service case study

Anthony Finkelstein & John Dowell
School of Informatics, City University, UK
{acwf@soi.city.ac.uk, johnd@soi.city.ac.uk}

Abstract

This paper provides an introduction to the IWSSD-8 case study - the "Report of the Inquiry Into the London Ambulance Service". The paper gives an overview of the case study and provides a brief summary. It considers how the case study can be used to orient discussion at the workshop and provide a bridge between the various contributions.

Introduction

The International Workshop on Software Specification & Design has established a tradition of using "case studies" to focus and provide coherence to its intensive working sessions. These case studies, supplied in advance to participants in the various tracks, have proved a fruitful way of working. Evidence of this can be seen most clearly in the "succeedings" or workshop reports which have followed previous workshops. It was decided for IWSSD-8 that, in order to provide common ground between the tracks, a single shared "case study" should be used, with each track drawing on it in a manner appropriate to their own interests and concerns. After some discussion we settled on the "Report of the Inquiry Into the London Ambulance Service" which is interesting in its own right, reflects aspects of requirements architecture, design, concurrency and distribution, and raises significant issues on the relation between these aspects.

The report is available by ftp, details of how to obtain it can be found at the bottom of the paper. Subsequent comments in this paper assume that you have access to the report. References in this paper are to report paragraph numbers.

Overview & Summary

Like most computing professionals in the UK we were aware of the failure, using this term broadly, of the computer aided despatch (CAD) system deployed by the London Ambulance Service (LAS) in, or shortly after, October 1992. We suspect, as London residents, we were more immediately aware of it than most. At any rate, both of us read the items that appeared in the newspapers with considerable interest and concern.

Neither of us can remember when we first saw a copy of the report, probably in summer 1993, but both remember clearly our initial reactions - a mixture of horror and, we must confess, a certain macabre enjoyment. If not a comedy of errors it is at least a compounding of them. It seemed, on first reading, as if everything had gone wrong - every component of good engineering practice had been ignored, every guideline of software engineering disregarded, basic management principles neglected, even the dictates of common sense overlooked. Subsequent readings have rather changed our understanding of the failure which now emerges as an example of "systemic failure" or "normal accident" of the type identified by Perrow (1984). This having been said it is evident that at the heart of the failure are breakdowns in specification and design common to many software development projects and that the context in which they occurred is far from atypical. Therein lies its particular interest and challenge from our standpoint.

The failure, and the subsequent reaction to it must be understood in a broad political setting. The National Health Service (NHS), the government supported "free-at-the-point-of-use" system of health care provision in the UK, was undergoing considerable changes - in particular the move towards more decentralised and directly financially accountable management. These changes were combined with a lack of prior investment, significant ongoing resource pressures and a reallocation of NHS priorities drawing money away from London. A further feature of the political environment was a strong focus on the "effectiveness" or "performance" of public services. The mix of these changes with a combative political scene and fraught labour relations gave a particular significance and weight to the failure and lead to the establishment of the inquiry which reported in February 1993.

Motivação para Testar (5)

- **Airbus A300 China Air Lines (1994):**

  - Desastre:
    - Avião caiu em 1994
  - Motivo:
    - Foi feita uma investigação e dentre as recomendações. Aconselharam mudanças no software de controle
      - Piloto sem querer pressiona o botão Takeoff/Go-around (TO/GA) o que aumentou a posição do acelerador
      - Os pilotos tentaram corrigir manualmente a situação reduzindo os aceleradores e empurrando os maçanetas para baixo
      - O piloto automático agiu contra as ações dos pilotos (como estava agendado que fizesse em caso que o botão TO/GA estivesse ativado), causando que o nariz se inclinasse abruptamente.
      - O acidente foi atribuído a um erro da tripulação por não ser capaz de corrigir os controles e a velocidade do ar. Nove meses antes a Airbus advertira os seus clientes de fazer uma modificação no sistema de vôo que podia desligar completamente o piloto automático "quando se aplica certas entradas a controles manuais no modo TO/GO
      - O avião acidentado tinha que receber a atualização, mas demoraria um tempo que a empresa não gostaria de perder e julgou que as modificações não eram urgentes
      - Considerou-se que esses fatores incidiram no acidente, depois de que o erro primário dos pilotos de fazer com o controle da situação começasse
  - Prejuízo:
    - **264 mortes**

Motivação para Testar (6)

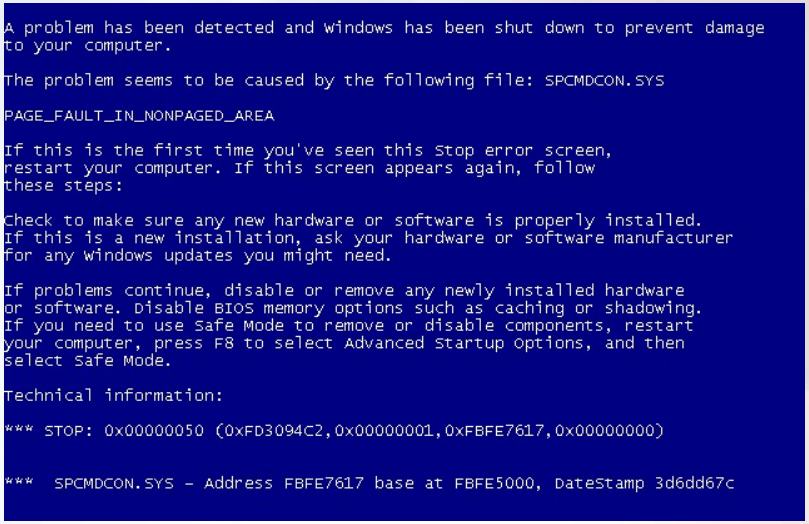
- **Outros Exemplos em Vídeo:**

Airbus 320:

Ariane 5:

Para pensar (1)

- Expectativa:
  - Pode-se esperar que o software funcione corretamente?
  - Softwares feitos com bastante cuidado:
    - 05 falhas / 1000 LOC
    - Software com 1 Milhão de LOC → 5000 falhas
  - Exemplo:
    - Windows XP teve 45 Milhões de LOC
    - 45 x 5000 = 225.000 falhas



Para pensar (2)

- **A importância das atividades de Qualidade:**
  - Piada: Se a indústria automobilística tivesse se desenvolvido como a indústria de software, existiriam carros por R$100,00, fazendo 5000 km com um galão de combustível
  - Porém, esse carro iria "quebrar" duas vezes por dia, sem motivo aparente, e quando você solicitasse assistência junto as concessionárias, eles iriam dizer para ser reinstalado o motor!
  - Carros são mais confiáveis que software????

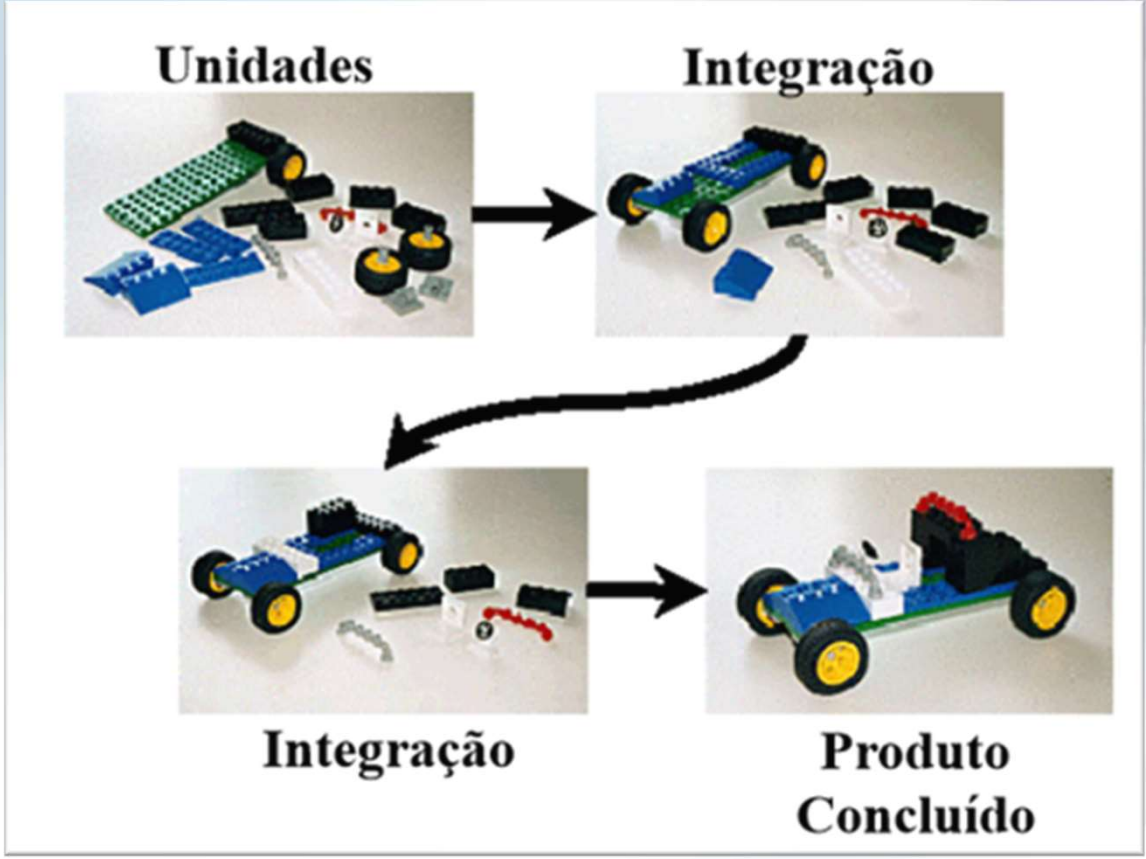
vs.

Para pensar (3)

- **A importância das atividades de Qualidade:**
  - Como os carros são desenvolvidos?
    - Requisitos:
      - Motor, rodas, ar-condicionado, som, espaço para bagagem, ...
    - Projeto Detalhado:
      - Projeto Arquitetônico, revisado várias vezes
    - Verificação do Projeto:
      - Simulação e Protótipos
    - Desenvolvimento dos Componentes:
      - Testa-se cada componente
      - Componentes são reutilizáveis
      - Produzidos em Massa
    - Montagem do Carro:
      - Testa-se o carro (teste de batidas, teste de resistência, teste de estabilidade)
      - Teste de Usabilidade

Para pensar (4)

- **A importância das atividades de Qualidade:**
  - Como os carros são desenvolvidos?



Teste de Software

Conceitos e Breve Introdução

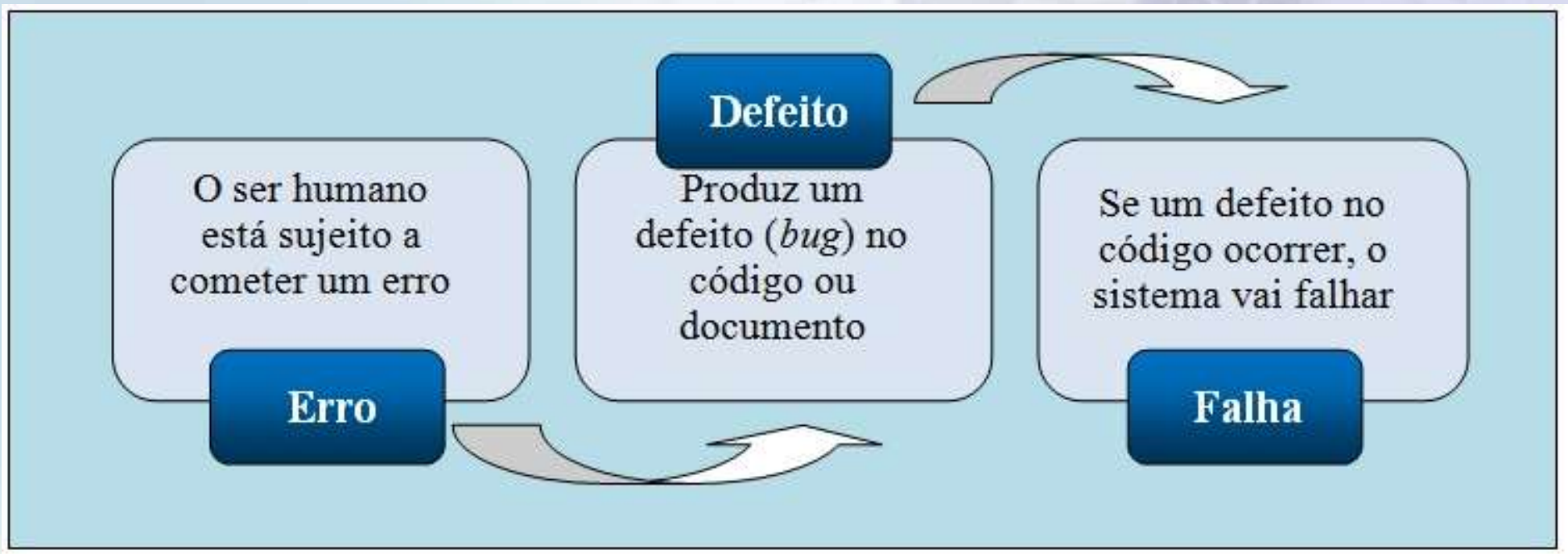
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Falhas vs. Defeito vs. Erro (1)

Quem provoca quem?

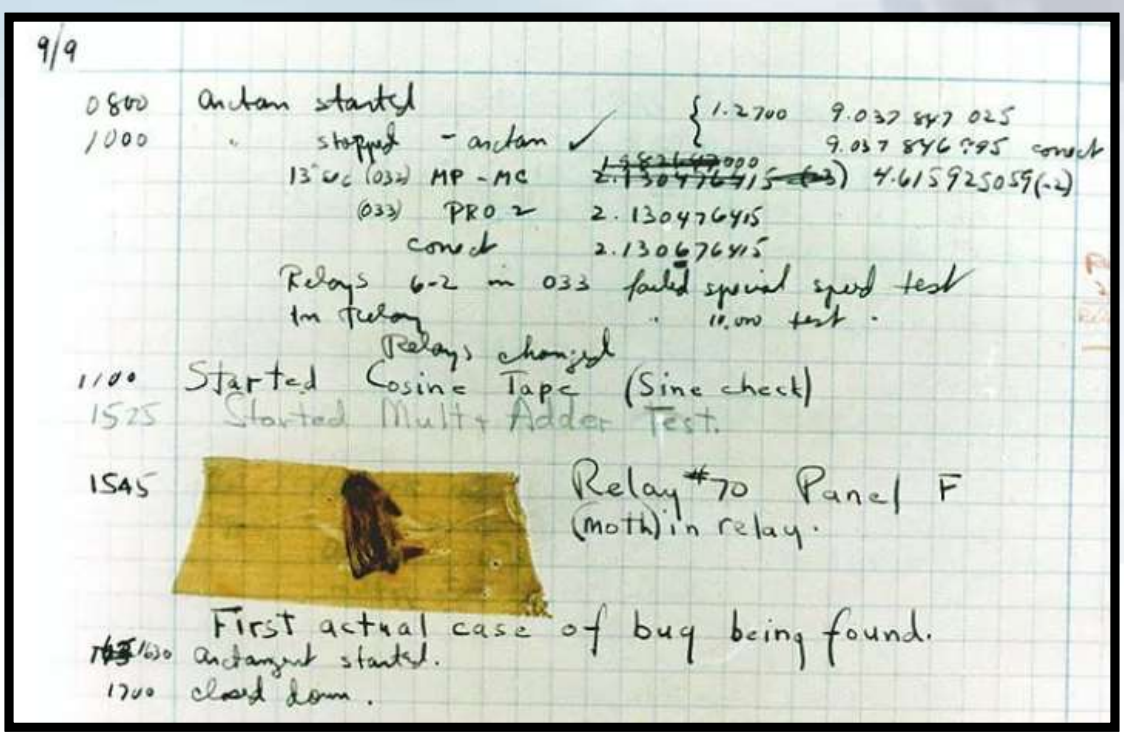
- **Falha:**
  - Incapacidade do software de realizar a função requisitada
- **Defeito (Bug):**
  - Causa de uma falha: Um defeito em um sistema pode ocorrer devido a omissão de informações, definições de dados ou comandos/instruções incorretas dentre outros fatores
- **Erro:**
  - Pode resultar em defeito, se propagado até a saída
  - Pode ser causado por uma ação humana

Falhas vs. Defeito vs. Erro (2)



Origem do Termo Bug (Defeito)

- Bug = “inseto” → Defeito
  - Termo utilizado pela primeira vez por Grace Hopper, programadora da marinha dos EUA, em 1947
    - Em um diário de bordo, Hopper usou a palavra para explicar o mau funcionamento do computador Mark II, da Universidade de Harvard
    - A falha teria sido causada por um inseto que ficou preso nos contatos de um relê, daí a associação com o bichinho e com as falhas em computadores



Teste de Software

- Representam uma oportunidade de detectar **defeitos** antes do software ser entregue aos usuários
- A atividade de testes pode ser feita de forma **manual** e/ou **automática** e tem por objetivos:
  - **Detectar Erros** para **Eliminar** os **Defeitos** e **Evitar** as **Falhas**
  - **Produzir casos de teste** que tenham elevadas probabilidades de **revelar um defeito ainda não descoberto**, com uma quantidade mínima de tempo e esforço
  - **Comparar o resultado dos testes com os resultados esperados** → produzir uma indicação da qualidade e da confiabilidade do software. Quando há diferenças, inicia-se um processo de depuração para descobrir a causa

Finalidade dos Testes

- **Averiguar** se todos os **requisitos** do sistema foram **corretamente** implementados
- **Assegurar**, na medida do possível, a **qualidade** e a **corretude** do **software** produzido
- **Reduzir custos** de **manutenção corretiva** e retrabalho

Teste de Software (1)

- A realização de testes não consegue mostrar ou provar que um software está correto
- O máximo que os testes de um software conseguem provar é que ele contém erros
- Quando os testes realizados com um determinado software não encontram erros críticos, haverá sempre duas possibilidades:
  - A qualidade do software é aceitável
  - Os testes foram inadequados (ineficientes)

Nunca podemos falar que o software encontra-se 100% testado!!!!

Teste de Software (2)

- Exemplo de impossibilidade de se testar tudo
- Análise do método a seguir (Binder, 2000):

```
int blech(int j) {
    j= j -1;          // deveria ser  j = j +1
    j=j/30000;
    return j;
    }
```

Teste de Software (3)

```
int blech(int j) {
    j= j -1;          // deveria ser  j = j +1
    j=j/30000;
    return j;
    }
```

Input	Resultado esperado	Resultado obtido
1	0	0
42	0	0
40000	1	1
-64000	-2	-2

- Se este pequeno método, com apenas 4 linhas de código necessita ser testado com 124.000 casos de teste, imaginem um código usual, com milhares ou milhões de linhas de código!

- Então, com quais valores inteiros (-62.000 a +62.000) detectaríamos este erro?

Teste de Software (4)

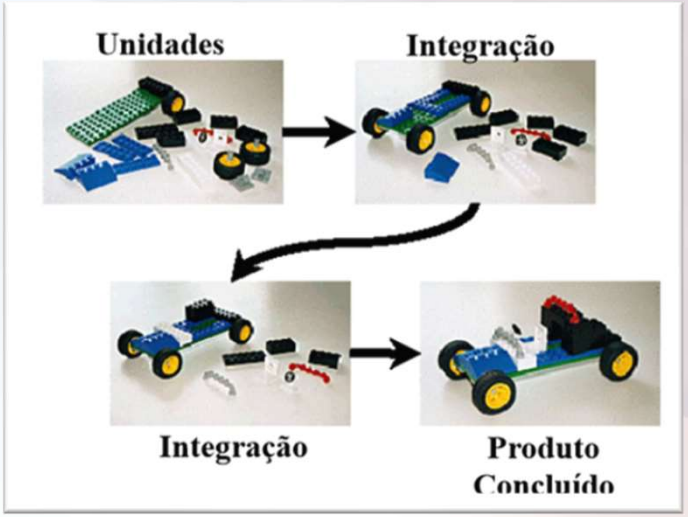
```
int blech(int j) {
    j= j -1;          // deveria ser  j = j +1
    j=j/30000;
    return j;
    }
```

- Então, com quais valores inteiros (-60.000 a +60.000) detectaríamos este erro?

Entradas	j = j + 1		j = j - 1	
-60.000	-59.999	-1	-60.001	-2
-59.999	-59.998	-1	-60.000	-2
-30.000	-29.999	0	-30.001	-1
-29.999	-29.998	0	-30.000	-1
29.999	30.000	1	29.999	0
30.000	30.001	1	29.999	0
59.999	60.000	2	59.998	1
60.000	60.001	2	59.999	1

Estágios de Teste (1)

- Existem diferentes estágios de teste associados ao desenvolvimento de um produto de software:
  - Teste de Unidade
  - Teste de Integração
  - Teste de Sistemas
  - Teste de Aceitação (Homologação)



Estágios de Teste (2)

- **Teste de Unidade:**
  - Visa **testar individualmente cada um dos componentes** (programas ou módulos), procurando garantir que funcionem adequadamente

Estágios de Teste (3)

- **Teste de Integração:**
  - Visa testar o relacionamento entre as diversas **unidades integradas** → Garantir que a interface entre os módulos funcione adequadamente, pois não há garantias de que **unidades testadas em separado funcionarão em conjunto**

Estágios de Teste (4)

- **Teste de Sistema:**
  - Conjunto de testes cujo objetivo primordial é colocar completamente à prova todo o sistema, baseado em um computador → Testa a integração do software com o **ambiente operacional** – hardware, pessoas e dados reais.

Estágios de Teste (4)

- **Teste de Aceitação (Homologação):**
  - São testes realizados pelo cliente / usuário com o objetivo de **validar o sistema** a ser implantado
  - A motivação maior para esses testes é o fato do **desenvolvedor nunca conseguir prever como o usuário realmente usará** um software numa situação real

Estágios de Teste (5)

- **Teste de Aceitação (Homologação):**
  - **Testes Alfa:**
    - São feitos por um determinado cliente, geralmente nas **instalações do desenvolvedor**, que observa e registra os erros e/ou problemas
  - **Teste Beta:**
    - São realizados por possíveis clientes, **em suas próprias instalações, sem a supervisão do desenvolvedor**. Cada cliente relata os problemas encontrados ao desenvolver, posteriormente

Para pensar

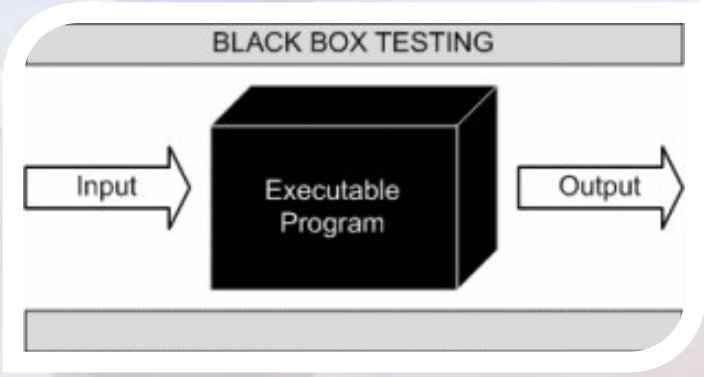
- Se fossemos testar a calculadora simples do Windows (funções básicas), dê exemplos de:
  - Testes Unitários
  - Testes de Integração
  - Testes de Sistema
  - Testes de Aceitação

Abordagens de Teste (1)

- Abordagem Funcional (Caixa Preta)
- Abordagem Estrutural (Caixa Branca)

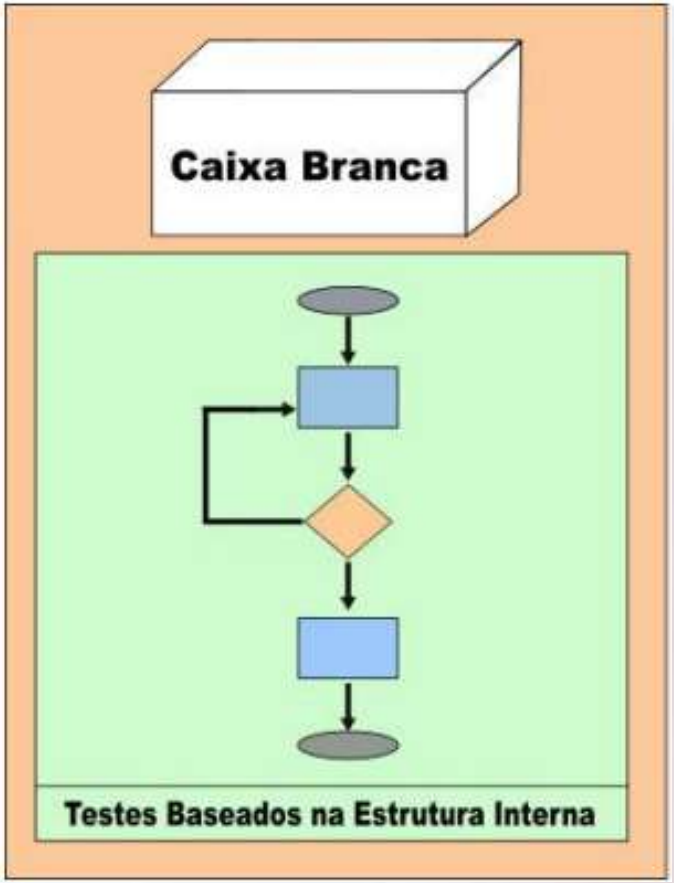
Abordagens de Teste (2)

- **Abordagem Funcional (Caixa Preta)**
  - Concentra-se nas interfaces do software e visa mostrar que:
    - As entradas são aceitas
    - As saídas são as esperadas
    - A integridade dos dados é mantida
  - Aplica-se, principalmente, aos testes de integração, sistemas e aceitação, mas pode ser adaptado e utilizado aos testes unitários



Abordagens de Teste (3)

- **Abordagem Estrutural (Caixa Branca)**
  - Visa mostrar que os componentes internos do software realmente funcionam
  - Em princípio, os testes de caixa branca pretendem garantir que todas as estruturas dos programas e todos os possíveis casos sejam testados
  - Aplica-se, principalmente, aos testes unitários e de integração



Tipos de Teste de Software (1)

- Existem vários tipos de teste que podem ser executados nos diversos estágios de teste e utilizando as diferentes abordagens existentes:
  - **Teste de funcionalidade / cenários**
  - **Teste de recuperação de falhas**
  - **Teste de Segurança de acesso**
  - **Teste de Carga**
  - **Teste de Desempenho**
  - **Teste de Portabilidade**
  - **Teste de Regressão**

Tipos de Teste de Software (2)

- **Teste de funcionalidade / cenários:**
  - **Definição:** Testa a funcionalidade geral do sistema, em termos de regras de negócio (fluxo de trabalho), considerando-se tanto as condições válidas quanto as inválidas
  - **Exemplo:** Deve-se testar tanto o login válido quanto o inválido

Tipos de Teste de Software (3)

- **Teste de Recuperação de Falhas:**
  - **Definição:** Seu objetivo é forçar o software a falhar de diversas maneiras e verificar se a recuperação é adequada
  - **Exemplo:**
    - Aplica-se ao Foguete Ariane 5 que explodiu
    - Verificar como se comporta os sistemas que controlam os aparelhos de um hospital quando a energia acaba

Tipos de Teste de Software (4)

- **Teste de Segurança de Acesso:**
  - **Definição:** Tenta certificar-se de que todos os mecanismos de proteção embutidos no software, de fato, o protegerão dos acessos indevidos
  - **Exemplo:**
    - Segurança Lógica: Tentar acessar uma funcionalidade que o usuário não tem privilégio
    - SQL Injection

Tipos de Teste de Software (5)

- **Teste de Carga:**
  - **Definição:** Tenta confrontar o software ou os programas com situações anormais. Ele executa o software de uma forma que exige recursos em quantidade, frequência e volume bem maiores do que o uso normal
  - **Exemplo:** Muitos acessos de usuário fazendo compras em um e-commerce simultaneamente

Tipos de Teste de Software (6)

- **Teste de Desempenho:**
  - **Definição:** São testes que visam verificar o desempenho ou performance do software
    - São, muitas vezes, combinados ou feitos juntamente com os testes de carga (estresse)

Tipos de Teste de Software (7)

- **Teste de Portabilidade:**
  - **Definição:** São testes que verificam o grau de portabilidade (facilidade de instalação ou operação) do produto de software em diferentes ambientes de hardware / software
  - **Exemplo:** Garantir que o sistema Web funcione no Chrome, Opera, Firefox e Edge (Se for possível, ☺)

Tipos de Teste de Software (8)

- **Teste de Regressão:**
  - **Definição:** Reexecução de testes feitos após uma manutenção corretiva ou evolutiva
    - Em processos de desenvolvimento iterativos, muitos dos artefatos produzidos nas primeiras iterações são usados em iterações posteriores

Executores do Teste de Software

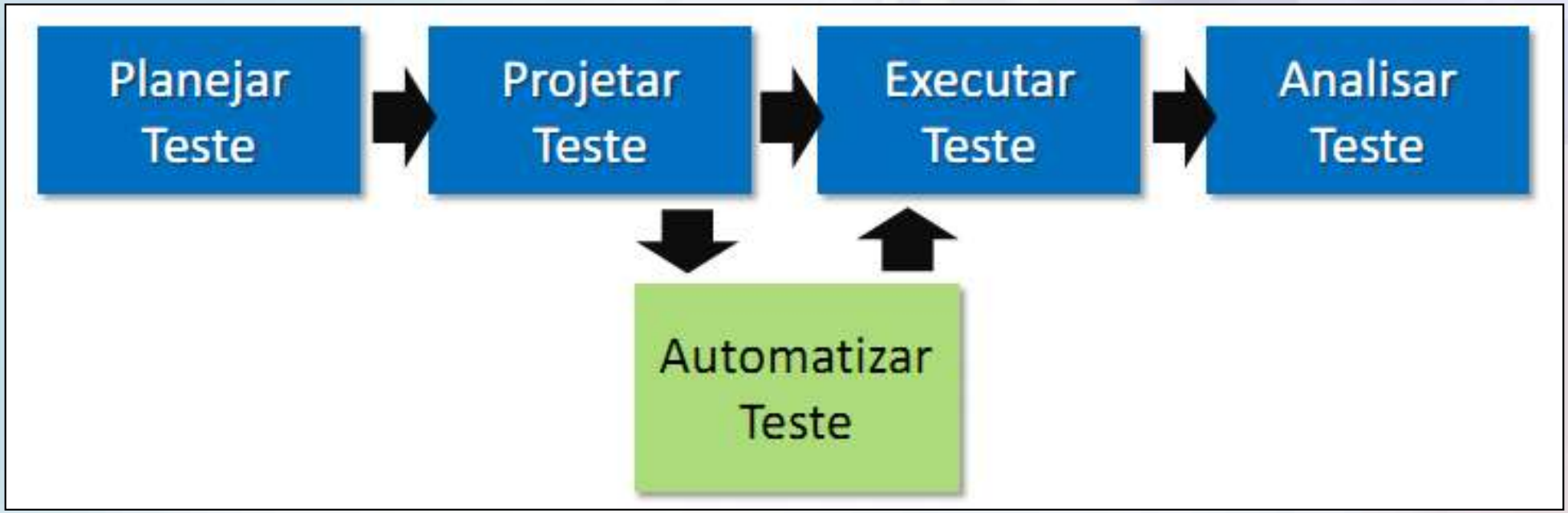
- Os testes, normalmente, são executados pela equipe de desenvolvimento
- No entanto, é ideal que pelo menos os testes de sistemas sejam feitos por uma equipe de testes independente ou pelos usuários.
- **Por quê???**

Ferramentas do Teste de Software

- O processo de testes pode ser desenvolvido com auxílio de ferramentas (automatizados ou não)

- Principais ferramentas existentes:
  - Testlink
  - Jest, JUnit, PHPUnit, NUnit, etc...
  - Cypress
  - Selenium
  - Sonar
  - Jmeter
  - Webload
  - Jabuti
  - Etc...

Próxima Aula: Processo Básico de Teste de Software



Engenharia de Software II / Qualidade e Teste de Software

Aula 02: Motivação e Conceitos Básicos de Teste de Software

Breno Lisi Romano

Dúvidas?

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)
Tecnologia em Sistemas para Internet – TSI (QTSI6)

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista