pvh( );

INSTITUTO FEDERAL
GOIÁS

Instituto Federal de Educação, Ciência e Tecnologia
Campus Inhumas

TESTES DE SOFTWARE

Visão Geral de Testes de Software
Tópico 2 do plano de ensino

Prof. Me. Victor Hugo Lázaro Lopes

AGENDA

2.Visão Geral de Testes de Software;

2.1 Terminologias e conceitos básicos;

2.2 Grandes erros de Software;

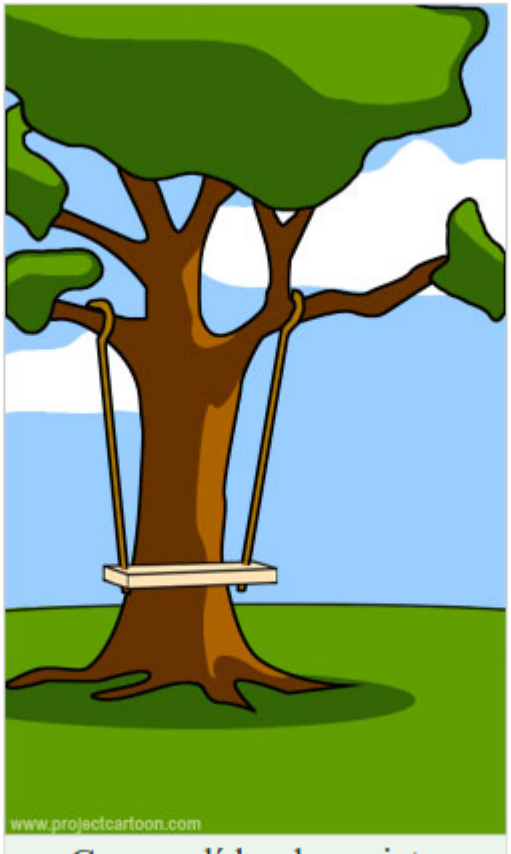
2.3 Bug vs defeito, prevenção vs detecção, verificação vs validação (Já tratado!);

2.4 Dimensões de teste;

2.5 Métricas de teste;

2.6 Visão geral da gerência de requisitos;

Como o cliente explicou

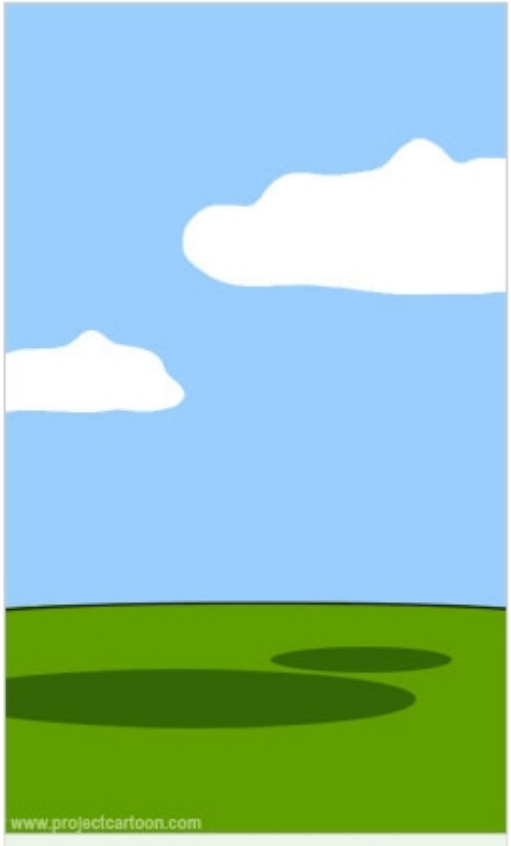


Como o líder de projeto entendeu

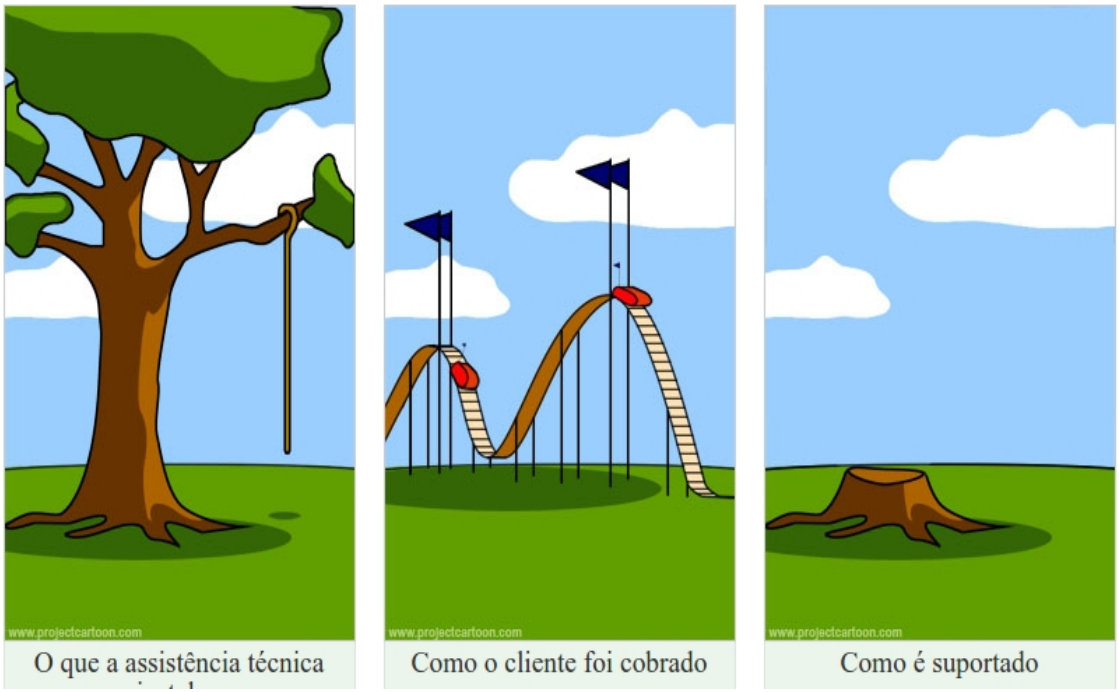
Como o analista planejou

Como o programador codificou

Como o consultor de negócio descreveu



Como o projeto foi documentado



http://projectcartoon.com/cartoon/611

Relembrando um ponto importante:

Matriz de responsabilidade

Os técnicos envolvidos no processo de testes possuem os seguintes perfis:

Líder do Projeto de Testes - LT	Técnico responsável pela liderança de um projeto de teste específico, normalmente relacionado a um sistema de desenvolvimento, seja um projeto novo ou uma manutenção.
Arquiteto de Teste - AT	É o técnico responsável pela montagem da infra estrutura de teste (ver a etapa de preparação), montando o ambiente de teste, escolhendo as ferramentas de teste e preparando a equipe para executar o seu trabalho neste ambiente de teste, seja em desenvolvimento ou em produção.
Analista de Teste - AN	É o técnico responsável pela modelagem e elaboração dos casos de teste e pelos scritps de teste. Pode ser que em alguns casos os scripts de teste sejam elaborados pelos testadores.
Testador - TE	Técnico responsável pela execução dos casos de teste e scripts de teste.

Big Picture

“Você pode enganar poucos durante muito tempo.
Você pode enganar muitos durante algum tempo,
Mas você não pode enganar todos o tempo todo.”
Sérgio Moro!! ;)

Não adianta se enganar, pois os “bugs” sempre existiram. O que se faz é diminuir o risco de ocorrer um “bug”, ou de, no mínimo, que esse “bug” gere menor impacto.

Big Picture

O supremo estado da arte em 1947 era o computador Mark II construído em Harvard. Um certo dia, técnicos operavam-no quando, de repente, parou. Eles se esforçaram e descobriram que o problema estava no contato entre relays e o computador, devido ao excesso de calor e movimentos do computador e à alta voltagem utilizada junto ao relay. Era na prática uma sobrecarga. Nascia aí o primeiro erro de computador. Esse erro não sumiu, foi controlado.

Hoje os “bugs” viraram uma ciência ao inverso: como caçá-los?

Terminologias

Segundo PRESSMAN (2011):

- **Testabilidade –** é a facilidade com que um programa de computador possa ser testado. As seguintes características levam a um software testável:
  - **Operabilidade**: Quanto melhor funcionar, mais eficientemente poderá ser testado. Se for projetado e implementado tendo em mente a qualidade, haverá poucos defeitos bloqueando a execução dos testes, permitindo que o teste ocorra sem sobressaltos.
  - **Observabilidade:** "O que você vê é o que você testa". Entradas fornecidas como parte do teste produzem saídas distintas. Saídas incorretas são facilmente observáveis e mensuráveis, e o estado atual do sistema são visíveis. O código fonte é acessível.
  - **Controlabilidade:** Todas as possíveis saídas podem ser geradas por meio de alguma combinação de entrada, e os formatos de entrada e saída são consistentes e estruturados. Os testes podem ser convenientemente especificados, automatizados e reproduzidos.

Terminologias

Segundo PRESSMAN (2011):

- **Testabilidade –** é a facilidade com que um programa de computador possa ser testado. As seguintes características levam a um software testável:
  - **Decomponibilidade:** Sistema construído a partir de módulos independentes que podem ser testados de forma independente.
  - **Simplicidade:** Programa com simplicidade funcional (características mínimas para cumprir os requisitos), simplicidade estrutural (arquitetura modular para limitar a propagação de falhas) e simplicidade de código (padrão de escrita).
  - **Estabilidade:** Software recupera-se bem de falhas, e as alterações nele são pouco frequentes, e quando ocorrem não resultam em perdas aos teste anteriores.
  - **Compreensibilidade:** O projeto arquitetural e as dependências entre componentes internos, externos e compartilhados são bem compreendidas e a documentação técnica é acessível.

2.2. Grandes erros de software

Disney - Em 1994, nos EUA, a Disney lançou um CD-ROM com um jogo multimídia baseado no filme Rei Leão. A campanha de marketing foi grande. Era o jogo do momento para todas as crianças. No dia 26 de dezembro, o telefone do SAC da Disney não parou de tocar com reclamações. Eram pais furiosos por causa de seus filhos que choravam sem parar devido ao jogo não funcionar. As vendas caíram direto. A história foi para na TV e nos jornais.
A disney resolveu testar o jogo em diferentes modelos de PC disponíveis no mercado. O software trabalhava corretamente com poucos modelos, justamente os usados pelos programadores, pouco comuns no mercado.

2.2. Grandes erros de software

Intel Pentium – em 1994 foi rastreado um resultado inesperado devido a problemas de divisão. O problema foi colocado na Internet e milhares de pessoas duplicaram o problema com situações adicionais. Por sorte da Intel, este era um caso de extrema raridade que aparecia em cálculos pesados de engenharia ou científicos.

Para a intel não era um “bug” de fato. Ele foi descoberto pelos engenheiros antes dele ser lançado, mas por decisão gerencial o problema não era severo. Porém, a repercussão foi negativa.

A intel teve um prejuízo de $400 milhões com reposição de chips.

2.2. Grandes erros de software

Nasa – em 1999 a sonda da NASA Mars Polar Lander desaparecia num pouso em Marte. Motivo: um bit inesperado no conjunto de instruções do programa. Na teoria era simples: quando a sonda percebia a superfície, lançava um pára-quedas para diminuir a velocidade da queda. Alguns segundos antes de o pára-quedas abrir, as pernas da sonda abriam, colocando-se em posição de pouso. Na prática, quando estivesse a 1800 metros de altura, a sonda acionava o pára-quedas e também uma “cama de pouso” para gentilmente pousar. Para economizar dinheiro, foi simplificado o mecanismo, determinando que a cama de pouso fosse desligada (poupar combustível). Em resumo, as engrenagens iriam “fritar” até que as pernas da sonda tocassem o solo. O problema foi descoberto depois de vários testes similares. Mas por que não foi descoberto antes? Simples: as equipes de engenheiros de teste (que eram distintas e não se integravam) testaram cada uma das partes do sistema como um todo. E ainda cada um testava do seu ponto de vista sem se importar com o processo como um todo.

2.2. Grandes erros de software

Míssil Patriot - EUA – em 1991 o míssil de guerra Patriot, usado como parte do sistema de defesa dos EUA, teve problemas sérios quando foi usado efetivamente na defesa do míssil Scud do Iraque. Ele acertou vários alvos errados. O problema foi um “pequeno” erro no temporizador do sistema de horas do míssil, pois quando o contador acumulava mais que 14 horas, acarretava um erro no sistema de rastreamento. O erro era de acurácia ou de precisão em relação aos alvos destinados. No ataque a Dhahran, o sistema “operou” por mais de 100 horas.

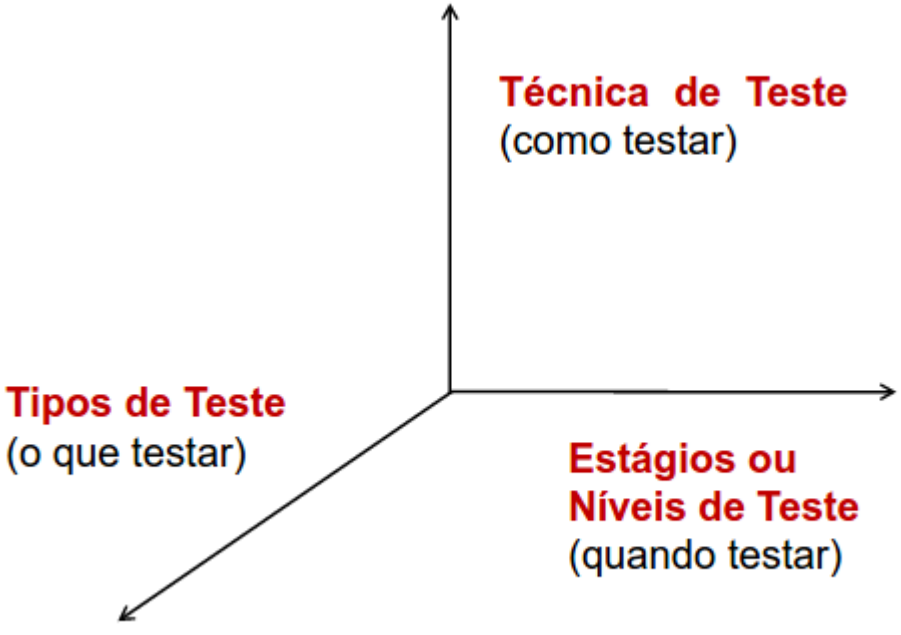
Prevenção de defeitos: Análise de Causa Raiz

RCA (Root Cause Analysis) é uma maneira de identificar as causas de um problema, afinal os problemas são melhores resolvidos ao tentar corrigir ou eliminar as suas causas.

Como podemos implementar a Análise de Causa Raiz?
Há muitas técnicas, com as quais podemos implementar a Análise de Causa Raiz, entre as principais se encontram:

Diagrama de Causa e Efeito (espinha de peixe): permite identificar, explorar e apresentar graficamente todas as possíveis causas, relacionadas a um único problema.
Cinco Porquês (5-why): desenvolvida por Sakichi Toyoda (fundador da Toyota), é baseada na realização de 5 iterações perguntando o porquê daquele problema, sempre questionando a causa anterior. E na prática não é necessário fazer 5 perguntas, pode ser mais ou menos, o importante é chegar à causa do problema.
Reunião de Análise Causal: as causas do problema são levantadas em reuniões do tipo “Brainstorming”.

Dimensões de teste de software, segundo IEEE 829-2008:



Primeira Dimensão: Estágio do Teste (“O Momento”):

Podem-se considerar três* (MOLINARI, 2008) tipos básicos:

***O teste de aceitação seria um quarto tipo, que determina a aceitação do usuário.**

Primeira Dimensão: Estágio do Teste (“O Momento”):

Teste Unitário – ou teste de unidade

Estágio mais baixo da escala de testes

Verifica o funcionamento de um “pedaço” do Software que possa ser testado isoladamente.

Sempre que possível pode ser automatizado.

Normalmente feito pelo programador na fase de implementação.

Pode ser conduzido em paralelo para diversos outros componentes.

Primeira Dimensão: Estágio do Teste (“O Momento”):

Teste Unitário – ou teste de unidade

Estágio mais baixo da escala de testes

Verifica o funcionamento de um “pedaço” do Software que possa ser testado isoladamente.

Sempre que possível pode ser automatizado.

Normalmente feito pelo programador na fase de implementação.

Primeira Dimensão: Estágio do Teste (“O Momento”):

Teste de Integração

Verifica se os componentes funcionam juntos, conforme as especificações;

É alimentado pelos módulos previamente testados individualmente pelo teste de unidade;

Em geral é feito ao término de cada iteração, dentro de um ambiente operacional controlado;

Normalmente feito pelo analista de sistemas para um módulo ou conjunto de programas.

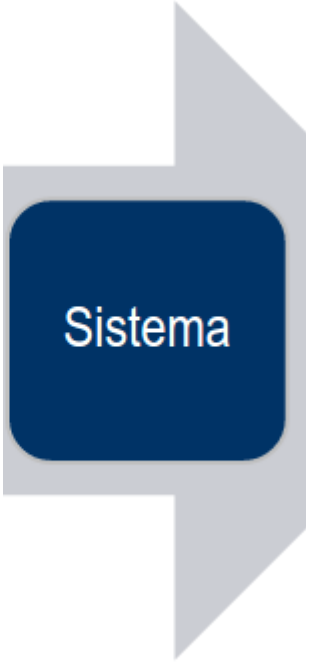
Primeira Dimensão: Estágio do Teste (“O Momento”):

Teste de Sistema

Visa a execução do sistema como um todo ou um subsistema, dentro de um ambiente operacional controlado;

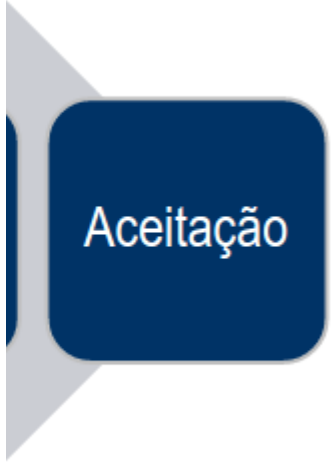
Valida a exatidão e perfeição na execução das funções;

São realizados pela equipe de testes.



Primeira Dimensão: Estágio do Teste (“O Momento”):

Teste de Aceitação



Verifica se a solução atende aos objetivos do negócio e a seus requisitos, no que diz respeito à funcionalidade e usabilidade, antes da sua utilização no ambiente de produção;

Executado no ambiente de homologação;

Testes finais da execução do sistema, realizados pelo usuário.

Segunda Dimensão: O que testar (tipos de teste):

A classificação dos tipos de teste é feita tendo como referência a característica de qualidade que se deseja alcançar. Existem testes específicos para se atingir cada uma das características relacionadas pela norma ISO 9126-1 de 2003.

Característica	Descrição
Funcionalidade	é a capacidade que o software tem de prover funções que atendam aos requisitos implícitos e explícitos.
Portabilidade	capacidade da transferência de um produto de software de um ambiente para outro.
Confiabilidade	capacidade que um produto tem de executar determinada função sem sofrer desgaste ou envelhecimento.
Manutenibilidade	capacidade que o produto de software possui de ser modificado.
Usabilidade	capacidade que o produto tem de ser compreendido pelo usuário.
Eficiência	capacidade do produto de apresentar um desempenho satisfatório quando lhe é proporcionado recursos suficientes.

Segunda Dimensão: O que testar (tipos de teste):

A lista é grande... veremos alguns neste momento

Testes de Carga

visam avaliar a resposta de um software sob uma pesada carga de dados, repetição de certas ações de entrada de dados, entrada de valores numéricos grandes, consultas complexas a base de dados, grande quantidades de usuários simultâneos para verificar o nível de escalabilidade.

Testes Back-to-back

o mesmo teste executado em versões diferentes do software e os resultados são comparados.

Segunda Dimensão: O que testar (tipos de teste):

A lista é grande... veremos alguns neste momento

Testes de Configuração

verificam se o software está apto a funcionar em diferentes versões ou configurações de ambientes (hardware e software).

Testes de Usabilidade

verificam o nível de facilidade de uso do software pelos usuários.

Testes de Instalação

verificam o processo de instalação parcial, total ou atualização do software.

Testes de Segurança

validam a capacidade de proteção do software contra acessos interno ou externo não autorizados.

Segunda Dimensão: O que testar (tipos de teste):

A lista é grande... veremos alguns neste momento

Testes de Recuperação

validam a capacidade e qualidade da recuperação do software após falhas de hardware ou problemas externos.

Testes de Compatibilidade

validam a capacidade do software de executar em um particular ambiente de hardware/software/sistema operacional ou rede.

Testes de Desempenho/Performance

visam garantir que o sistema atenda aos níveis de desempenho e tempo de resposta acordados com usuários e definidos nos requisitos (SLAs).

Segunda Dimensão: O que testar (tipos de teste):

A lista é grande... veremos alguns neste momento

Testes Funcionais

grupos de testes que validam se o que foi especificado foi implementado.

Teste de Qualidade de Código

grupos de testes com o intuito de verificar o código fonte dos programas em consonância com padrões,melhores práticas, instruções não executadas e outros.

Testes Alfa

são executados quando o desenvolvimento está próximo asua conclusão.

Testes Beta

são executados quando o desenvolvimento e testes estão praticamente concluídos.

Terceira Dimensão: Técnica do Teste (“como vou testar”):

Especificação de como serão realizados os testes.

Podem ser divididas em:

- **Técnicas Funcionais**
- **Técnicas Estruturais**

Terceira Dimensão: Técnica do Teste (“como vou testar”):

- **Técnicas Funcionais**

Os testes funcionais têm por objetivo a verificação da entrada dos dados, do processamento, e da resposta a este processamento. Através destes métodos também são verificados se o sistema atende aos requisitos de negócio. Estes métodos são do tipo caixa-preta.

Alguns exemplos destes tipos de teste são:
• Teste baseado em casos de uso
• Valores limites
• Particionamento de equivalência

Terceira Dimensão: Técnica do Teste (“como vou testar”):

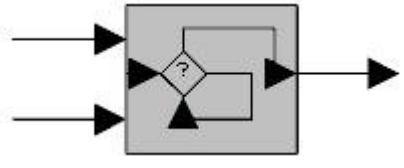
- **Técnicas Funcionais**

Os testes funcionais têm por objetivo a verificação da entrada dos dados, do processamento, e da resposta a este processamento. Através destes métodos também são verificados se o sistema atende aos requisitos de negócio. Estes métodos são do tipo caixa-preta.

Terceira Dimensão: Técnica do Teste (“como vou testar”):

- **Técnicas Estruturais**

Estas técnicas têm por objetivo determinar defeitos na estrutura interna ou no código do software. Também são chamadas de teste de caixa-branca, e são normalmente feitas pelos programadores na fase dos testes de unidades



Quarta Dimensão: Onde será o teste (“o ambiente de testes”):

- Teste de Aplicações Mainframe
- Teste de Aplicações Client
- Teste de Aplicações Server
- Teste de Aplicações Network
- Teste de Aplicações Web

2.5. Métricas de Teste

Por que medir o teste e a automação de teste?

Duas razões importantes: ROI e melhoria da qualidade do produto final.

O que podemos medir?

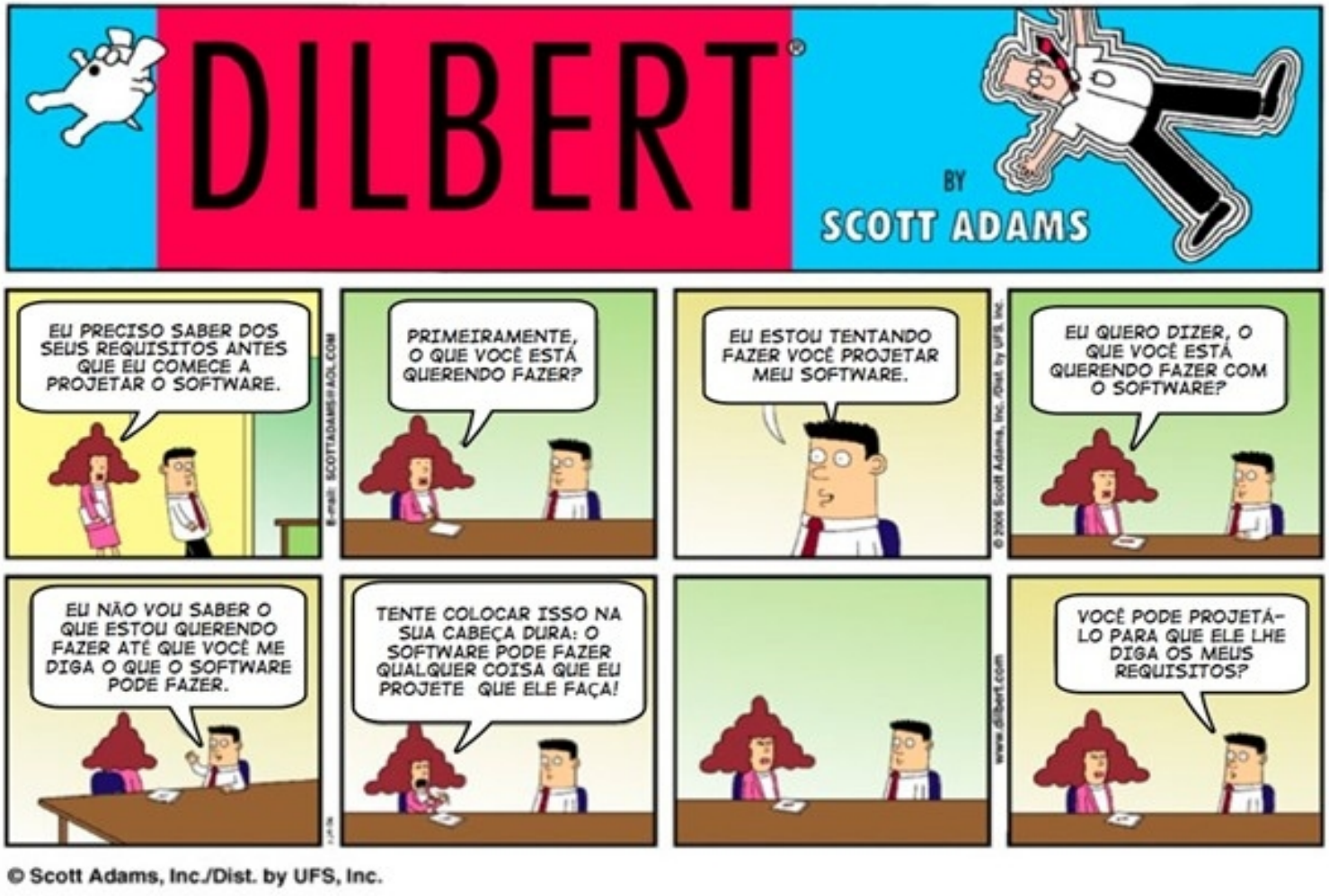
O teste, o momento e si, a automação de teste e o processo.

Métricas de Teste – Atributos:

- **Métricas de "ausência de eficiência":** voltadas para a análise de defeitos encontrados no processo.
  - **Percentual de defeitos encontrados:** Total de defeitos no teste / total de defeitos conhecidos.
  - **Percentual de defeitos resolvidos:** Total de defeitos encontrados antes do release / total de defeitos encontrados.
  - **Métrica de conferência e análise de teste:** refere-se a uma situação em que são encontrados poucos defeitos, devendo-se avaliar o processo de teste.
  - **Métricas de código:** voltadas para estágios de desenvolvimento da aplicação ou onde a quantidade de defeitos encontrada é pouca. Podem ser **subjetivas** (o código foi mesmo todo testado?), ou **objetivas ou de cobertura,** baseadas em percentual (o que foi testado/tudo que deveria ser testado).

Métricas de Teste – Atributos:

- **Métricas de evolução do software:**
  - **Métricas de requerimentos de teste:** acompanhamento e planejamento dos testes em nível gerencial.
  - **Métrica de estimativa de ciclo de teste:** mostra o tempo de teste para executar uma tarefa.


DILBERT
BY
SCOTT ADAMS
EU PRECISO SABER DOS SEUS REQUISITOS ANTES QUE EU COMECE A PROJETAR O SOFTWARE.
PRIMEIRAMENTE, O QUE VOCÊ ESTÁ QUERENDO FAZER?
EU ESTOU TENTANDO FAZER VOCÊ PROJETAR MEU SOFTWARE.
EU QUERO DIZER, O QUE VOCÊ ESTÁ QUERENDO FAZER COM O SOFTWARE?
EU NÃO VOU SABER O QUE ESTOU QUERENDO FAZER ATÉ QUE VOCÊ ME DIGA O QUE O SOFTWARE PODE FAZER.
TENTE COLOCAR ISSO NA SUA CABEÇA DURA: O SOFTWARE PODE FAZER QUALQUER COISA QUE EU PROJETE QUE ELE FAÇA!
VOCÊ PODE PROJETÁ-LO PARA QUE ELE LHE DIGA OS MEUS REQUISITOS?
© Scott Adams, Inc./Dist. by UFS, Inc.

Gerência de Requisitos e Gerência de Configuração – suas implicações no teste de software e Garantia de Qualidade de Software (SQA)

Requisitos, ou requerimentos?

“São as descrições do que o sistema deve fazer, os serviços que deve oferecer e as restrições a seu funcionamento (...) refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como controlar um dispositivo, colocar um pedido ou encontrar informações” (SOMMERVILLE, 2011).

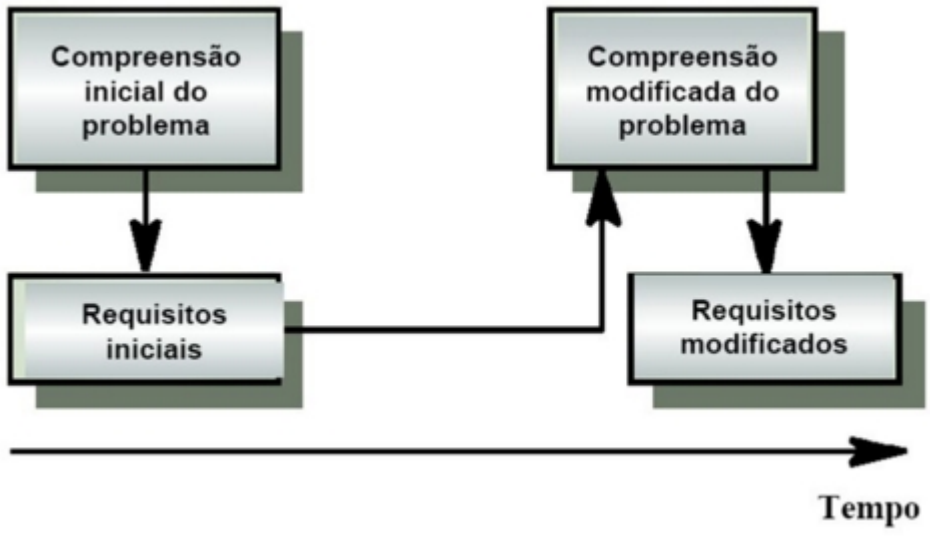
Pra quem quiser acompanhar este estudo: capítulo 4 e 25 do Sommerville (2011), capítulo 5 e 22 do Pressman (2011) e capítulo 3 e 4 do Molinari (2003).

Gerência de Requisitos e Gerência de Configuração – suas implicações no teste de software e Garantia de Qualidade de Software (SQA)

Por que se preocupar tanto com os requisitos?

- **Problemas de escopo??**
- **Problemas de entendimento??**
- **Problemas de volatilidade??**

Tais problemas são tão emblemáticos que são tratados pela Engenharia de Requisitos.



Gerência de Requisitos e Gerência de Configuração – suas implicações no teste de software e Garantia de Qualidade de Software (SQA)

Gerência(mento) de requisitos:

É um conjunto de atividades que ajuda a equipe do projeto a identificar, controlar e acompanhar as necessidades e suas mudanças a qualquer momento enquanto o projeto segue (PRESSMAN, 2011).

É o processo de compreensão e controle das mudanças nos requisitos do sistema, que deve começar assim que uma versão preliminar do documento de requisitos estiver disponível (SOMMERVILLE, 2011).

Gerência(mento) de requisitos:

- **Planejamento de gerenciamento de requisitos:** é o primeiro estágio essencial no processo, e determina o nível de detalhamento requerido, em que deve-se decidir:
  - Como cada requisito deve ser identificado unicamente para poder ser rastreado e comparado;
  - Processo de gerenciamento de mudanças: impacto e custo das mudanças;
  - Políticas de rastreabilidade: relacionamentos entre cada requisito e como registrar os requisitos;
  - Ferramenta de apoio: o que utilizar nesse processo, como softwares específicos ou planilhas e Bds.

Gerência(mento) de requisitos:

- **Gerenciamento de mudança de requisitos:** é o gerenciamento necessário para gerir todas as mudanças propostas aos requisitos de um sistema, a partir da aprovação do documento de requisitos. Descrito pelos estágios:
  - Análise de problema e especificação de mudanças;
  - Análise de mudanças e custos;
  - Implementação de mudanças.

Gerência(mento) de requisitos:

- **Rastreabilidade dos Requisitos:** conjunto de atividades que permita a rápida recuperação dos artefatos associados à um requisito, bem como recuperar as ligações a outros requisitos.

  Matriz de rastreabilidade:

	Requisito 1	Requisito 2	Requisito 3	Requisito 4
Requisito 1		X		
Requisito 2			X	X
Requisito 3	X			X
Requisito 4				

Permite, por exemplo, rastrear a ligação entre requisitos derivadores e requisitos derivados.

Gerência(mento) de configuração (GCS, ou CM – Configuration Management):

"Se a história de uma civilização possuir registros, erros passados podem ser evitados no futuro. Se você possuir a história de um software, erros do passado podem ser consertados e boas práticas podem ser retomadas." Ditado de Gerência de Configuração.

- GCS é a arte de identificar, organizar e controlar modificações no software que está sendo construído por uma equipe de desenvolvimento.
- Objetivo: maximizar a produção, minimizando os erros.
- Base da engenharia de software "moderna", usada para controlar o desenvolvimento do produto e suas versões.

Elementos (atividades) da Gerência de configuração (GCS):

- Gerenciamento de mudanças
- Gerenciamento de versões
- Construção do sistema
- Gerenciamento de *releases*

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de mudanças**
  A mudança é uma realidade para grandes sistemas. Ex.:
  - Alteração de requisitos organizacionais e regras de negócios;
  - Reparação de bugs que exigem modificações sistêmicas;
  - Alterações no ambiente de execução.

Para garantir que as mudanças sejam aplicadas ao sistema de forma controlada, é necessário um conjunto de processos de gerenciamento de mudanças, apoiado por ferramentas.

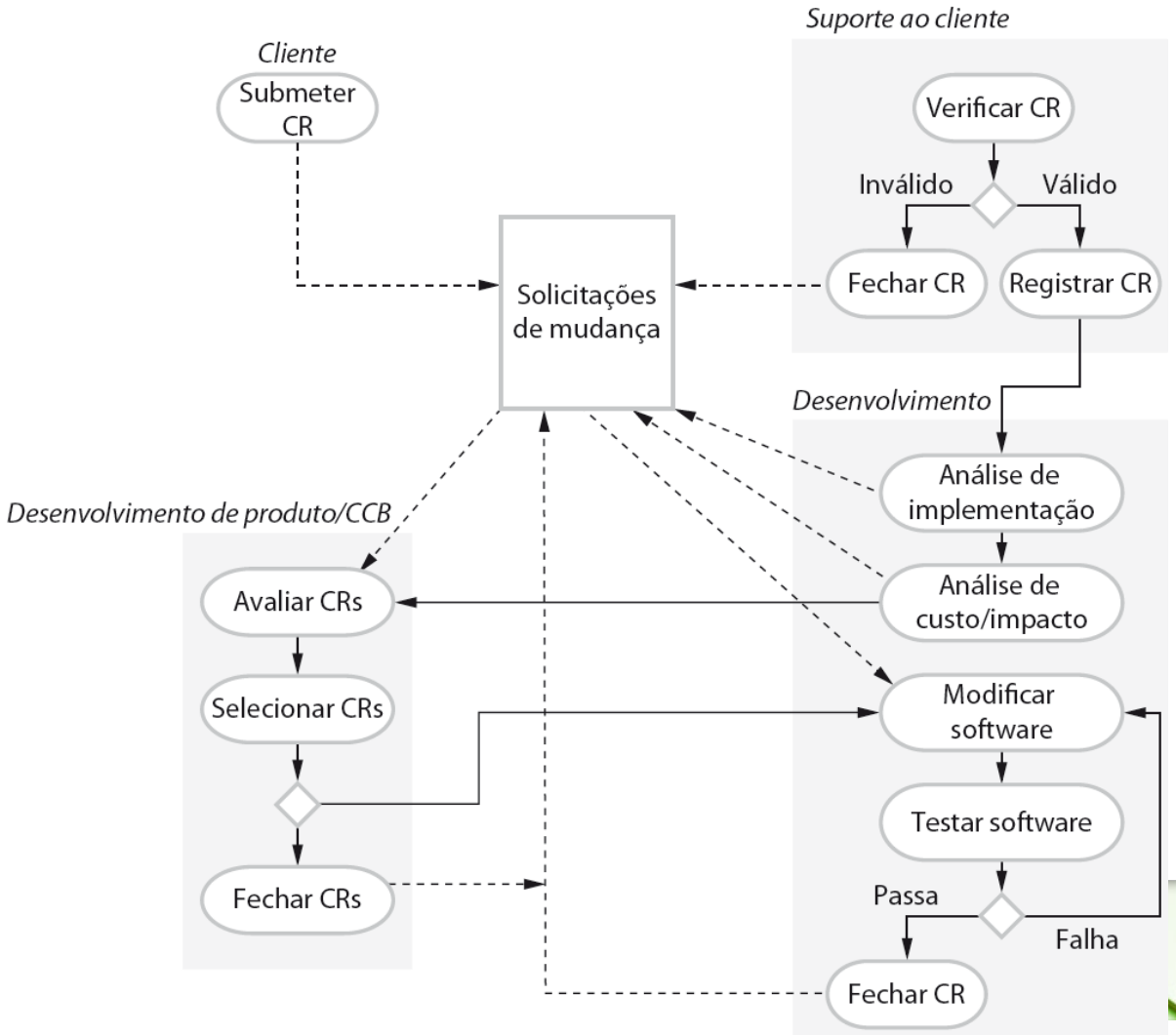
O GM destina-se a garantir que a evolução do sistema seja um processo gerenciado e que seja dada prioridade às mudanças mais urgentes e efetivas.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de mudanças**
  Está relacionado com a análise de custos e benefícios das mudanças propostas.
  - inicia-se com o preenchimento da solicitação de mudança (CR – *Change Request*), que gera o formulário de solicitação de alteração (CRF – *Change Request Form*).
  - A medida que é processada a solicitação de mudança, informação é adicionada ao CRF a fim de registrar as decisões tomadas em cada estágio.
  - O CRF registra as recomendações relativas à mudança, a estimativa de custos e datas da solicitação, aprovação, implementação e validação. Pode conter seção com descrições de implementação feitas pelo desenvolvedor.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de mudanças**



Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de mudanças**
  O grupo de desenvolvimento de produto considera o impacto da mudança:
  - As consequências de não fazer a mudança.
  - Os benefícios da mudança.
  - O número de usuários afetados pela mudança (ou não).
  - Os custos de se fazer a mudança.
  - O ciclo de release de produto (próximo do lançamento de outro release?).

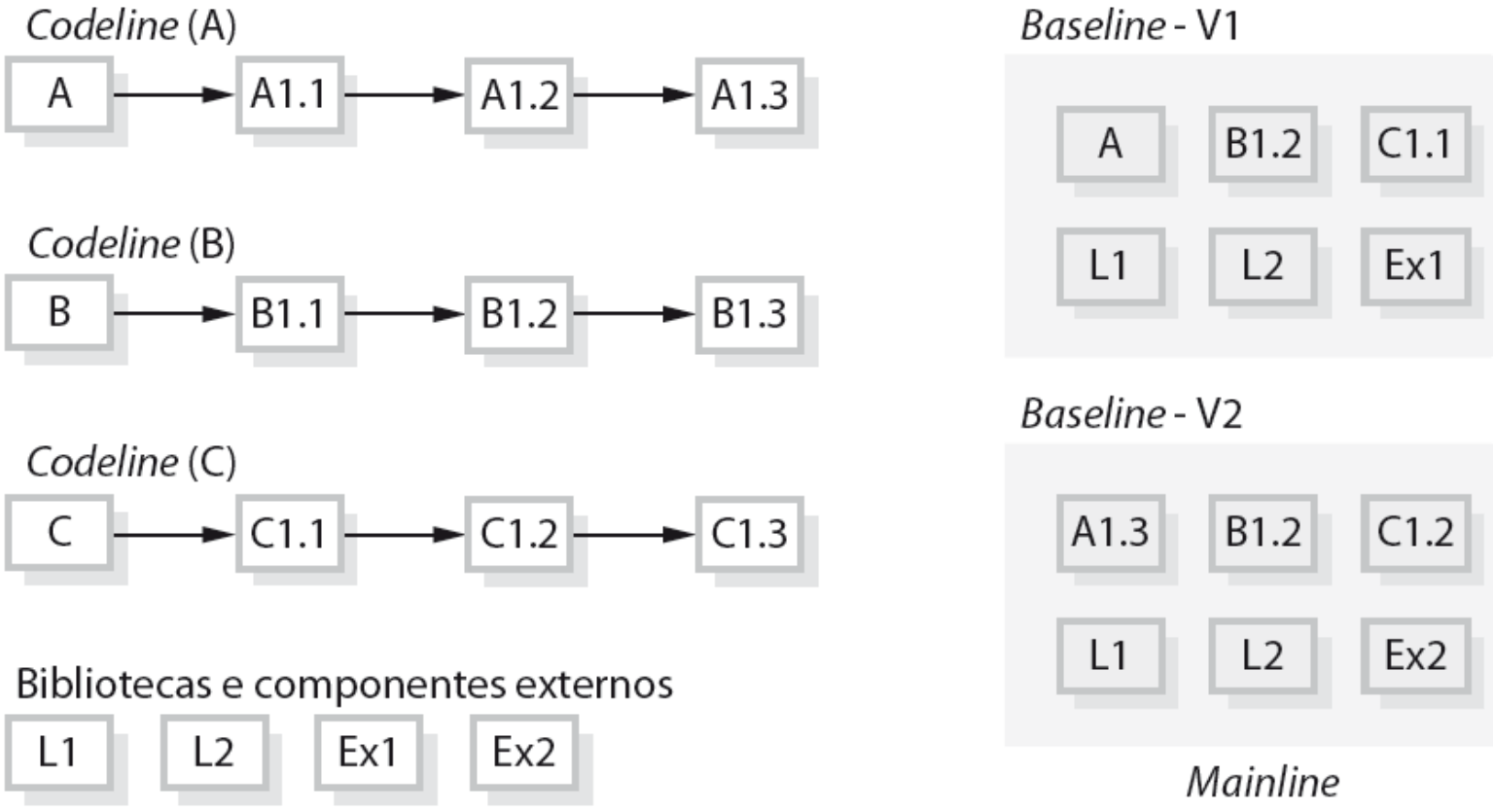
  Uma vez que a equipe de desenvolvimento mude os componentes de software, eles devem manter um registro das mudanças feitas para cada componente: história de derivação de componente.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de versões (controle de versões)**
  VM – do inglês *version management* é o processo de acompanhamento de diferentes versões de componentes de software ou itens de configuração e os sistemas em que esses componentes são usados.
- Também envolve a garantia de que as mudanças feitas por diferentes desenvolvedores para essas versões não interfiram umas nas outras.
- Processo de gerenciamento de *codelines (versões de códigos)* e *baselines (versões de sistemas inteiros).*

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de versões (controle de versões)**

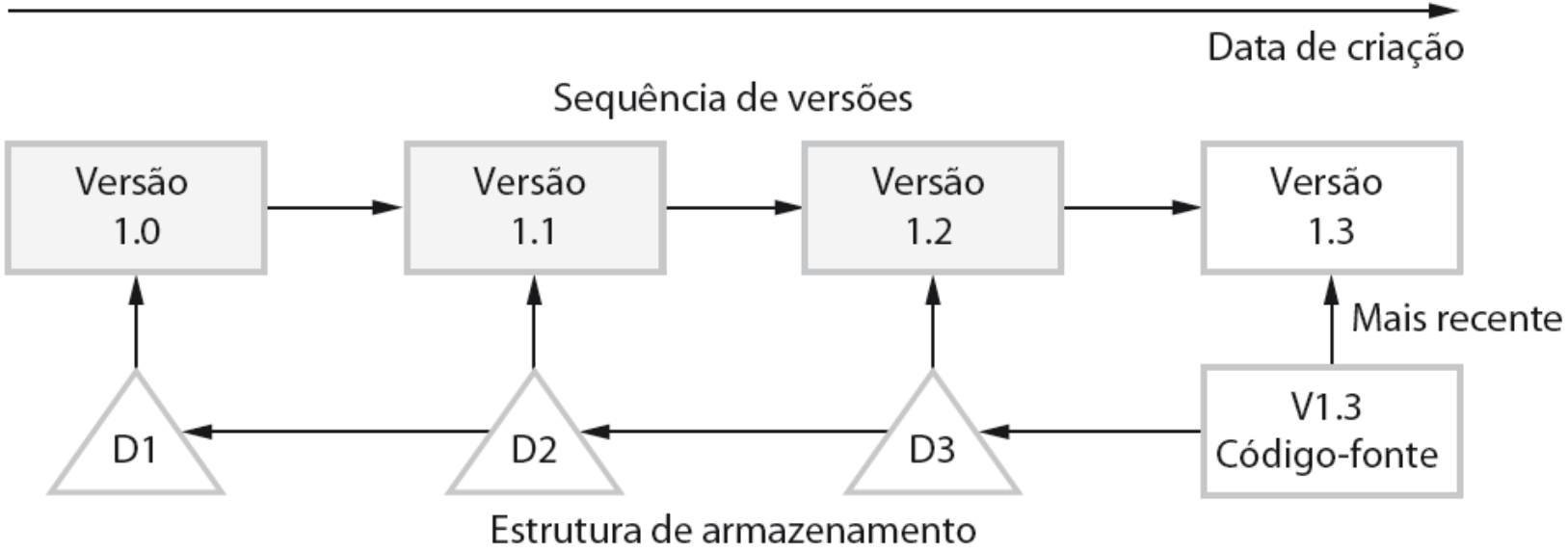


Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de versões (controle de versões)**

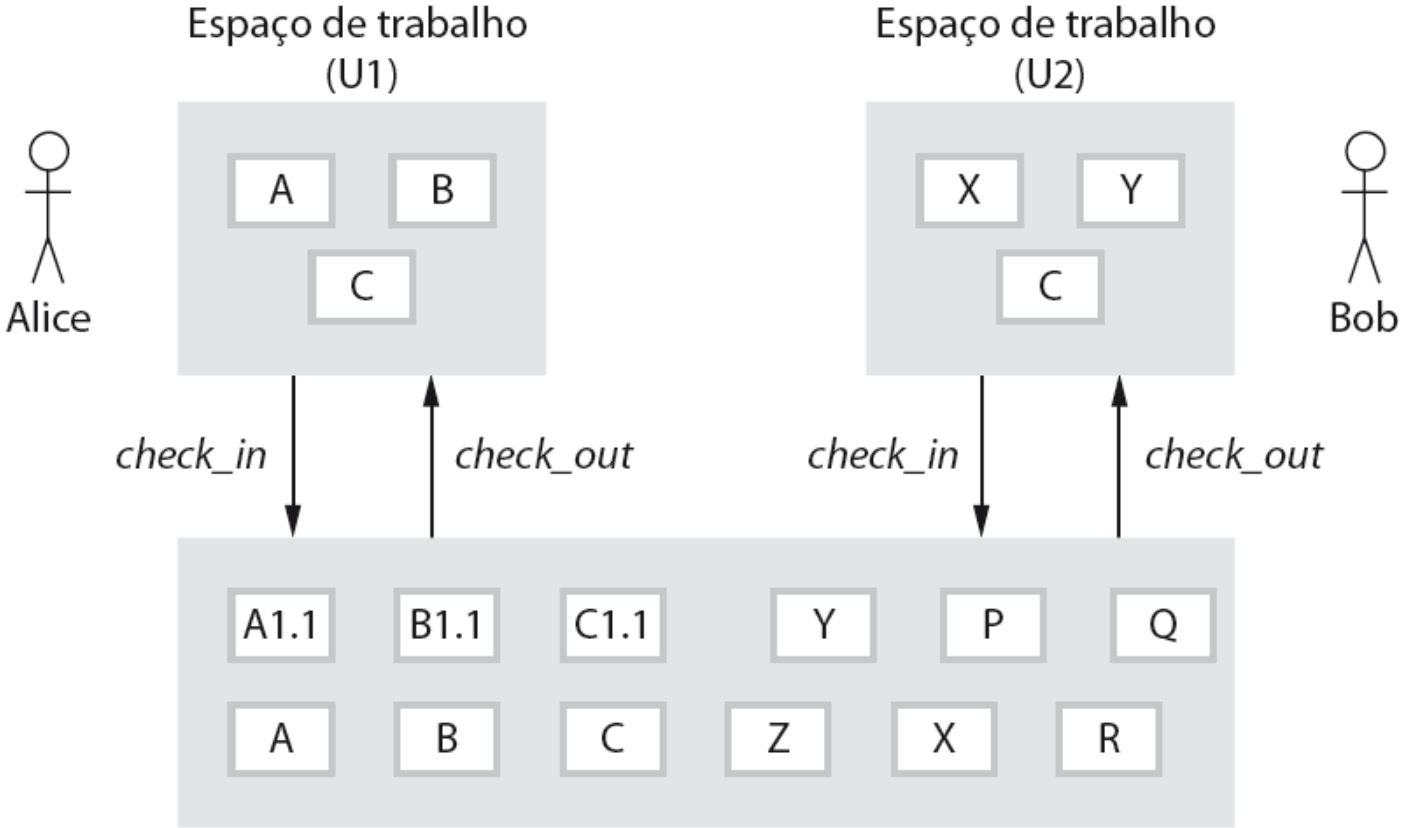
Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de versões (controle de versões)**



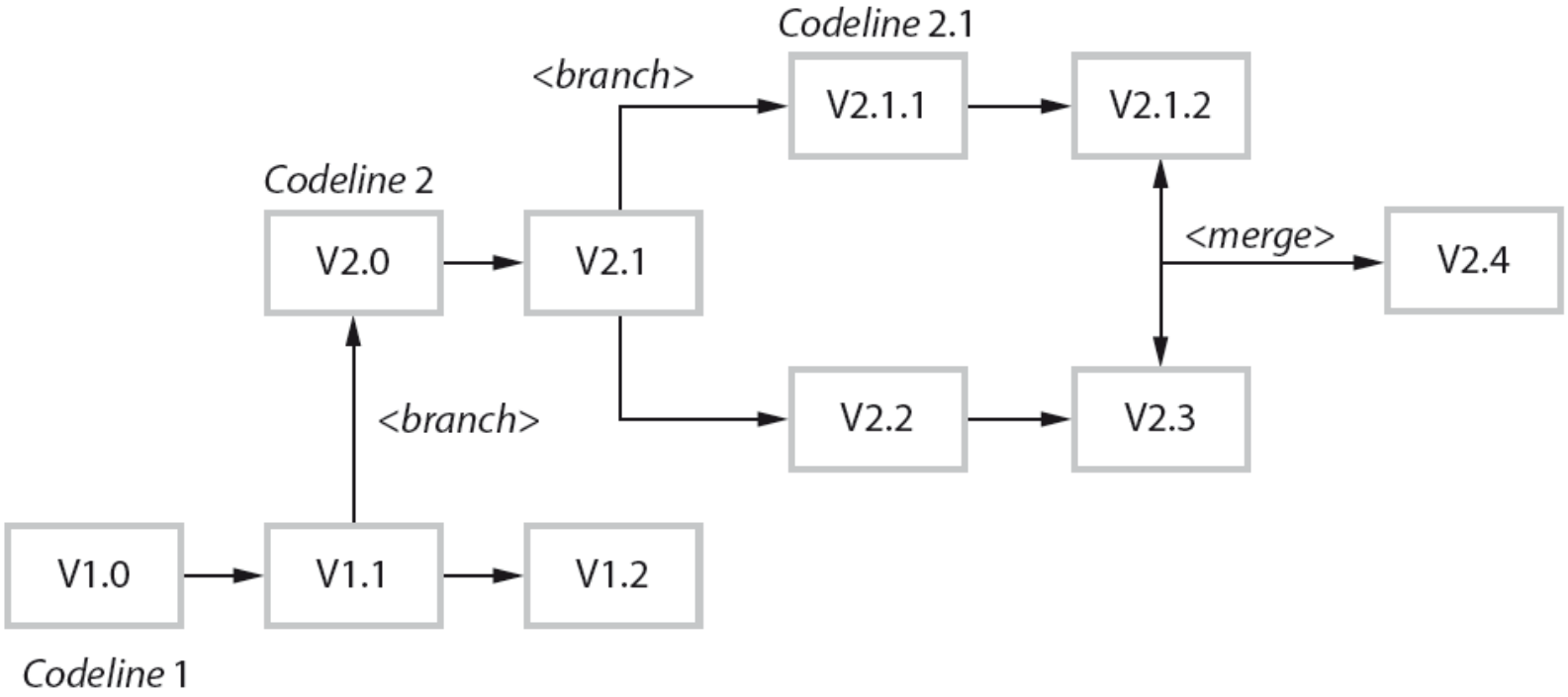
Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de versões: check in e check out a partir de um repositório de versões.**



Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de versões: ramificação de *codelines*.**



Elementos (atividades) da Gerência de configuração (GCS):

- **Construção do sistema**

Construção de sistemas é o processo de criação de um sistema completo e executável por compilar e ligar os componentes do sistema, bibliotecas externas, arquivos de configuração, etc.

Ferramentas de construção de sistema e ferramentas de gerenciamento de versões devem se comunicar com o processo de construção já que envolve a realização de *check-out* de versões de componentes do repositório gerenciado pelo sistema de gerenciamento de versões.

A descrição de configuração usada para identificar uma *baseline* também é usada pela ferramenta de construção do sistemas.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de *releases***

Um release de um sistema é uma versão de um sistema de software distribuído aos clientes.

Para um software de mercado, normalmente é possível identificar dois tipos de release: releases grandes que proporcionam nova funcionalidade importante, e releases menores, que reparam bugs e solucionam os problemas dos clientes.

Para softwares customizados ou linhas de produto de software, os releases do sistema podem ter que ser produzidos para cada cliente e clientes individuais podem estar executando várias versões diferentes do sistema, ao mesmo tempo.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de *releases: acompanhamento de releases***

No caso de um problema, pode ser necessário reproduzir exatamente o software que foi entregue para um cliente particular.

Quando é produzida uma versão do sistema, essa deve ser documentada para assegurar que possa ser recriada no futuro.

Isso é particularmente importante para sistemas embutidos e customizados, de longa vida útil, tais como os que controlam máquinas complexas.

Os clientes podem usar um único release desses sistemas por muitos anos e podem exigir mudanças específicas para um sistema de software especial muito tempo depois da data do release original.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de *releases: documentação de releases***

Para um documento de release, você precisa gravar as versões específicas dos componentes do código-fonte que foram usados para criar o código executável.

Você deve manter cópias dos arquivos de código-fonte, executáveis correspondentes e todos os dados e arquivos de configuração.

Você também deve gravar as versões do sistema operacional, bibliotecas, compiladores e outras ferramentas usadas para construir o software.

Elementos (atividades) da Gerência de configuração (GCS):

- **Gerenciamento de *releases: planejamento de releases***

Bem como o trabalho técnico envolvido na preparação e distribuição de um release, deve-se preparar material de publicidade e divulgação além de estratégias de marketing para convencer os clientes a comprarem o novo release do sistema.

Calendário de release

- Se os releases são muito frequentes ou exigem atualizações do hardware, os clientes podem não mudar para o novo release, especialmente se tiverem que pagar por isso.
- Se os releases do sistema são muito pouco frequentes, pode se perder parte do mercado pois os clientes mudam para sistemas alternativos.

Referências

Myres , G. F. “The Art of Software Testing”. Ed. John Wiley & Sons, Inc. New Jersey, 2004.

Dijkstra, E. W. "The Humble Programmer". Communications of the ACM 15 (10): 859–866, 1972.

Rios, Emerson. “Teste de software”. Alta Books Editora, 2006.

Gelperin, David, and Bill Hetzel. "The growth of software testing." Communications of the ACM 31.6 (1988): 687-695.

MOLINARI, Leonardo. “Testes de software: produzindo sistemas melhores e mais confiáveis: qualidade de software: soluções, técnicas e métodos”. Érica, 2003.

IEEE . Standard for Soft ware & System. Test Documentation. IEEE 829-2008. New York: IEEE, 2008.

PRESSMAN, Roger S. Engenharia de software: uma abordagem profissional. 7ª Edição. Ed: McGraw Hill, 2011.

SOMMERVILLE, Ian,. Engenharia de Software. São Paulo: Pearson, 9ª edição, 2011.