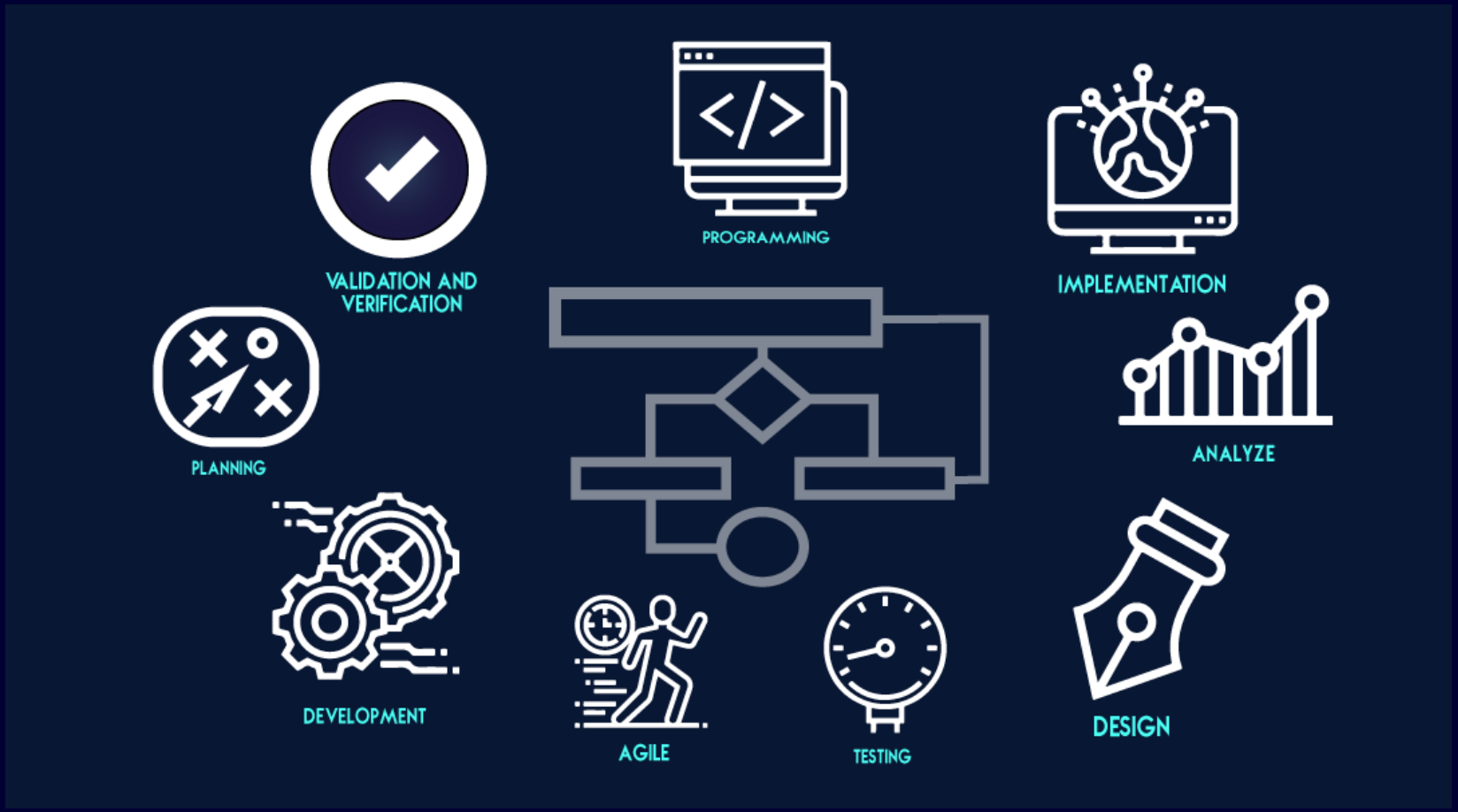
Unichristus
Centro Universitário Christus



Engenharia de Software: Introdução

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: Introdução
5. - Mitos e Realidade
6. - Tipos de Sistemas
7. - Ética
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

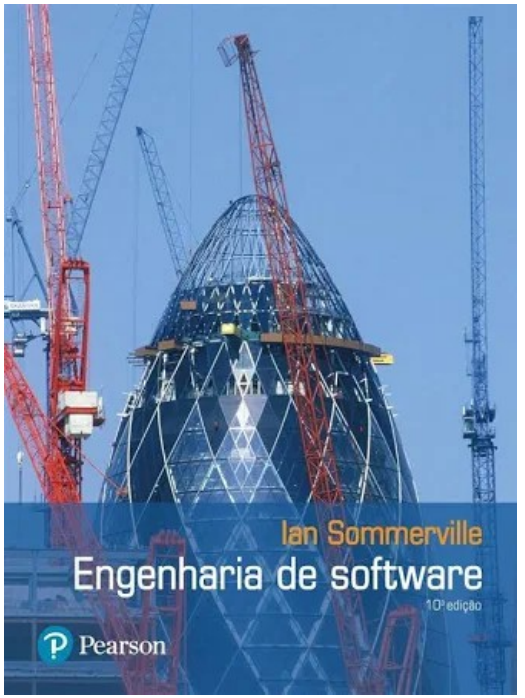
Apresentação

DOCÊNCIA

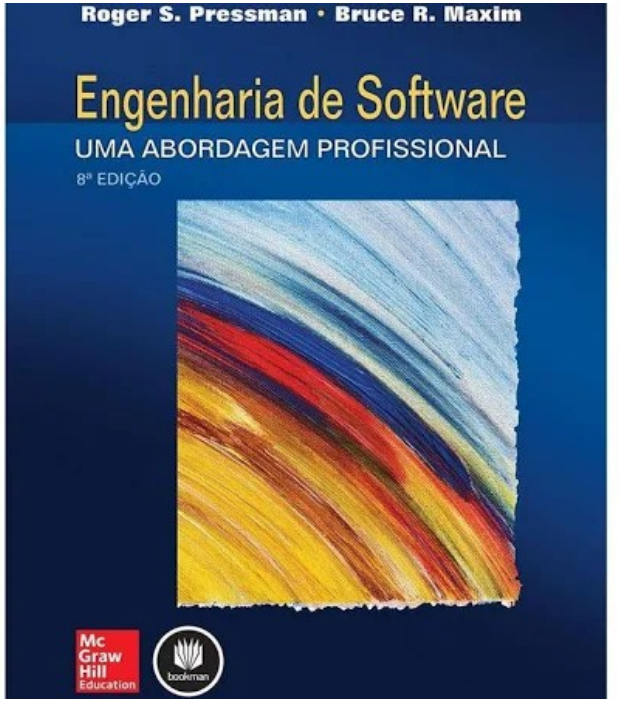
- Professor Substituto das Disciplinas de Sistemas de Informação – FA7 (2011 - 2012)
- Professor da Especialização em Sistemas WEB – FJN (2011 - 2012)
- Professor de Bancas de graduação em Sistemas de Informações – FA7 (2012)
- Professor dos Cursos de Tecnologia da Informação da Unifanor (2015 – 2018)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



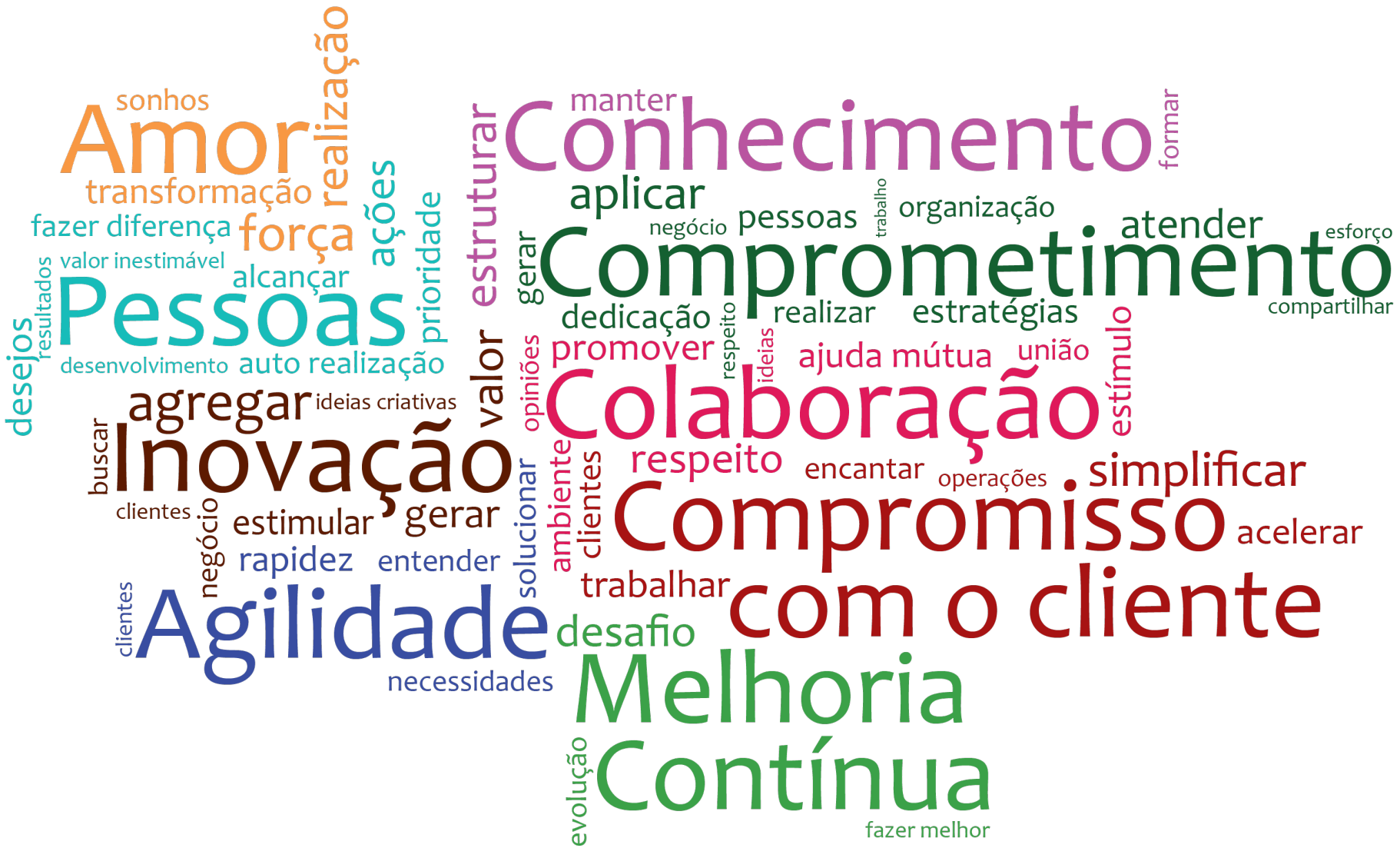
Dicas de Convivência

- Horários
- Conversas
- Dúvidas
- Celular
- Avaliações

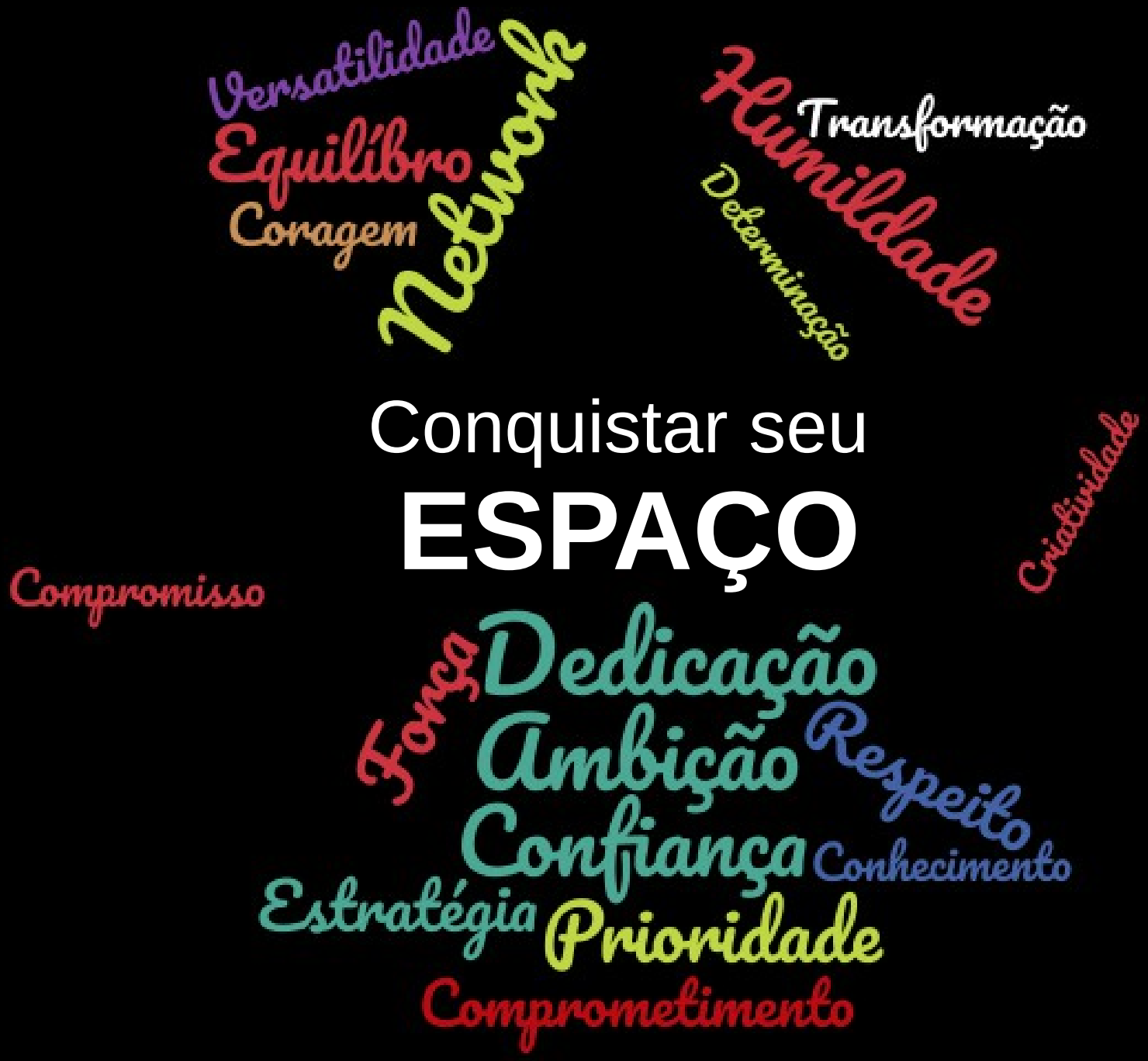
Questionamentos

Bem vindo!!

Princípios e Valores



Princípios e Valores

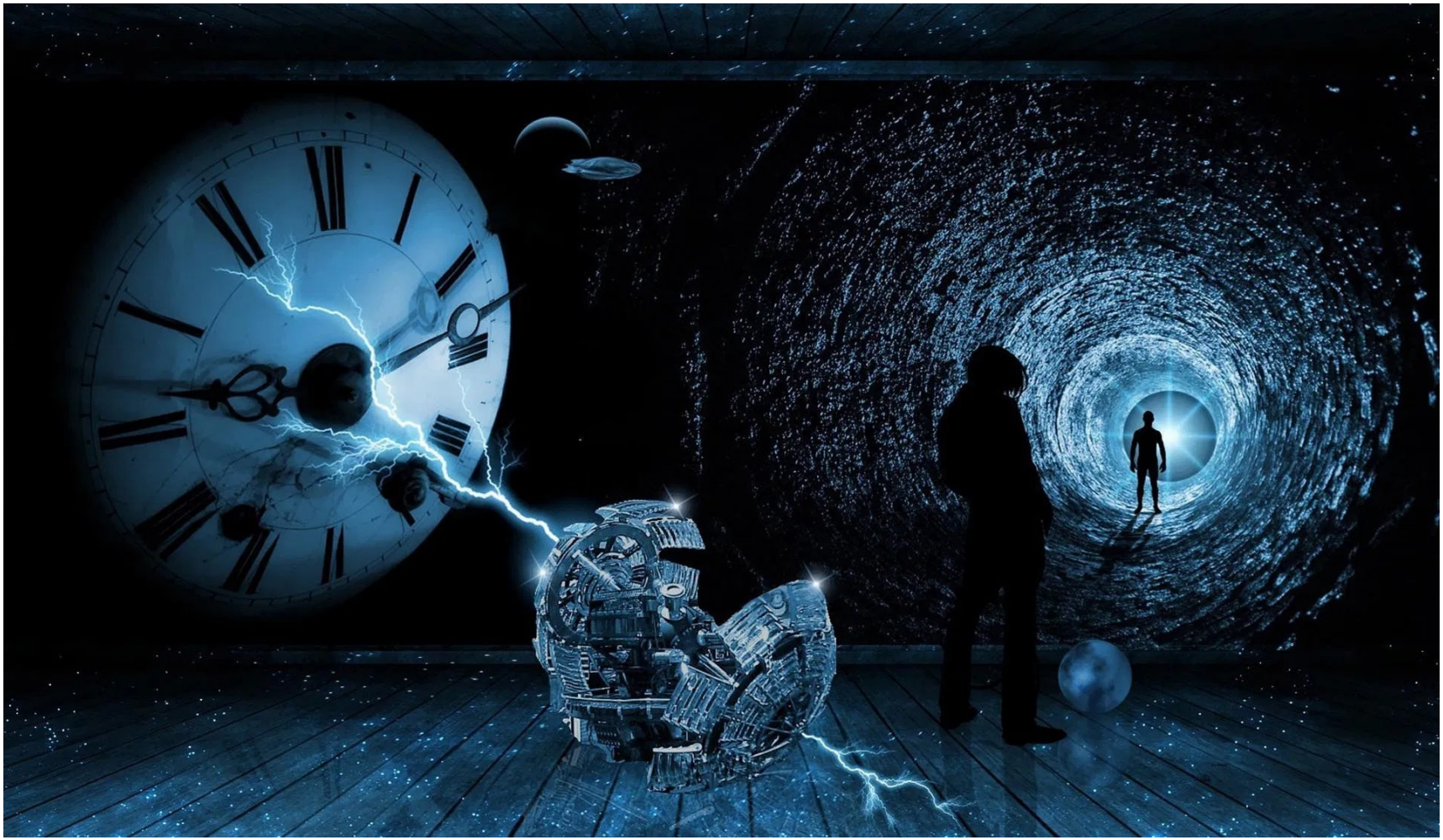

Versatilidade
Equilíbrio
Coragem
Networking
Humildade
Transformação
Determinação
Conquistar seu
ESPAÇO
Compromisso
Criatividade
Força
Dedicação
Ambição
Respeito
Confiança
Conhecimento
Estratégia
Prioridade
Comprometimento

Que cadeira você deseja sentar hoje?

Tópicos apresentados

- Desenvolvimento profissional de software.
- O que se entende por engenharia de software.
- Ética na engenharia de software.
- Uma breve introdução a questões éticas que afetam a engenharia de software.
- Uma introdução aos três exemplos utilizados nos capítulos posteriores do livro.

Vamos fazer uma viagem no tempo?



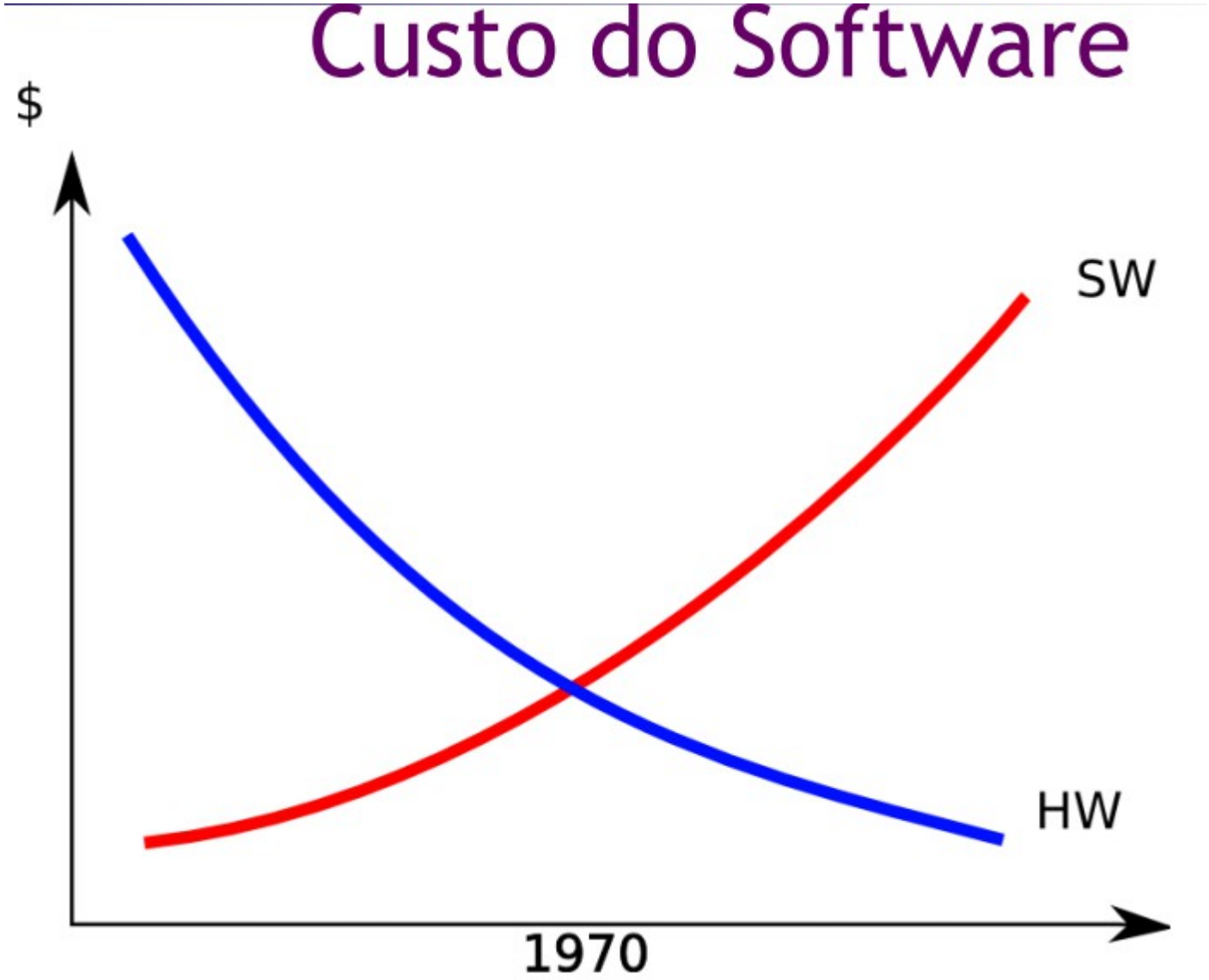
https://www.timetoast.com/timelines/historia-da-engenharia-de-software

"O mundo de hoje não poderia viver sem o software"

Sommerville, 2011

- Lei das "**consequências não pretendidas**" (Pressman, 2006)
  - Ex: isopor, penicilina, microondas, facebook...
  - Na década de 50 não era esperado que o software tomasse as proporções atuais
  - Quais as consequências disso?

- Problemas com o software:
  - Software que não funciona
  - Altos custos
  - Prazos não cumpridos
  - Baixa qualidade:
    - cheios de defeitos
    - confiabilidade duvidosa
    - difíceis de usar
    - lentos,
    - não portáveis,
    - etc...


Custo do Software
$
SW
HW
1970

Bugs históricos

- **Mariner I – 1962**
  - Missão observar planeta Vênus
  - Fórmula matemática foi equivocadamente transcrita para o computador
  - Desviou do curso e foi destruído 4 min após lançamento
  - Prejuízo: US$ 18,5 mi

Bugs históricos

- **Divisão de pontos flutuantes nos processadores Pentium da Intel – 1993**
  - Erro em divisões dentro de uma faixa de números (erro ~0,006% no arrendondamento)
  - 3 a 5 milhões de peças com defeito
  - *Recall* para todos que quiseram trocar
  - Custou à Intel US$ 475 milhões

$$\frac{4195835}{3145727} = 1.333820449136241002$$

$$\frac{4195835}{3145727} = 1.333739068902037589$$

Bugs históricos

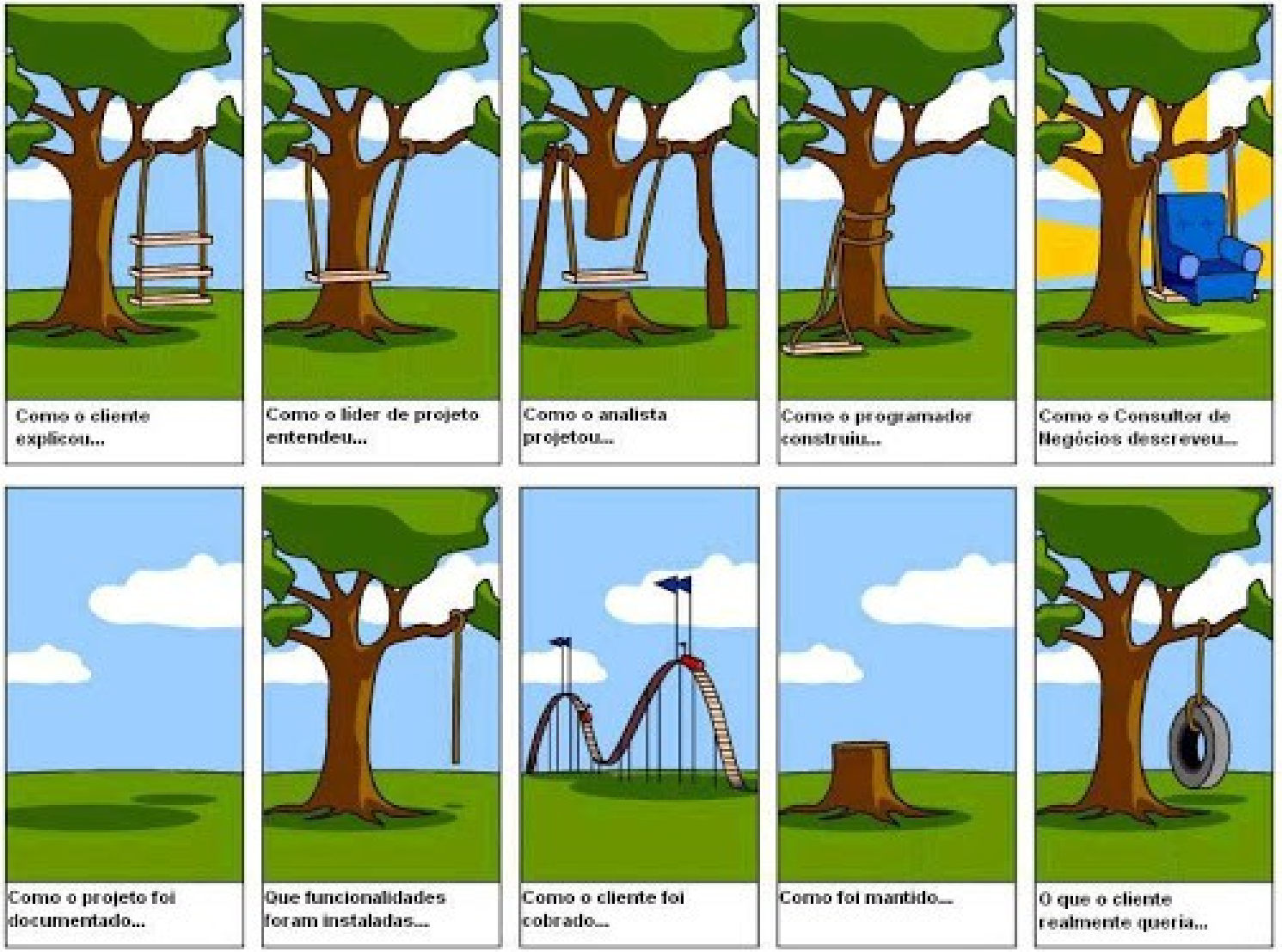
- **Ariane 5 vôo 501 – 1996**
  - Levou uma década de desenvolvimento e custou 7 bilhões de dólares.
  - Foguete com código reutilizado do Ariane 4 (outro hardware);
  - *Overflow* de inteiro: conversão de *float* de 64-bits para inteiro 16-bits com sinal;
  - O processador primário do foguete sobrecarregou os motores que se desintegraram em 40 segundos;
  - Não tripulado (sem vítimas); prejuízo de US$ 370 milhões

Bugs históricos

- **Bug do milênio (Y2K) – 2000**
  - Datas com apenas 2 dígitos para o ano
  - Uma das maiores histerias da história
  - Ao virar o ano 2000, a preocupação era que contasse como 1900
  - Entre US$ 300 e US$ 500 bi no mundo todo

Porque tantos bugs ocorrem?

- Falta de testes?
- Projetos mal feitos?
- Falta de controle de qualidade?
- O cliente não sabe o que quer?
- Os desenvolvedores não entendem o que o usuário quer?
- Etc. etc etc


Como o cliente explicou...
Como o líder de projeto entendeu...
Como o analista projetou...
Como o programador construiu...
Como o Consultor de Negócios descreveu...
Como o projeto foi documentado...
Que funcionalidades foram instaladas...
Como o cliente foi cobrado...
Como foi mantido...
O que o cliente realmente queria...

Engenharia de Software

Conjunto de **processos**, **métodos**, **técnicas** e **ferramentas** que ajudam a produzir software com maior qualidade e com menor custo.

Incluem **processo de desenvolvimento** e atividades de **gerenciamento de projetos**

Alguns marcos importantes da Engenharia de Software

- 1950 a 1968: primeiros tempos: desenvolvimento ad hoc
- 1968: Crise do software
  - Termo Engenharia de Software oficialmente definido
- 1970: Modelo Cascata (Royce)
- 1974: Análise/projeto estruturado (Stevens, Myers, Constantine, de Marco e Yourdon)
- 1975: O mítico homem-mês (Fred Brooks)
- 1975: 1st National Conference on Software Engineering - tornou-se Internacional em 1976 (**ICSE**)
- 1987: 1o Simpósio Brasileiro de Engenharia de Software

Alguns marcos importantes da Engenharia de Software

- 1980 a 2000:
  - desenvolvimento das várias áreas de ES (por exemplo, planejamento de projeto, estimativas de software (COCOMO), gerência de riscos, modelos de processo e modelos de qualidade, teste de software, ferramentas e ambientes de desenvolvimento, etc)
- Fim dos anos 80: surgimento da análise orientada a objetos
- 1997: UML (Unified Modelling Language)
- 2001: O manifesto ágil
- 2002 em diante: vários processos ágeis, foco na interação com o cliente e facilidade de evolução
- 2017: 39th International Conference on Software Engineering
- 2017: 31o Simpósio Brasileiro de Engenharia de Software (dentro do VIII CBSOFT)

Evolução do Software

Crise do Software

- Termo crise de software usado por muito tempo para se referir aos problemas do software não acompanhar o desenvolvimento do hardware
- Reconhecimento da falta de técnicas e processos adequados para o desenvolvimento de software de qualidade
- Afirmação de Djikstra (+-1970): especificações de software complexas e obscuras, que tornam o programador “as mais infelizes criaturas da face da Terra”.

- As economias de todas as nações desenvolvidas são dependentes de softwares.
- Mais e mais sistemas são controlados por software.
- A engenharia de software se preocupa com teorias, métodos e ferramentas para desenvolvimento de softwares profissionais.
- As despesas de software representam uma fração significativa do PIB em todos os países desenvolvidos.

Custos de Softwares

- Os custos de software geralmente dominam os custos do sistema de computador.
- Em um PC, geralmente, os custos de software são maiores que os custos do hardware.
- Custa mais para se manter um software do que para desenvolvê-lo.
- Para sistemas com uma vida longa, os custos de manutenção podem ser várias vezes os custos do desenvolvimento.
- A engenharia de software está preocupada com que o desenvolvimento de software seja custo efetivo.

Produtos de Softwares

- Produtos genéricos
  - ü Sistemas autônomos que são comercializados e vendidos a qualquer cliente que deseja comprá-los. Exemplos - Softwares para PC tais como programas gráficos, ferramentas de gerenciamento de projetos; software CAD; software para propósitos específicos, tais como sistemas de registros odontológicos.
- Produtos sob encomenda
  - ü O software que é encomendado por um cliente específico para atender suas próprias necessidades. Exemplos - sistemas de controle integrado, software de controle de tráfego aéreo, sistemas de monitoramento de tráfego.

Especificação de Produtos

- Produtos genéricos
  - ü A especificação do que o software deve fazer é de propriedade do desenvolvedor de software e as decisões sobre as mudanças de software são feitos pelo desenvolvedor.
- Produtos sob encomenda
  - ü A especificação do que o software deve fazer é propriedade do cliente para o software e eles tomam decisões sobre as mudanças necessárias no software.

Perguntas frequentes sobre a engenharia de software

Pergunta	Resposta
O que é software?	Softwares são programas de computador e documentação associada. Produtos de software podem ser desenvolvidos para um cliente específico ou para o mercado em geral.
Quais são os atributos de um bom software?	Um bom software deve prover a funcionalidade e o desempenho requeridos pelo usuário; além disso, deve ser confiável e fácil de manter e usar.
O que é engenharia de software?	É uma disciplina de engenharia que se preocupa com todos os aspectos de produção de software.
Quais são as principais atividades da engenharia de software?	Especificação de software, desenvolvimento de software, validação de software e evolução de software.
Qual a diferença entre engenharia de software e ciência da computação?	Ciência da computação foca a teoria e os fundamentos; engenharia de software preocupa-se com o lado prático do desenvolvimento e entrega de softwares úteis.

Perguntas frequentes sobre a engenharia de software

Pergunta	Resposta
Qual a diferença entre engenharia de software e engenharia de sistemas?	Engenharia de sistemas se preocupa com todos os aspectos do desenvolvimento de sistemas computacionais, incluindo engenharia de hardware, software e processo. Engenharia de software é uma parte específica desse processo mais genérico.
Quais são os principais desafios da engenharia de software?	Lidar com o aumento de diversidade, demandas pela diminuição do tempo para entrega e desenvolvimento de software confiável.
Quais são os custos da engenharia de software?	Aproximadamente 60% dos custos de software são de desenvolvimento; 40% são custos de testes. Para software customizado, os custos de evolução frequentemente superam os custos de desenvolvimento.
Quais são as melhores técnicas e métodos da engenharia de software?	Enquanto todos os projetos de software devem ser gerenciados e desenvolvidos profissionalmente, técnicas diferentes são adequadas para tipos de sistemas diferentes. Por exemplo, jogos devem ser sempre desenvolvidos usando uma série de protótipos, enquanto sistemas de controle críticos de segurança requerem uma especificação analisável e completa. Portanto, não se pode dizer que um método é melhor que outro.
Quais diferenças foram feitas pela Internet na engenharia de software?	A Internet tornou serviços de software disponíveis e possibilitou o desenvolvimento de sistemas altamente distribuídos baseados em serviços. O desenvolvimento de sistemas baseados em Web gerou importantes avanços nas linguagens de programação e reúso de software.

Engenharia de Software

A engenharia de software é uma disciplina da engenharia que se preocupa com todos os aspectos da produção de software desde o início da especificação do sistema até a manutenção do sistema após esse estar sendo usado.

- Disciplina de engenharia

Utiliza teorias e métodos adequados para resolver os problemas tendo em mente as restrições organizacionais e financeiras.

- Todos os aspectos da produção de software

Não se preocupa apenas com o processo técnico de desenvolvimento, mas também com o gerenciamento de projetos e o desenvolvimento de ferramentas, métodos, etc. para dar apoio à produção de software.

A importância da Engenharia de Software

- Cada vez mais, os indivíduos e a sociedade dependem de sistemas de software avançados.
- Precisamos ser capazes de produzir sistemas confiáveis com economia e rapidamente.
- Geralmente, é mais barato, no longo prazo, usar métodos de engenharia de software e técnicas para os sistemas de software em vez de apenas escrever os programas como se fosse um projeto de programação pessoal.
- Para a maioria dos tipos de sistemas, a maior parte dos custos são os custos de alterar o software em uso.

Atividades de processo de sotware

- A especificação de software, onde os clientes e engenheiros definem o software que deve ser produzido e as restrições sobre o seu funcionamento.
- Desenvolvimento de software, em que o software é projetado e programado.
- Validação de software, em que o software é verificado para garantia de atender ao que o cliente necessita.
- Evolução de software, em que o software é modificado para refletir as mudanças de requisitos do cliente e do mercado.

Questões gerais que afetam a maioria dos softwares

- Heterogeneidade

Cada vez mais, os sistemas são necessários para operar como sistemas distribuídos através de redes que incluem diferentes tipos de computadores e dispositivos móveis.

- Mudança de negócio e social

Negócio e sociedade estão mudando com uma rapidez incrível, na medida em que as economias emergentes se desenvolvem e as novas tecnologias se tornam disponíveis. Elas precisam ser capazes de alterar os softwares existentes e desenvolver novos softwares rapidamente.

- Segurança e confiança

Como o software está entrelaçado com todos os aspectos de nossas vidas, é essencial que possamos confiar nele.

Diversidade na engenharia de software

- Existem muitos tipos diferentes de sistemas de software e não existe um conjunto universal de técnicas de software aplicável a todos eles.

- Os métodos de engenharia de software e ferramentas usadas dependem do tipo da aplicação que será desenvolvida, os requisitos do cliente e os antecedentes da equipe de desenvolvimento.

Tipos de Aplicações

Aplicações stand-alone – são aplicações executadas em um computador local, tal como um PC. Incluem toda a funcionalidade necessária e não precisam estar conectadas a uma rede.

Aplicações interativas baseadas em transações – são aplicações executadas em um computador remoto e são acessadas pelos usuários a partir dos seus próprios PCs ou terminais. Essas incluem aplicações web tais como para e-commerce.

Sistemas de controle embutidos – são sistemas de software de controle que controlam e gerenciam dispositivos de hardware. Numericamente, provavelmente existem mais sistemas embutidos do que qualquer outro tipo de sistema.

- *Sistemas de entretenimento* – são, principalmente, para uso pessoal e se destinam a entreter o usuário.

Tipos de Aplicações

5. *Sistemas de processamento de lotes* – são sistemas corporativos projetados para processar dados em grandes lotes. Eles processam um grande número de entradas individuais para criar saídas correspondentes.

6. *Sistemas de modelagem e simulação* – são desenvolvidos por cientistas e engenheiros para modelar processos físicos ou situações, que incluem muitos, objetos separados que interagem entre si.

7. *Sistemas de coleta de dados* – são sistemas que coletam dados do seu ambiente usando um conjunto de sensores e envia esses dados para outros sistemas, para processamento.

8. *Sistemas de sistemas* – são sistemas compostos por uma série de outros sistemas de software.

O que é Sistema?

é um conjunto de elementos inter-

dependentes que realizam

operações visando atingir metas

especificadas.

Sistema de Computação

é aquele destinado ao suporte
ou
automação de tarefas através
de
processamento de
informações.

Componentes de Sistemas de Computação

Hardware	Computadores,periféricos e redes.
Software	Os programas e arquivos de dados.
Usuários	Usuários e operadores que realizam as tarefas e procedimentos.

Componentes de Sistemas de Computação (cont.)

Procedimentos	Atividades realizadas pelos usuários e operadores, bem como pelos programas.
Documentação	Manuais e formulários que descrevem as operações do sistema.

Exemplos de Sistemas Computacionais

- Automação Bancária
- Frequência e Folha de Pagamento
- Controle de Tráfego Urbano
- Controle Acadêmico
- Editoração de Jornais e Revistas
- Controle de Elevadores
- Automação de Biblioteca

O que é Software?

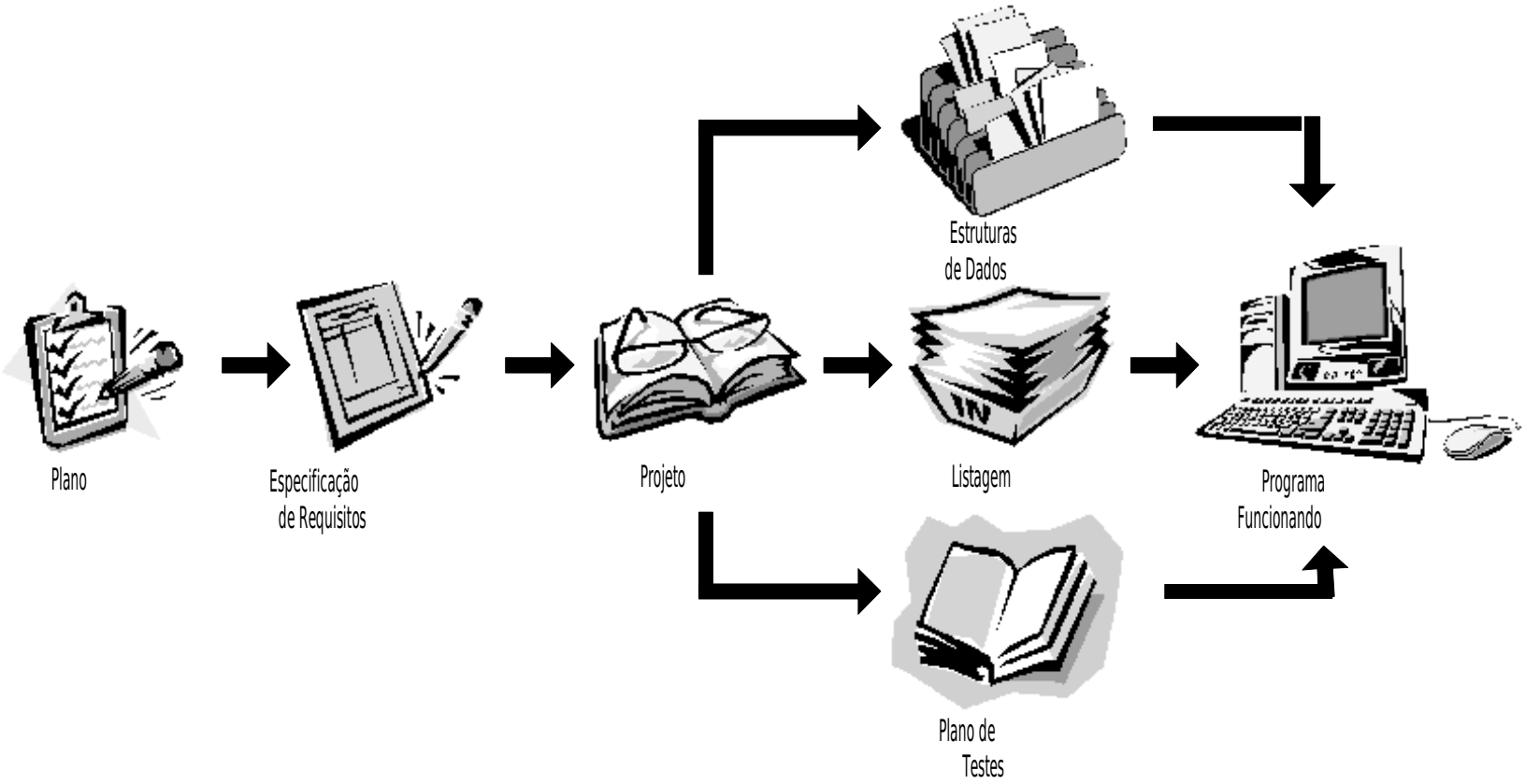
- Programas de computadores associados a documentação.

- É um conjunto de soluções algorítmicas, codificadas numa linguagem de programação, executado numa máquina real.

Tipos de produtos de software

- **Genéricos** (COTS – Commercial Off-The Shelf) - tipo stand-alone, pacotes de software, como por exemplo, processadores de texto, ferramentas de gerenciamento.
- **Sob encomenda** (personalizado) – desenvolvido para um cliente em particular.

Componentes do Software



Características do Software

- Complexidade
- Conformidade
- Mutabilidade

Características do Software

- **Complexidade**

– Software é mais complexo do que qualquer outro produto construídos por seres humanos.

Características do Software

- **Conformidade**

– O software deve ser desenvolvido conforme o ambiente. Não é o ambiente que deve se adaptar ao software.

  - Desenvolvido ou projetado por engenharia, não manufaturado no sentido clássico (industrial).
  - Sucesso é medido pela qualidade e não quantidade.

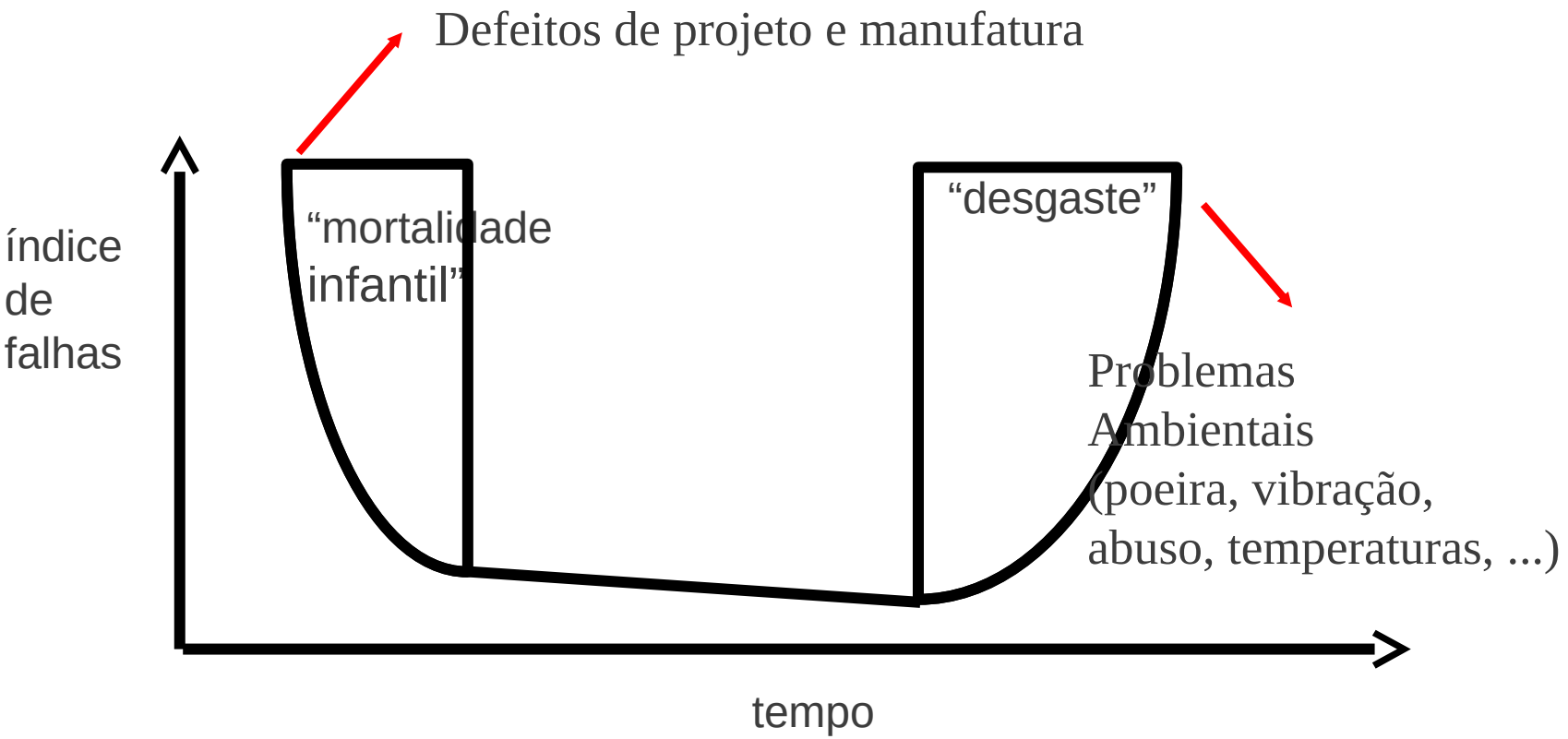
Características do Software

- **Mutabilidade**

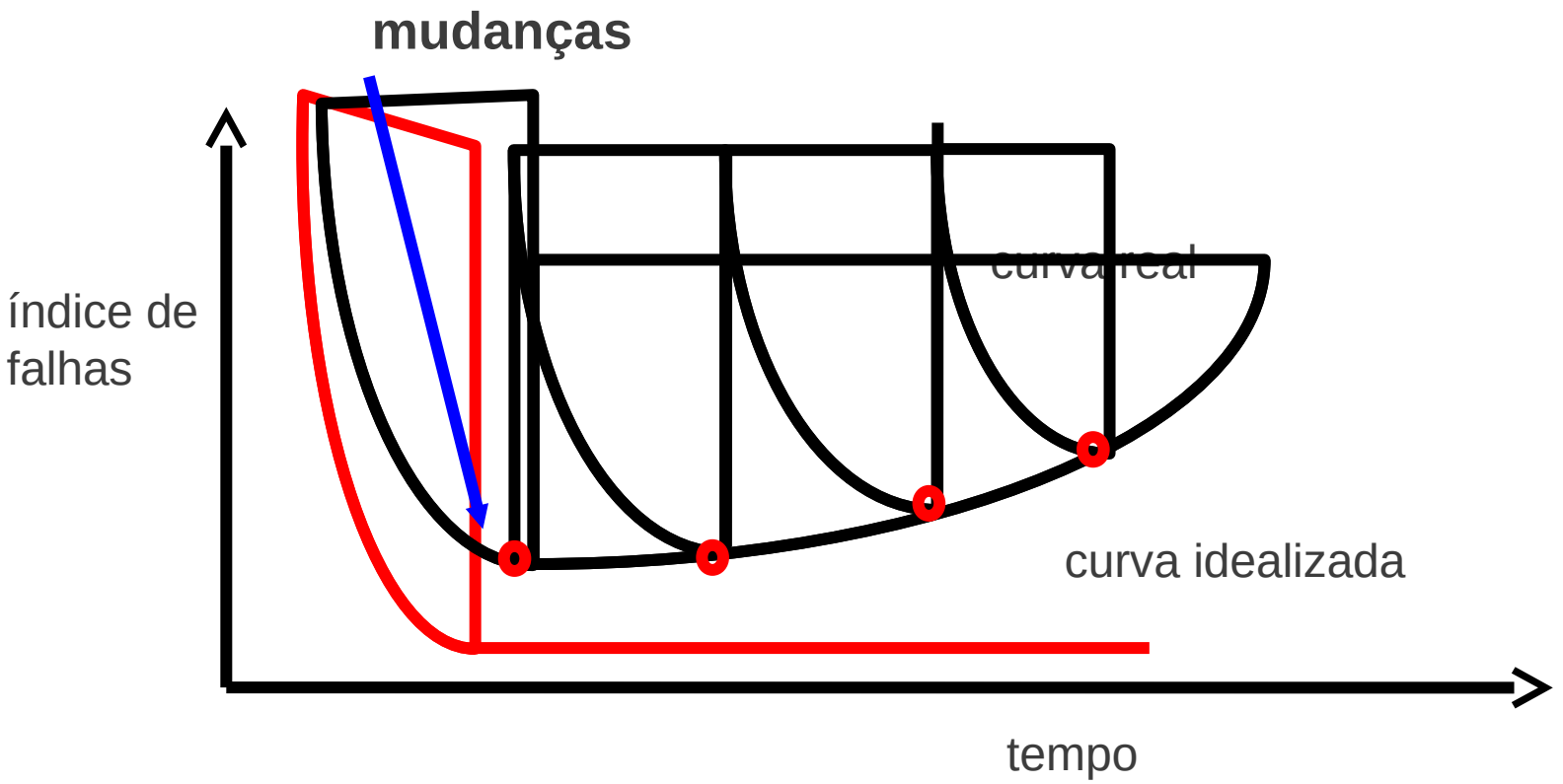
– Existe sempre uma pressão para se fazer mudanças em um software.

  - Não se “desgasta”, mas se deteriora devido as mudanças.
  - A maioria é feita sob medida em vez de ser montada a partir de catálogos de componentes existentes (reusabilidade de software).

Falhas do Hardware



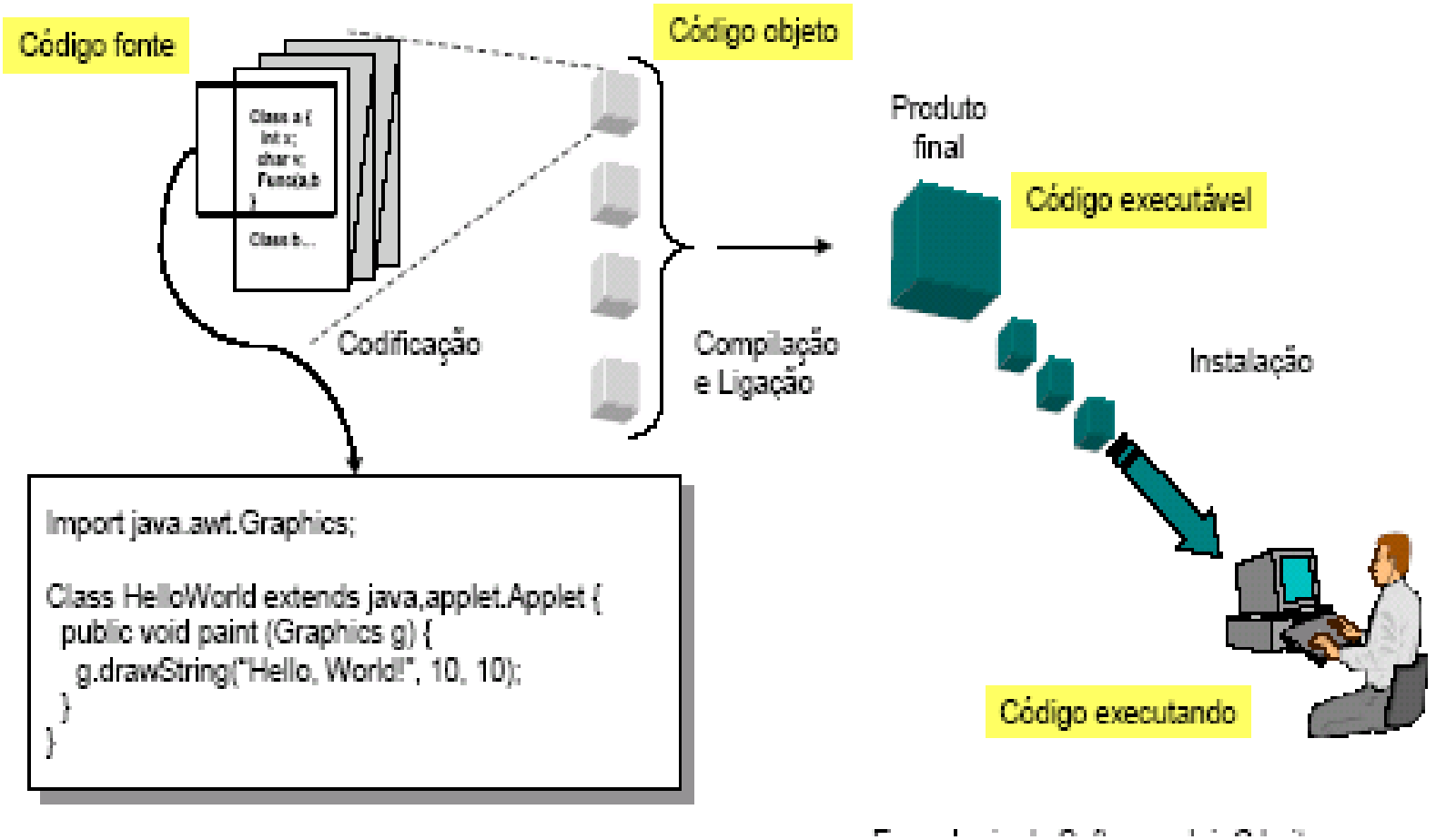
Falhas do Software



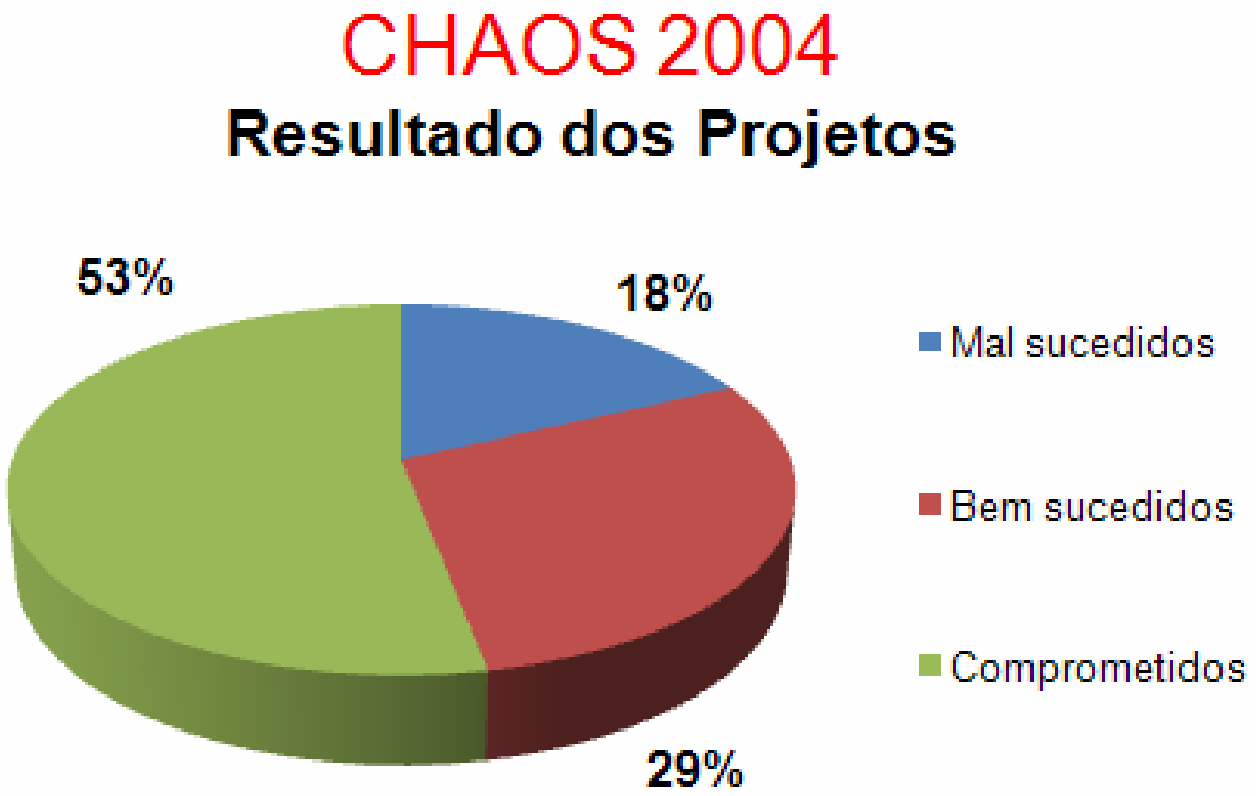
Falhas do Hardware/Software

- Quando um componente de hardware se desgasta é substituído por uma “peça de reposição”
- Não existe “peça de reposição” para software
  - Toda falha indica um erro no projeto ou no processo de tradução para o código executável
  - Manutenção do software é mais complexa do que a do hardware

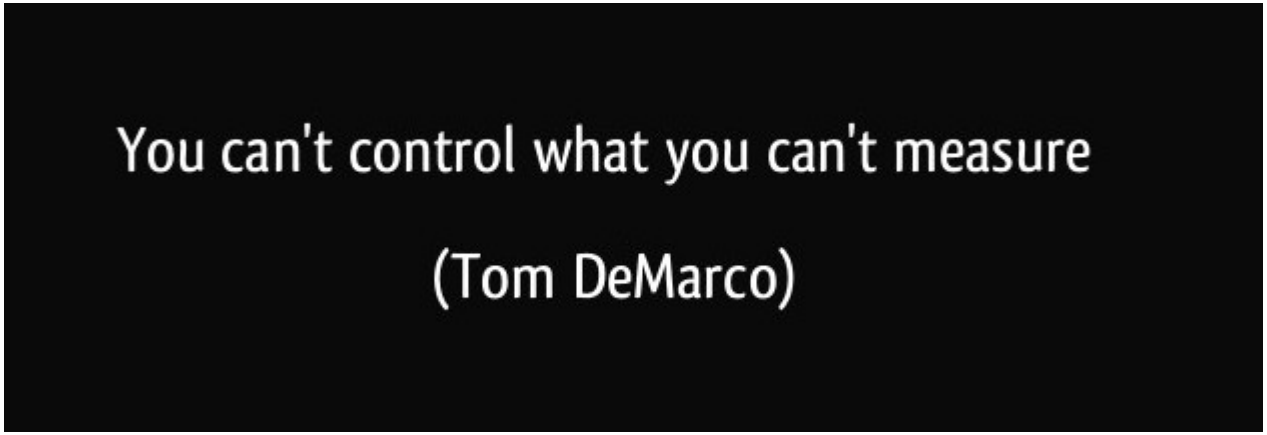
Formas do Software



CRISE DO SOFTWARE



CRISE DO SOFTWARE



Fundamentos de Engenharia de Software

Alguns princípios fundamentais se aplicam a todos os tipos de sistema de software, independentemente das técnicas de desenvolvimento utilizadas:

1. Os sistemas devem ser desenvolvidos através de um processo de desenvolvimento gerenciado e compreendido. Naturalmente, diferentes processos são usados para diferentes tipos de software.

2. Confiança e desempenho são importantes para todos os tipos de sistemas.

3. É importante entender e gerenciar as especificações e requisitos do software (o que o software deve fazer).

4. Quando possível, você deve reusar software que já foi desenvolvido, em vez de escrever um novo software.

Engenharia de Software e a Internet

- Atualmente, a Internet é uma plataforma de aplicativos em execução e, cada vez mais as organizações estão desenvolvendo sistemas baseadas na web, em vez de sistemas locais.

- Web services permitem que a funcionalidade da aplicação seja acessada pela Internet.

- Computação em Nuvem, é uma abordagem para a prestação de serviços de informática, em que as aplicações são executadas remotamente na 'nuvem'.

- Usuários não compram softwares, mas pagam de acordo com o uso.

Engenharia de Software e a Internet

- O reuso de softwares é a abordagem dominante para a construção de sistemas baseados na web.
- Ao construir esses sistemas, você deve pensar sobre como você pode montá-los a partir de sistemas e componentes pré-existentes de software.
- Sistemas baseados na web deve ser desenvolvidos e entregues de forma incremental.
- Atualmente, geralmente se reconhece que é impraticável especificar todos os requisitos para tais sistemas antecipadamente.
- Interfaces de usuário são limitadas pela capacidade de navegadores web.

Engenharia de Software e a Internet

- Tecnologias como Js permitem que as interfaces ricas sejam criadas dentro de um navegador web, mas ainda são difíceis de usar. Formulários web com scripts locais são mais usados .

- Sistemas baseados na web são sistemas complexos distribuídos, mas os princípios fundamentais da engenharia de software discutidos anteriormente também são aplicáveis a eles, assim como para qualquer outro tipo de sistema.

- As ideias fundamentais da engenharia de software, discutidas na seção anterior, se aplicam a softwares baseados em web da mesma forma como eles se aplicam a outros tipos de sistema de software.

Pontos Importantes

- A engenharia de software é uma disciplina da engenharia que se preocupa com todos os aspectos da produção de software.
- Atributos essenciais do produto de software são a manutenibilidade, confiança, proteção, eficiência e aceitabilidade.
- As atividades de alto nível de especificação, desenvolvimento, validação e evolução fazem parte de todos os processos de software.
- As ideias fundamentais da engenharia de software são universalmente aplicáveis a todos os tipos de desenvolvimento do sistema.
- Existem muitos tipos diferentes de sistemas e cada um requer ferramentas de engenharia de software e técnicas apropriadas para o seu desenvolvimento.

Mitos do Software

Mitos do Administrativos, do Cliente e do Profissional

Mitos Administrativos

- **Mito:** Meu pessoal tem ferramentas de desenvolvimento de software de última geração; afinal compramos para eles os mais novos computadores.

Mitos Administrativos

- **Realidade:**
  - É preciso muito mais do que os mais recentes computadores para se fazer um desenvolvimento de software de alta qualidade.
  - Ferramentas de Engenharia de Software Auxiliada por Computador - **CASE (Computer-Aided Software Engineering)** são mais importantes do que o hardware.

Mitos Administrativos

- **Mito:**

Se nós estamos atrasados nos prazos, podemos adicionar mais programadores e tirar o atraso.

Mitos Administrativos

- **Realidade:**
  - O desenvolvimento de software não é um processo mecânico igual à manufatura.
  - Acrescentar pessoas em um projeto torna-o ainda mais atrasado. Pessoas podem ser acrescentadas.

Mitos dos Clientes

- **Mito:** Uma declaração geral dos objetivos é suficiente para se começar a escrever programas - podemos preencher os detalhes mais tarde.

Mitos dos Clientes

- **Realidade:**
  - Uma definição inicial ruim é a principal causa de fracassos dos esforços de desenvolvimento de software.
  - É fundamental uma descrição formal e detalhada do domínio da informação, função, desempenho, interfaces, restrições de projeto e critérios de validação.

Mitos dos Clientes

- **Mito:** Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível.

Mitos dos Clientes

- **Realidade:**
  - Requisitos podem ser mudados, mas o impacto varia de acordo com o tempo que é introduzido (projeto e custo).
  - Uma mudança, quando solicitada tardiamente num projeto, é mais dispendiosa do que a mesma mudança solicitada nas fases iniciais.

Mitos do Profissional

- **Mito:** Assim que escrevermos o programa e o colocarmos em funcionamento nosso trabalho estará completo.

Mitos do Profissional

- **Realidade:**
  - Os dados da indústria indicam que entre 50 e 70% de todo esforço gasto num programa serão despendidos depois que ele for entregue pela primeira vez ao cliente

Mitos do Profissional

- **Mito:** Enquanto não tiver o programa "funcionando", eu não terei realmente nenhuma maneira de avaliar sua qualidade.

Mitos do Profissional

- **Realidade:**
  - Mecanismo (Revisão Técnica Formal) de garantia de qualidade de software é aplicado desde o começo do projeto.
  - Revisões de software são um “filtro de qualidade” - descobre erros/defeitos.

Mitos do Profissional

- **Mito:** A única coisa a ser entregue em um projeto bem sucedido é o programa funcionando.

Mitos do Profissional

- **Realidade:**
  - Um programa funcionando é somente uma parte de uma Configuração de Software que inclui todos os itens de informação produzidos durante a construção e manutenção do software.

A DOCUMENTAÇÃO é o alicerce

Categorias de Tamanho de Softwares

Categoria	Tamanho da Equipe	Duração	Tamanho do Fonte (linhas de código)
Trivial	1	1-4 semanas	500
Pequeno	1	1-6 meses	1000 a 2000
Médio	2-5	1-2 anos	5 mil a 50 mil
Grande	5-20	2-3 anos	50 mil a 100 mil
Muito grande	100-200	4-5 anos	1 milhão
Extremamente grande	2000-5000	5-10 anos	1 a 10 milhões

https://exame.abril.com.br/tecnologia/servicos-do-google-somam-2-bilhoes-de-linhas-de-codigo/

Éticas na Engenharia de Software

- A engenharia de software envolve responsabilidades mais amplas do que a simples aplicação de habilidades técnicas.
- Engenheiros de software devem se comportar de uma maneira honesta e eticamente responsável para serem respeitados como profissionais.
- Comportamento ético é mais do que simplesmente agir em concordância com a lei, envolve seguir um conjunto de princípios moralmente corretos.

Questões de responsabilidade profissional

- Confidencialidade
  - ü Normalmente, os engenheiros devem respeitar a confidencialidade de seus empregadores ou clientes, independentemente de haver ou não um acordo de confidencialidade formal assinado entre eles.
- Competência
  - ü Engenheiros não devem falsear seus níveis de competência. Eles não devem aceitar trabalhos que estão fora da sua competência.

Questões de responsabilidade profissional

- Direitos de propriedade intelectual
  - ü Engenheiros devem estar cientes das leis locais que regulam a a propriedade intelectual, tais como patentes, direitos autorais, etc. Eles devem ser cuidadosos para assegurar que a propriedade intelectual dos empregadores e clientes esteja protegida.
- Uso indevido de computador
  - ü Engenheiros de software não devem usar suas habilidades técnicas para uso indevido de computadores de outras pessoas. A variação do mau uso do computador vai desde relativamente trivial (brincar com jogos na máquina de um empregador, por exemplo) a extremamente sérios (disseminação de vírus).

Código de ética ACM/IEEE

- As sociedades profissionais nos EUA têm cooperado para produzir um código de conduta ética.
- Membros destas organizações se comprometem com o código de ética quando entram nelas.
- O Código contém oito princípios relacionados ao comportamento e decisões tomadas por engenheiros de software profissionais, incluindo profissionais, educadores, gestores, supervisores e políticos, bem como estagiários e estudantes da profissão.

Justificativas para o código de ética

- Os computadores têm um papel central e crescente no comércio, indústria, governo, medicina, educação, entretenimento e sociedade em geral.

- Os engenheiros de software são aqueles que contribuem através da participação direta ou através do ensino, para a análise, especificação, projeto, desenvolvimento, certificação, manutenção e testes de sistemas de software.

- Por causa de seus papéis no desenvolvimento de sistemas de software, os engenheiros de software têm significativas oportunidades de fazer o bem ou causar o mal, ou influenciar outros a fazerem o bem ou causarem o mal.

- Para garantir, tanto quanto possível, que seus esforços sejam usados para o bem, engenheiros de software devem se comprometer a fazer engenharia de software uma profissão benéfica e respeitada.

Príncipios Éticos

1. PÚBLICO — Engenheiros de software devem agir de acordo com o interesse público.

2. CLIENTE E EMPREGADOR — Engenheiros de software devem agir de maneira que seja do melhor interesse de seu cliente e empregador e de acordo com o interesse público.

3. PRODUTO — Engenheiros de software devem garantir que seus produtos e modificações relacionadas atendam aos mais altos padrões profissionais possíveis.

4. JULGAMENTO — Engenheiros de software devem manter a integridade e a independência em seu julgamento profissional.

Príncipios Éticos

5. GERENCIAMENTO — Gerentes e líderes de engenharia de software devem aceitar e promover uma abordagem ética para o gerenciamento de desenvolvimento e manutenção de software.

6. PROFISSÃO — Engenheiros de software devem aprimorar a integridade e a reputação da profissão de acordo com o interesse público.

7. COLEGAS — Engenheiros de software devem auxiliar e ser justos com seus colegas.

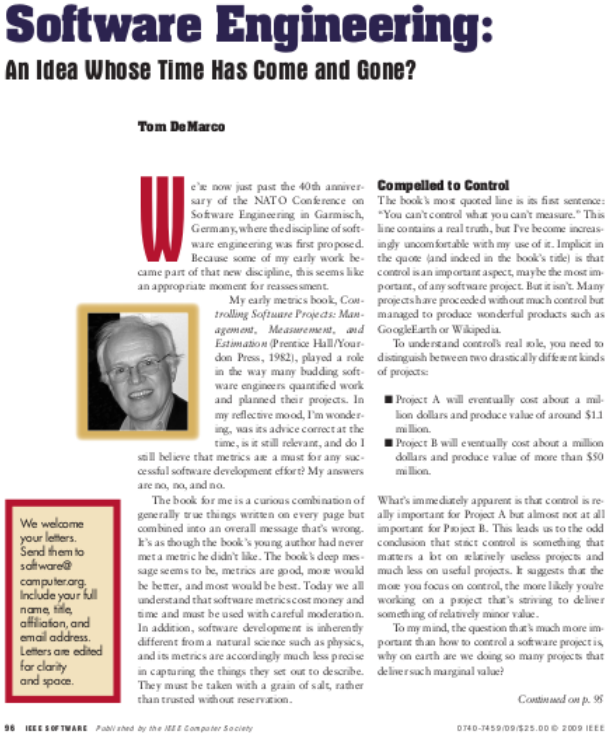
8. SI PRÓPRIO — Engenheiros de software devem participar da aprendizagem contínua durante toda a vida, e devem promover uma abordagem ética para a prática da profissão.

Príncipios Éticos

- Desacordo, em princípio, com as políticas da gerência sênior.
- Seu empregador age de forma antiética e libera um sistema crítico de segurança sem terminar os testes do sistema.
- Participação no desenvolvimento de sistemas de armas militares ou sistemas nucleares.

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)