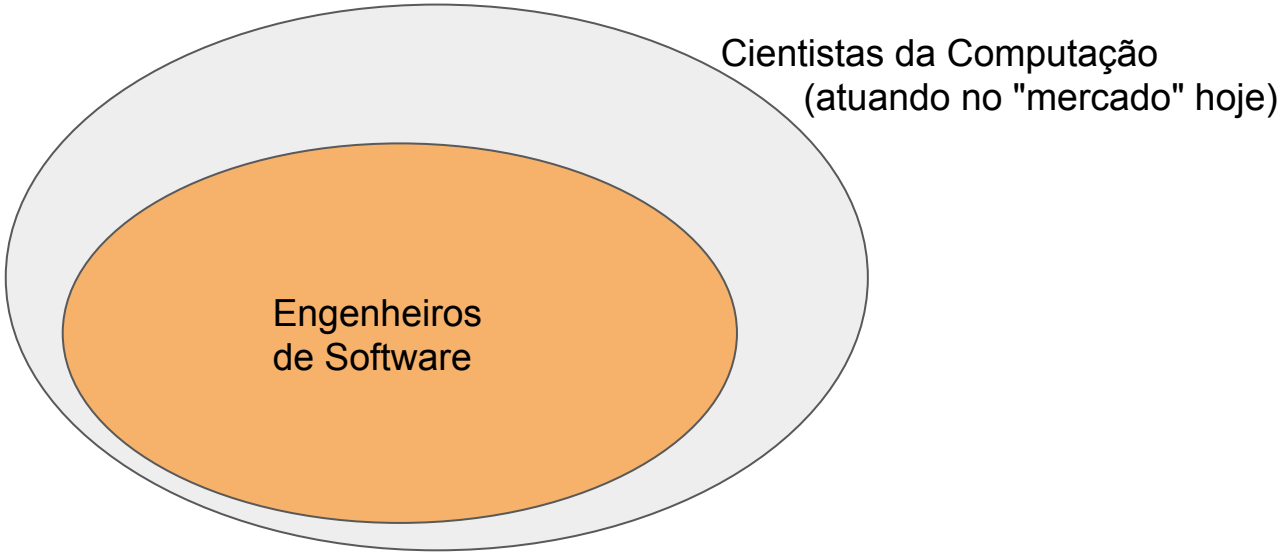
Introdução

Prof. Marco Tulio Valente

mtov@dcc.ufmg.br

Importância do Curso

- No futuro, existe grande chance de seu "título" ser **Engenheiro de Software**
  - Ou "fullstack developer", "arquiteto de software", "devops engineer", "mobile developer" etc
  - "Core" competência de todos eles: habilidades para desenvolver software de qualidade



Conferência da OTAN (Alemanha, 1968)

- Primeira vez que o termo "Engenharia de Software" foi usado.

Working Conference on **Software Engineering**

Trechos do relatório final da conferência

- The present report is concerned with a problem crucial to the use of computers, the so-called software ...
- The phrase **software engineering** was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.

Comentário de um participante [no relatório final]

- The basic problem is that certain classes of systems are placing demands on us which are beyond our capabilities and our theories and methods of design and production at this time.
- There are many areas where there is no such thing as a crisis — sort routines, payroll applications, for example.
- It is large systems that are encountering great difficulties. We should not expect the production of such systems to be easy.

O que se estuda em Engenharia de Software?

- Uma boa resposta pode ser encontrada no **Software Engineering Body of Knowledge (SWEBOK)**, editado pela IEEE:
  - "a guide to the body of knowledge developed over more than 4 decades"
  - "constitutes a valuable characterization of the software engineering"

Definição de Engenharia de Software (SWEBOK)

- Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar e manter software
- É a aplicação de princípios de engenharia na construção de software

Em seguida, define 15 Áreas de Conhecimento (KA)

Table I.1. The 15 SWEBOK KAs
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Models and Methods
Software Quality
Software Engineering Professional Practice
Software Engineering Economics
Computing Foundations
Mathematical Foundations
Engineering Foundations

Essas três últimas áreas são áreas de suporte às áreas "core" de ES

Restante destes slides

- Vamos dar uma visão rápida de cada um destas áreas
- Para que o aluno tenha um entendimento horizontal (embora ainda superficial) do que é Engenharia de Software
- No restante do curso, vamos nos aprofundar em algumas das áreas de conhecimento da Engenharia de Software (processos, projeto, testes e manutenção)

(1) Requisitos de Software

- Requisitos: "o que sistema deve fazer para atender aos seus clientes"
- Engenharia de Requisitos: atividades onde os requisitos de um sistema são especificados, analisados, documentados e validados.
- Requisitos podem ser funcionais vs não-funcionais:
    - **Funcionais**: "o que" um sistema deve fazer; quais funcionalidades ou serviços ele deve implementar. Exemplo: quando uma nota de uma prova for lançada, os alunos devem ser notificados por e-mail
    - **Não-funcionais**: "como" um sistema deve operar, sob quais "constraints" e qual qualidade de serviço ele deve oferecer. Exemplo: o e-mail acima deve chegar em até um minuto.

Exemplos de Requisitos Não-Funcionais

- Desempenho: "deve dar o saldo da conta em menos de 5 segundos"
- Disponibilidade: "deve estar no ar 99.99% do tempo"
- Capacidade: "deve ser capaz de armazenar dados de 1M de clientes"
- Tolerância a falhas: "deve continuar operando mesmo se São Paulo cair"
- Segurança: "deve criptografar todos os dados trocados com as agências"
- Privacidade: "não deve armazenar dados de localização dos usuários"
- Interoperabilidade: "deve se integrar com o sistema X do Banco Central"
- Manutenibilidade: "bugs devem ser corrigidos em até 24 horas"
- Usabilidade: "deve ter uma versão para celulares e tablets"

(2) Projeto de Software

- Definição da arquitetura e principais **componentes** de um sistema; descreve como um sistema é organizado em componentes e as **interfaces** que esses componentes devem usar para se comunicarem
- Dois níveis de "design":
  - High-level (ou arquitetural): interfaces entre componentes de maior granularidade (exemplo: subsistemas)
  - Low-level: interfaces entre componentes de menor granularidade (exemplo: classes)

Information Hiding (exemplo de Princípio de Projeto)

- Todo módulo deve possuir:
  - Funções **públicas** (estáveis, que constituem sua interface ou API)
  - Funções **privadas** (com código sujeito a mudanças).
- Principais benefícios desse princípio:
  - Paraleliza o desenvolvimento (cada "time" trabalha em um módulo)
  - Facilita o entendimento (você pode começar a contribuir após dominar um único módulo)
  - Facilita manutenções (que tendem a ficar "isoladas" em módulo)

Uma tradução "pragmática" de Information Hiding

- Mail enviado por Jeff Bezos, da Amazon em 2002 para os seus funcionários:
  - All teams will henceforth expose their data and functionality through service interfaces.
  - Teams must communicate with each other through these interfaces.
  - There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

Source: Engineering Software As a Service: An Agile Approach Using Cloud Computing, Armando Fox & David Patterson

Uma tradução "pragmática" de Information Hiding

- Continuação:
  - It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.
  - All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
  - Anyone who doesn't do this will be fired.
  - Thank you; have a nice day!

(3) Construção de Software

- Implementação (codificação) do sistema
- Algumas "preocupações":
  - Algoritmos e estruturas de dados
  - Reúso, bibliotecas e frameworks
  - Tratamento de exceções; design by contract
  - Padrões de nome, layout e documentação de código
  - Ferramentas (IDEs, GUI builders, debuggers)

(4) Testes de Software

- Testes de software verificam se um programa apresenta um comportamento esperado, ao ser executado com um conjunto finito de casos de teste
- Por consequência:
  - “Program testing can be used to show the presence of bugs, but never to show their absence!” — Edsger W. Dijkstra
- Atualmente, teste não é mais visto como uma atividade que começa após a fase de codificação, tendo o único propósito de detectar falhas.
- Ao contrário, testes é uma atividade pervasiva a todas as etapas do ciclo de desenvolvimento de um produto de software

Terminologia: Defeitos, Bugs, Fallhas

- O seguinte código possui um **defeito** (defect) ou um **bug**:

```
if (condition)
    area = pi * raio * raio * raio;     // codigo defeituoso; o certo é: area = pi * raio * raio
```

- Quando (e somente quando) esse código for executado ele vai causar uma **falha** (failure); ou seja, o programa vai dar um resultado errado.
- Portanto: nem todo defeito/bug causa falhas; pois o código defeituoso pode nunca vir a ser executado/testado.

Falha de Software Famosa: Explosão do Ariane 5 (1996)

Veja também o vídeo: https://www.youtube.com/watch?v=kYUrqdUyEpI

30 segundos depois

Custo do foguete e satélite: US$ 500 milhões

Relatório do Comitê de Investigação

- The failure was caused by the complete loss of guidance and altitude information 37s after start of the main engine ignition sequence.
- This loss of information was due to **specification and design errors in the software** of the inertial reference system (SRI).
    - The SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.
    - The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.
- Uma das recomendações do relatório: **Review all flight software**

Código original (em ADA)

```
-- Overflow is correctly handled for the vertical component
L_M_BV_32 := TBD.T_ENTIER_16S((1.0 / C_M_LSB_BH) *
                                  G_M_INFO_DERIVE(T_ALG.E_BH));
if L_M_BV_32 > 32767 then
 P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elseif L_M_BV_32 < -32768 then
 P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
 P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TBD.T_ENTIER_16S(L_M_BV_32));
end if;

-- But not for the horizontal one
P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS(TBD.T_ENTIER_16S
                                  ((1.0 / C_M_LSB_BH) *
                                  G_M_INFO_DERIVE(T_ALG.E_BH));
```

Source: https://github.com/ICTeam28/ProgramAnalysis/blob/master/summaries/stories/ariane5.md

Código com defeito corrigido

```
L_M_BV_32 := TBD.T_ENTIER_16S((1.0 / C_M_LSB_BH) *
                                  G_M_INFO_DERIVE(T_ALG.E_BH));
if L_M_BV_32 > 32767 then
 P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elseif L_M_BV_32 < -32768 then
 P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
 P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TBD.T_ENTIER_16S(L_M_BV_32));
end if;

L_M_BH_32 := TBD.T_ENTIER_16S((1.0 / C_M_LSB_BH) *
                                  G_M_INFO_DERIVE(T_ALG.E_BH));
if L_M_BH_32 > 32767 then
 P_M_DERIVE(T_ALG.E_BH) := 16#7FFF#;
elseif L_M_BH_32 < -32768 then
 P_M_DERIVE(T_ALG.E_BH) := 16#8000#;
else
 P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS(TBD.T_ENTIER_16S(L_M_BH_32));
end if;
```

Test-Driven Development (TDD)

- Prática proposta por XP (um método ágil)
- Ideia: primeiro escreve-se o código de teste, para então escrever o código que será testado
- Vantagens:
  - Incentiva a escrita de testes; pois evita que desenvolvedores adiem indefinidamente a escrita de testes
  - Força desenvolvedores a pensar primeiro na interface do código que vão implementar; pois ao escrever o teste os desenvolvedores se colocam na posição de primeiros usuários do seu próprio código

(5) Manutenção de Software

- Pode chegar a 80%, 90% dos custos de um projeto de software
- Por que? manutenção de software não é apenas correção de bugs
  - Corretiva: correção de bugs reportados
  - Preventiva: correção de bugs latentes (via, por exemplo, inspeção e revisão de código)
  - Adaptativa: "migrar meu sistema de Python 2.7 para Python 3"
  - Perfectiva: novas features e melhorias no código fonte (refactorings) ou na documentação

Table 5.1. Software Maintenance Categories

	Correction	**Enhancement**
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

Exemplo de Manutenção Preventiva: Y2K Bug

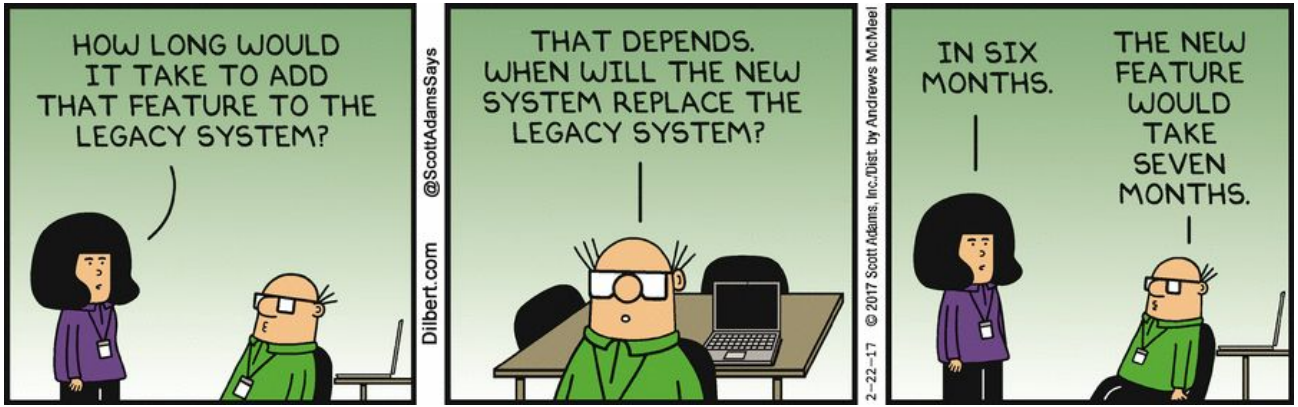
- Manutenção realizada no "tamanho" de campos data, para armazenar o ano com 4 dígitos, em vez de 2 (de DDMMAA para DDMMAAAA)
- Foi realizada "preventivamente" por empresas, antes da virada do milênio
- Riscos e problemas foram super-estimados, na maioria das vezes

Refactoring

- Refactoring: manutenção (perfectiva) realizada exclusivamente para incrementar manutenibilidade; ou seja, não corrige bug, não implementa nova funcionalidade etc
- Exemplos clássicos:
  - Rename variable/class/etc
  - Extract function/class/interface/package
  - Move function/class
  - etc

Sistemas Legados

- Sistemas antigos (usando linguagens, sistemas operacionais, bancos de dados antigos), cuja manutenção é normalmente custosa e arriscada
- Muitos vezes, são sistemas importantes e críticos (legado != irrelevante)



(6) Gerência de Configuração

- Garantir que as mudanças realizadas em um sistema sejam devidamente identificadas, rastreadas e passíveis de recuperação
- Atualmente, realizada via um sistema de controle de versões (como, git)
- Algumas preocupações:
  - Qual sistema de controle de versões será usado?
  - Qual o esquema será usado para numerar versões?
  - Como releases serão identificadas? (releases= versão de um sistema distribuída para seus clientes).

Exemplo: Google's monorepo

- Todos os sistemas do Google usam um único repositório monolítico (um monorepo), gerenciado por um sistema de controle de versão proprietário
- Vantagens: tende a facilitar reúso e refactorings (em larga escala) e também o gerenciamento de dependências
- Requer o desenvolvimento de ferramentas de gerenciamento próprias.

Google repository statistics, January 2015.

Total number of files	1 billion
Number of source files	9 million
Lines of source code	2 billion
Depth of history	35 million commits
Size of content	86TB
Commits per workday	40,000

(7) Gerência de Projetos

- Desenvolvimento de software demanda o uso de práticas e atividades de gerenciamento de projetos
  - Negociação de contratos (com clientes): prazos, valores, cronogramas
  - Gerência de recursos humanos (contratação, treinamento etc)
  - Gerenciamento de riscos (exemplo: turnover)
  - Acompanhamento da concorrência, marketing etc
  - Análise de viabilidade
- Sempre lembrar da Lei de Brooks: "Adicionar novos desenvolvedores em um projeto que esteja atrasado, só vai torná-lo mais atrasado ainda"

Stakeholders

- Designa todas as "partes interessadas" em um projeto de software
- São aqueles que **afetam** ou que **são afetados** pelo projeto
- Inclui desenvolvedores, clientes, usuários, gerentes, empresas sub-contratadas, fornecedores de qualquer natureza, governo etc

Comentário interessante

- A surprisingly **small portion of what people think of as a software project is actually not about software development.**
  - interacting with 3rd parties, writing contracts, hiring developers;
  - planning how your project interacts with other projects and the organization,
  - responding to regulations and external enquiries; managing the demand for certainty from stakeholders;
  - obtaining funding, maintaining financial control...
- These are the issues that **real software projects have to deal with**

Source: https://news.ycombinator.com/item?id=18662668

O que dá fazer com metade do orçamento?

Source: https://dilbert.com/strip/1999-09-13

(8) Processos de Desenvolvimento de Software

- Um processo de software define quais atividades devem ser seguidas para construir e entregar um sistema de software
- Será o primeiro tema a ser aprofundado neste curso
- Dois principais modelos:
  - Waterfall ("cascata")
  - Ágil (ou incremental ou iterativo)

Modelo em Cascata

- Inspirado em processos usados em engenharias tradicionais (Civil, Mecânica); onde os custos de uma falha de design ou implementação podem ser enormes, até em termos de vidas humanas
- Proposto na década de 70 e muito comum até a década de 90
- Parte do sucesso pode ser explicada pela sua "padronização" pelo Departamento de Defesa Norte-Americano, em 1985.

Modelo em Cascata

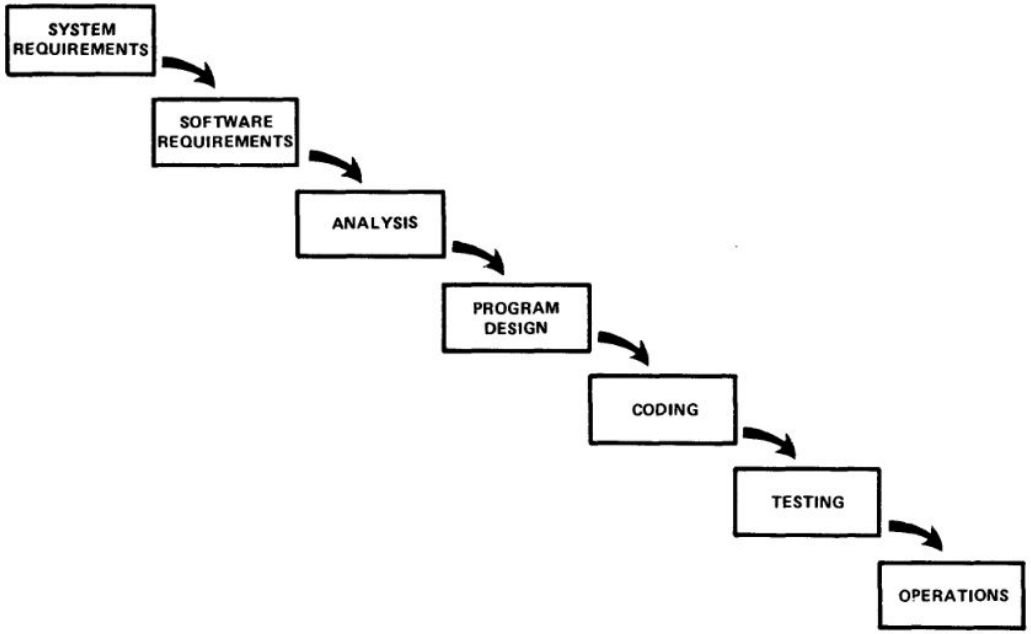


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Source: Winston Royce, "Managing the Development of Large Software Systems", 1970.

Problemas com Modelo Waterfall (1)

- Requisitos de software mudam com frequência (e a tendência no mundo "moderno" é que eles mudem com mais frequência ainda)
- Clientes às vezes não sabem o que "querem"
- Documentações de software (textuais e mesmo gráficas) tendem a ser pesadas e verbosas, podendo rapidamente se tornarem obsoletas

Problemas com Modelo Waterfall (2)

- Às vezes, é muito difícil projetar totalmente um sistema "antes" de qualquer código ser implementado, testado e validado pelos usuários
- Isto é, normalmente a melhor solução é vislumbrada no "meio do caminho"
- Levantamento completo de requisitos e projeto detalhado podem levar muito tempo; quando ficarem prontos, o "mundo já mudou"
- Como resultado, sistemas projetados neste modelo podem atrasar e/ou estourar seus cronogramas

CHAOS Report (Standish Group, 1994)

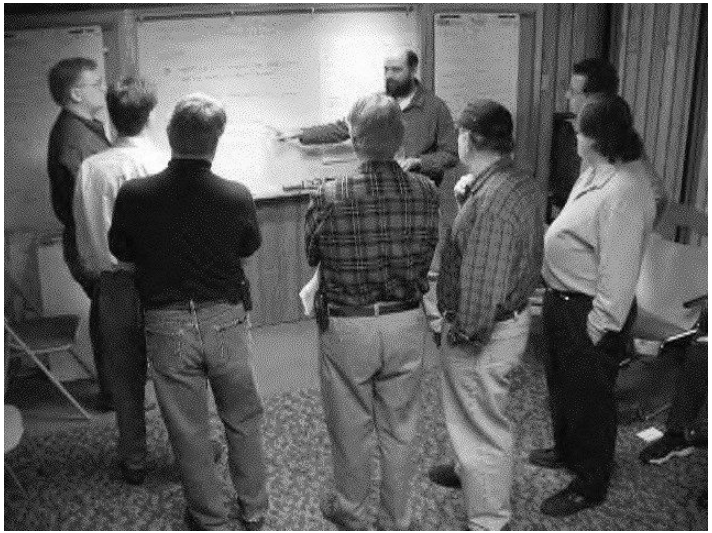
- Survey realizado com empresas que contrataram projetos de desenvolvimento de software; revelou uma situação "caótica", com atrasos frequentes e não cumprimento de orçamento

Time Overruns	% of Responses
Under 20%	13.9%
21 - 50%	18.3%
51 - 100%	20.0%
101 - 200%	35.5%
201 - 400%	11.2%
Over 400%	1.1%

Cost Overruns	% of Responses
Under 20%	15.5%
21 - 50%	31.5%
51 - 100%	29.6%
101 - 200%	10.2%
201 - 400%	8.8%
Over 400%	4.4%

Manifesto Ágil (2001)

- Lançado após um "encontro" de 17 engenheiros de software em Utah
- Uma crítica a "paradigmas" sequenciais e pesados de desenvolvimento
- Proposta de um novo paradigma, mais ágil, incremental, iterativo etc



Manifesto Ágil

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, **while there is value in the items on the right**, we value the items on the left more

Desenvolvimento Ágil

- Profundo impacto na indústria de software e hoje é largamente adotado
- Hoje, tudo é "ágil" …
- Talvez o adjetivo esteja até "desgatado"

Agilidade ganhou até capa de revista no Brasil

Revista Você S/A, março 2019

Desenvolvimento Ágil

- Recomendado quando o custo de falhas é tolerável
- Contra-exemplos: sistemas críticos e/ou embarcados, que lidam com vidas humanas
- Métodos ágeis: XP, Scrum, Kanban, Lean Development etc
- Baseado em novas práticas:
    - Pair programming (ou code reviews), test-first programming, incremental design, continuous integration, etc

Exemplo: Waterfall vs Agile (apenas para ilustrar)

- Suponha o seguinte projeto: construir uma ponte sobre um rio
- Waterfall (*plan-driven* ou *plan-and-document*):
  - Projeto preliminar da ponte e requisitos (largura etc); então assina-se o contrato ($$$, prazos)
  - Faz uma maquete; um projeto de engenharia, estrutural, impacto ambiental etc
  - Simula em um computador ou túnel de vento (?)
  - Constrói a ponte; em em seguida entrega e inaugura
- Agile (incremental, iterativo):
  - Constrói uma primeira versão, com uma única pista
  - Em seguida, constrói uma segunda pista
  - Depois, duplica as duas pistas, etc
- Exemplo apenas ilustrativo; projetos de engenharia, como uma ponte, são sempre *plan-driven*; apenas para projetos com maior grau de "incerteza", como é o caso de software, um método ágil (incremental) faz mais sentido

Práticas Ágeis: Impacto ainda em andamento

- Indústria de software está sempre procurando "encurtar" tempo de lançamento de releases.

waterfall or unified process	evolutionary development	agile development	Facebook	continuous deployment
once	months	weeks	one day	<hour

Source: Development and Deployment at Facebook. IEEE Internet Computing 2013.

- Objetivo: rápido feedback do usuário.
- Sistemas da Netflix: toda feature é tratada como um experimento; se ela não é bem recebida, ela é reformulada. Se mesmo assim ela não faz sucesso, ela é deletada (Source: Top-10 Adages in Continuous Deployment, IEEE Software, 2017)

Todo projeto de software precisa de um processo?

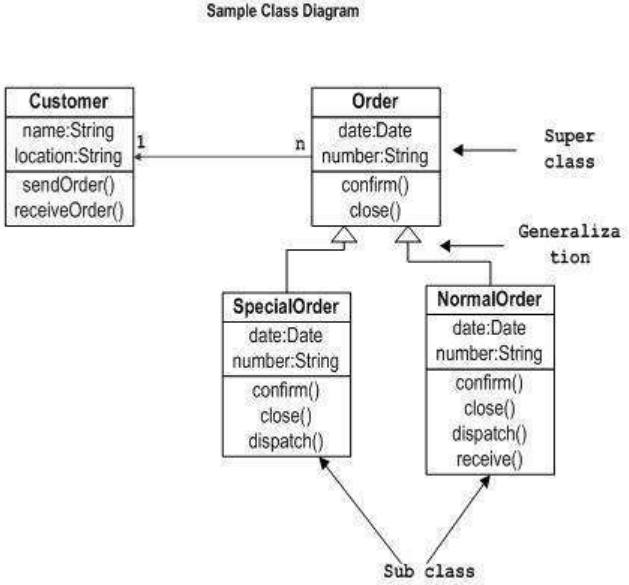
- Argumento comum:
  - Linus Torvalds criou o Linux sem usar nenhum processo
  - Idem para Donald Knuth (Latex), Tim Berners-Lee (Web), etc
  - Logo, por que eu preciso seguir um processo?
- Claro, que existem exceções; como as mencionadas acima
- Porém, na maioria dos casos, pessoas precisam de processos, mesmo que leves, mas com regras, práticas, controles, deadlines, prazos etc.
- Principalmente no caso de **sistemas desenvolvidos em equipes**, por organizações, sob contratos, com salários etc (o que cria um contexto muito diferente de um sistema pessoal; mais sobre isso à frente neste curso)

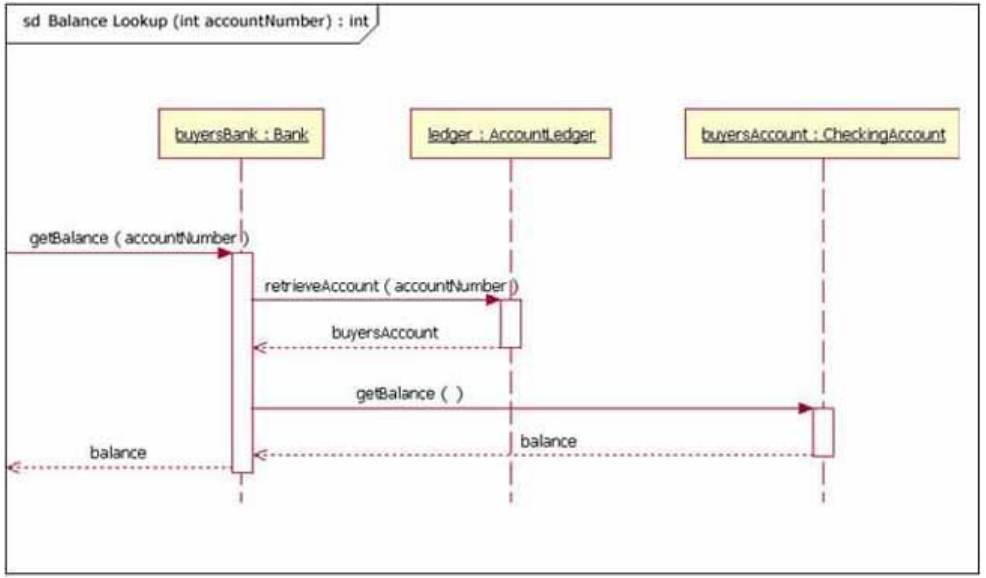
(9) Modelagem de Software

- Modelo = representação mais alto nível, do que o código fonte
- Às vezes, é útil criar um modelo de um software antes de sua implementação
  - para explicar o "design" para o time de desenvolvimento
  - para documentar o projeto
  - para facilitar compreensão, manutenção e evolução futuras etc
  - às vezes, para gerar código automaticamente, se modelo for formal
- Mesmo em métodos ágeis, existe alguma documentação, fora do código
  - "Software em funcionamento mais que documentação abrangente"
  - Assim, é uma questão de "prioridade" e não de eliminar toda documentação

UML (Unified Modeling Language)

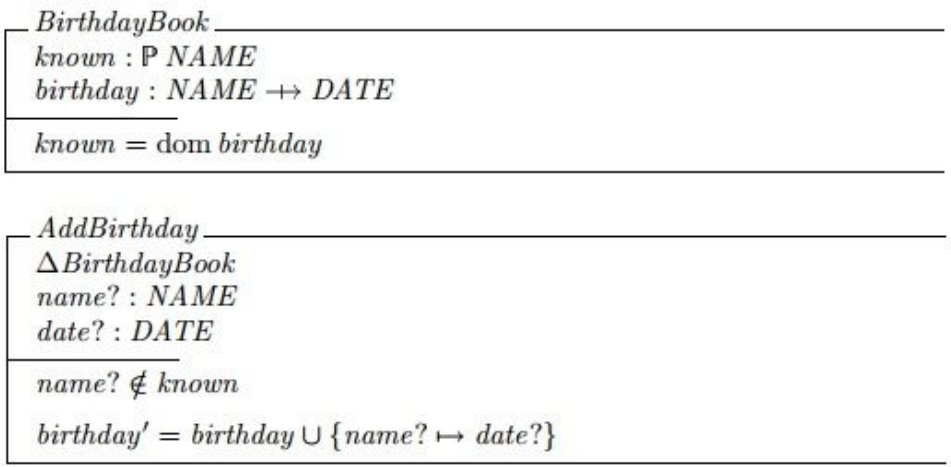
- Linguagem (gráfica) para modelagem de software
- Diagramas estruturais (exemplo: diagrama de classes, à esquerda) e comportamentais (exemplo: diagrama de atividades, à direita)





Métodos Formais

- Tentativa de se "formalizar" a especificação e modelagem de um sistema, usando uma linguagem matemática (por exemplo, baseada em teoria de conjuntos, lógica etc)



(10) Qualidade de Software

- Qualidade Externa:
  - Correção, robustez, extensibilidade, reusabilidade, eficiência, compatibilidade, facilidade de uso, portabilidade, verificabilidade
- Qualidade Interna:
  - Modularidade, legibilidade, testabilidade, manutenibilidade etc
- Métricas de software:
  - "You can't control what you can't measure" (Tom de Marco)
  - Métricas de código: tamanho, complexidade ciclomática, coesão, acoplamento etc
  - Métricas de processo: número de bugs / linha de código

(11) Prática Profissional

- Diversos pontos
  - Ética
  - Certificações
  - Regulamentação da Profissão
  - Cursos de graduação
  - Currículo de Referência

Aspectos Éticos: Google (2018)

- Engenheiros de Software começam a questionar decisões de suas empresas
- Includindo o uso que elas podem fazer do software desenvolvido por eles
- Por exemplo, começam a se preocupar com o potencial uso militar dos sistemas nos quais eles estão trabalhando

Cybersecurity

Google Engineers Refused to Build Security Tool to Win Military Contracts

A work boycott from the Group of Nine is yet another hurdle to the company's efforts to compete for sensitive government work.

https://www.bloomberg.com/news/articles/2018-06-21/google-engineers-refused-to-build-security-tool-to-win-military-contracts

(12) Aspectos Econômicos

- Qual o retorno de investimento?
- Como monetizar minha startup? (anúncios, assinaturas, etc)
- Qual a melhor licença para o meu software?

Para finalizar: Tipos ABC de sistemas

- Classificação proposta por Bertrand Meyer
  - https://bertrandmeyer.com/2013/03/25/the-abc-of-software-engineering/
- Segundo essa classificação, existem três tipos de sistemas de software:
  - Sistemas C (Casuais)
  - Sistemas B (Business)
  - Sistemas A (Acute)
- Uma mesma empresa pode ter sistemas A, B e C

Sistemas C (Casuais)

- Não existe pressão para terem níveis altos de qualidade
- Podem ter bugs; às vezes, são sistemas descartáveis; podem ter código duplicado; não precisam ter desempenho ou uma boa interface
- Desenvolvidos por 1-2 engenheiros
- Sistemas pequenos, para controlar algo sem grande importância
- Tipo mais comum de sistema
- Não se beneficiam das práticas, técnicas e processos deste curso.
- No caso destes sistemas, o risco é "over-engineering"

Sistemas B (Business)

- Sistemas críticos para uma organização; geram lucro etc
- São os sistemas que vão se beneficiar do que veremos neste curso
- Risco: não usarem técnicas de ES e rapidamente se tornarem um "passivo", em vez de um "ativo" para as organizações

Sistemas A (Acute)

- Sistemas onde nada pode dar errado, pois o custo é imenso, em termos de vidas humanas e/ou grandes montantes financeiros
- Exemplos: sistemas de transporte, médicos, aviação, espaciais etc
- Normalmente, requerem certificações
- Estão fora do escopo deste curso

Leitura Complementar

- "No Silver Bullet Essence and Accident in Software Engineering"
  - http://www.cs.nott.ac.uk/~pszcah/G51ISS/Documents/NoSilverBullet.html
- Ensaio muito famoso, de autoria de Frederick Brooks, Prêmio Turing de 1999
  - Por contribuições nas áreas de Arquitetura, Sistemas Operacionais e Engenharia de Software
  - "No Silver Bullet" é um dos ensaios de um livro do autor, chamado "The Mythical Man-Month"
  - Outras ideias famosas deste livro: Lei de Brooks, Integridade Conceitual etc