Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - IFSP
Câmpus Jacareí

Tecnologia em Análise e Desenvolvimento de Sistemas - ADS

2º Semestre de 2023

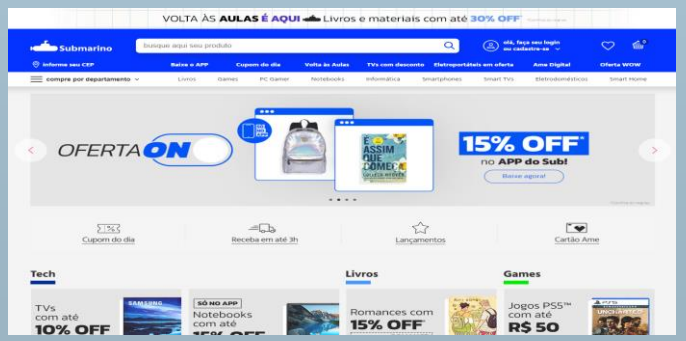
Engenharia de Software 1 – JCRESW1

Prof. Lineu Mialaret

Aula 4: Conceitos Básicos em Engenharia de Software (2)

Introdução

- Software no Mundo Atual:
  - Empresas de qualquer tamanho dependem dos mais diversos sistemas de software para automatizar seus processos.
  - Governos também interagem com os cidadãos por meio de sistemas de software, para por exemplo, coletar impostos, fornecer documentos ou realizar eleições.
  - Empresas vendem, por meio de sistemas de comércio eletrônico, uma gama imensa de produtos, diretamente para os consumidores.



Introdução (cont.)

- Software no Mundo Atual:
  - Software está também embarcado em diferentes dispositivos e produtos de engenharia, incluindo automóveis, aviões, satélites, robôs, etc.
  - Por fim, software está contribuindo para renovar indústrias e serviços tradicionais, como telecomunicações, transporte em grandes centros urbanos, hospedagem, lazer e publicidade.

Engenharia de Software

- Existe uma área da Ciência da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software, principalmente aqueles mais complexos e de maior tamanho, de forma produtiva (no prazo e orçamento estipulado) e com qualidade.
- Essa área é chamada de Engenharia de Software.
  - Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para conceber, modelar, desenvolver, operar, manter e evoluir software.
  - É a área da Computação que se preocupa em propor e aplicar princípios de engenharia na construção de software.

Criação da Engenharia de Software

- Historicamente, a Engenharia de Software surgiu no final da década de 60 do século passado.
- Nas duas décadas anteriores, os primeiros computadores modernos foram projetados e começaram a ser usados principalmente para resolução de problemas científicos.
  - Ou seja, nessa época software não era a preocupação central, mas sim construir máquinas que pudessem executar alguns poucos programas.
  - Computadores eram usados por poucos e para resolver apenas problemas científicos.

Criação da Engenharia de Software (cont.)

- No entanto, progressos contínuos nas tecnologias de construção de hardware mudaram de forma rápida esse cenário.
- No final da década de 60, computadores já eram mais populares, estavam presentes em várias universidades norte-americanas e europeias e chegavam também em algumas grandes empresas.
- Os cientistas da computação dessa época se viram diante de um novo desafio:
  - Como os computadores estavam se tornando mais populares, novas aplicações não apenas se tornavam possíveis, mas começavam a ser demandadas pelos usuários dos grandes computadores da época.

Criação da Engenharia de Software (cont.)

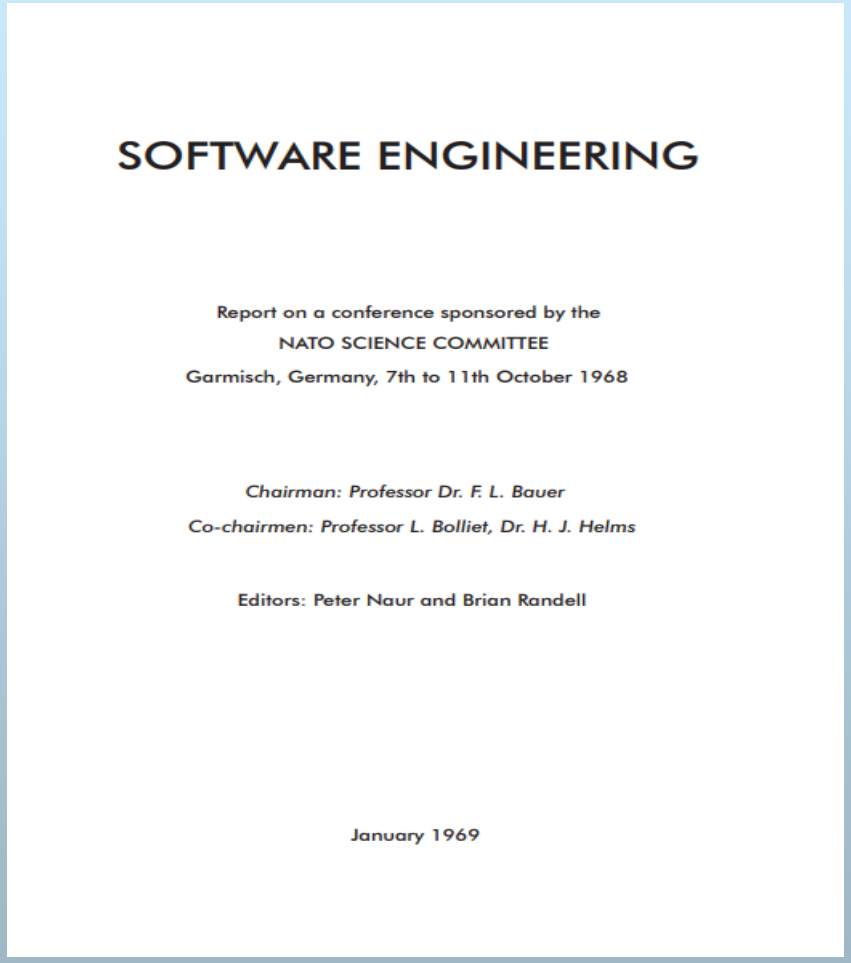
- Na verdade, os computadores eram grandes no sentido físico e não em poder de processamento, se comparado com os computadores atuais.
  - Dentre essas novas aplicações demandadas, as principais eram sistemas comerciais, como folha de pagamento, controle de clientes, controle de estoques, etc.

Criação da Eng. de Software (cont.)

Cientistas na conferência da OTAN de 1968 sobre Engenharia de Software

- Em outubro de 1968, um grupo de cerca de 50 renomados cientistas se reuniu durante uma semana em Garmisch, na Alemanha, em uma conferência patrocinada por um comitê científico da OTAN.
  - O objetivo da conferência era chamar a atenção para um problema crucial do uso de computadores, o chamado software.

Criação da Eng. de Software (cont.)



SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

January 1969

Relatório produzido na conferência da OTAN de 1968 sobre Engenharia de Software

- A conferência produziu um relatório, com mais de 130 páginas, que afirmava a necessidade de que software fosse construído com base em princípios práticos e teóricos, tal como ocorre em ramos tradicionais e bem estabelecidos da Engenharia.
  - Para deixar essa proposta mais clara, decidiu-se cunhar o termo Engenharia de Software.
  - Por isso, a Conferência da OTAN é considerada o marco histórico de criação da área de Engenharia de Software.

Criação da Eng. de Software (cont.)

- O comentário a seguir, de um dos participantes da Conferência da OTAN, ilustra os desafios que esperavam a recém-criada área de pesquisa:

> *"O problema é que certas classes de sistemas estão colocando demandas sobre nós que estão além das nossas capacidades e das teorias e métodos de projeto que conhecemos no presente tempo. Em algumas aplicações não existe uma crise, como rotinas de ordenação e folhas de pagamento, por exemplo. Porém, estamos tendo dificuldades com grandes aplicações. Não podemos esperar que a produção de tais sistemas seja fácil."*

55 Anos Depois

- Hoje, já se tem conhecimento de que software, na maioria das vezes, não deve ser construído em fases estritamente sequenciais, como ocorre com produtos tradicionais de engenharia, tais como Engenharia Civil, Engenharia Mecânica, Engenharia Eletrônica, etc.
- Já existem também padrões (*patterns*) que podem ser utilizados por Engenheiros de Software em seus novos sistemas, de forma que eles não precisem reinventar a roda toda vez que enfrentarem um novo problema de projeto (*design*) ou programação.

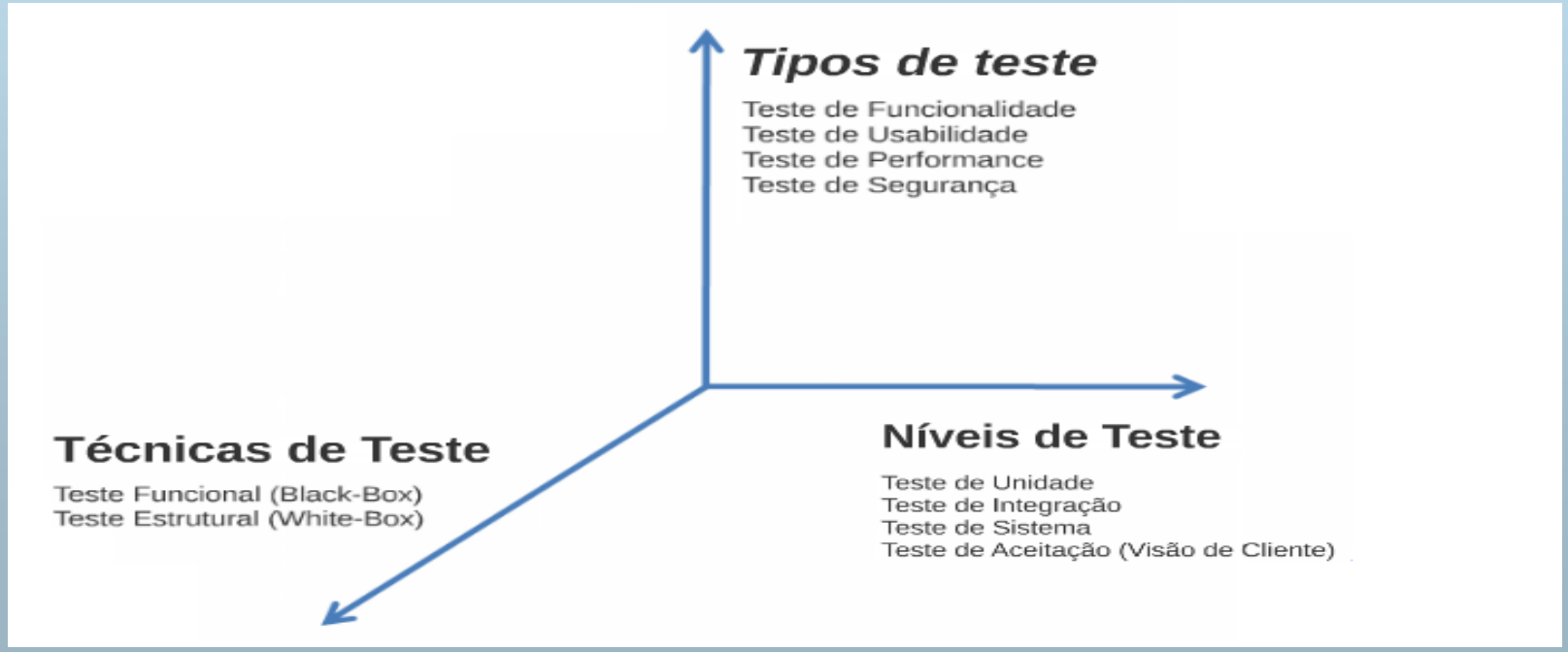
55 Anos Depois (cont.)

- Bibliotecas e *Frameworks* para os mais diversos fins estão largamente disponíveis, de forma que desenvolvedores de software podem reusar código sem se preocupar com detalhes inerentes a tarefas como implementar interfaces gráficas, criar estruturas de dados, acessar bancos de dados, criptografar mensagens, etc.

> Frameworks são estruturas de classes que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de software. A grande vantagem desta abordagem é a promoção de reuso de código e projeto, que pode diminuir o tempo e o esforço exigidos na produção de software.

55 Anos Depois (cont.)

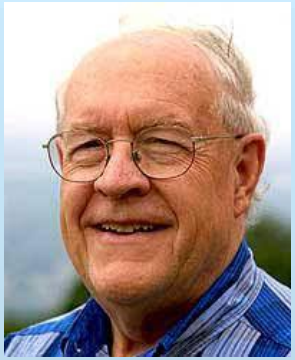
- Diversos Níveis, Tipos e Técnicas de Teste estão disponíveis e devem ser utilizadas para garantir que os sistemas em construção tenham qualidade e que falhas não ocorram quando eles entrarem em produção e forem usados por clientes reais.

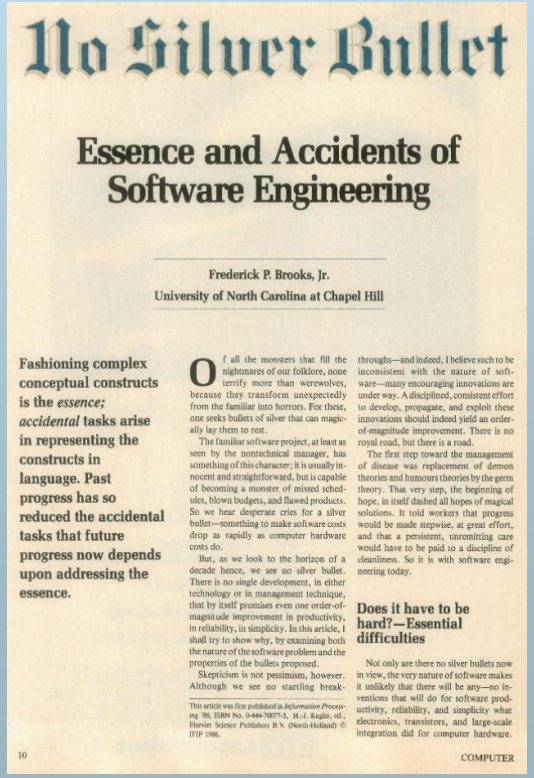


55 Anos Depois (cont.)

- Sabe-se também que sistemas se tornam obsoletos (envelhecem), como outros produtos de engenharia.
- Logo, software também precisa de manutenção, não apenas corretiva, para corrigir *bugs* (falhas) reportados por usuários, mas também para garantir que os sistemas continuem fáceis de entender e manter, mesmo com o passar dos anos.

Dificuldades no Desenvolvimento de Software





No Silver Bullet

Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Fashioning complex conceptual constructs is the *essence;* *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

The familiar software project, at least as seen by the nontechnical manager, has something of this character; it usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed.

Skepticism is not pessimism, however. Although we see no startling breakthroughs—and indeed, I believe such to be inconsistent with the nature of software—many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

Does it have to be hard?—Essential difficulties

Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware.

This article was first published in *Information Processing '86*, ISBN No. 0-444-70077-3, H.-J. Kugler, ed., Elsevier Science Publishers B.V. (North-Holland) © IFIP 1986.

- O desenvolvimento de software é diferente de qualquer outro produto de engenharia, principalmente quando se compara software com hardware.
- Frederick Brooks (ganhador do Prêmio Turing em Computação no ano de 1999) e um dos pioneiros da área de Engenharia de Software, foi um dos primeiros a chamar a atenção para esse fato.
  - Em 1987, em um ensaio intitulado *Não Existe Bala de Prata: Essência e Acidentes em Engenharia de Software*, ele discorreu sobre as particularidades da área de Engenharia de Software.

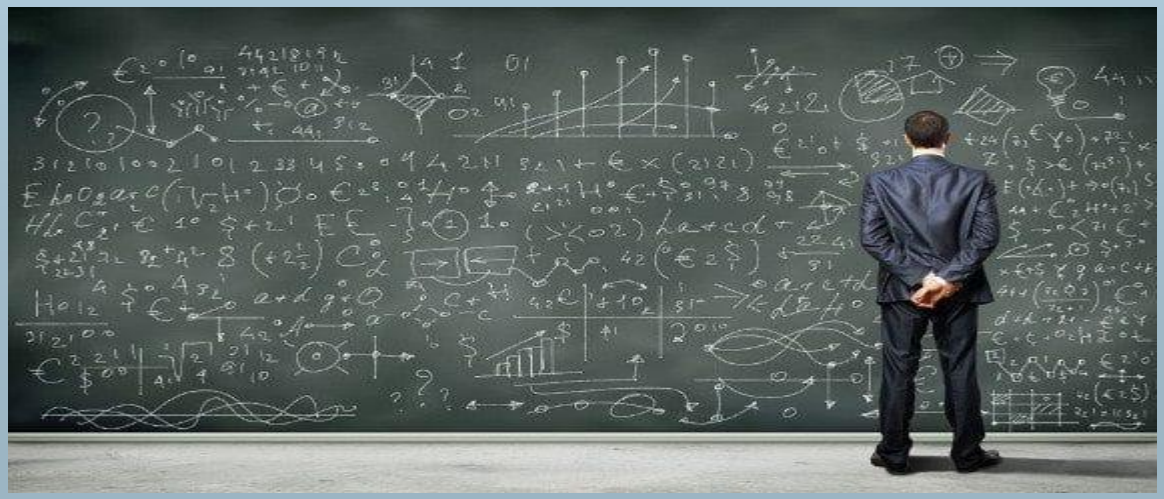
Dificuldades no Desenvolvimento de Software (cont.)

- Existem dois tipos de dificuldades em desenvolvimento de software:
  - Dificuldades Essenciais.
  - Dificuldades Acidentais.

- As Dificuldades Essenciais são da natureza da área (software) e dificilmente serão superadas por qualquer nova tecnologia ou método que se invente.
  - Complexidade
  - Conformidade
  - Facilidade de Mudança
  - Invisibilidade

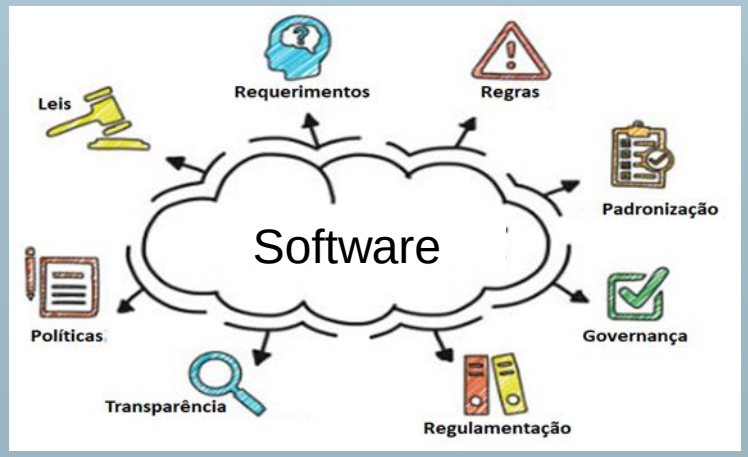
Dificuldades Essenciais

- 1) Complexidade
  - Dentre as construções que o homem se propõe a realizar, software é uma das mais desafiadoras e mais complexas que existe.
  - Como já mencionado antes, mesmo em construções de engenharia tradicional, como um satélite, uma usina nuclear ou um foguete, são cada vez mais dependentes de software.



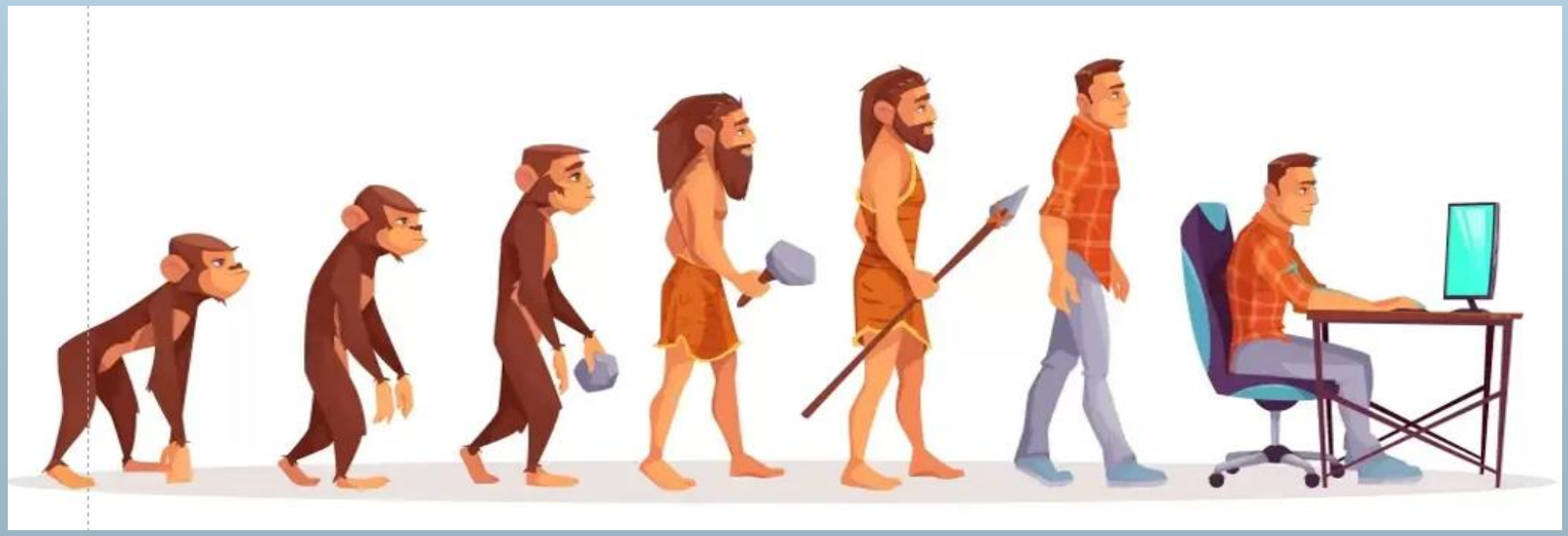
Dificuldades Essenciais (cont.)

- 2) Conformidade
  - Pela sua natureza software tem que se adaptar ao seu ambiente, que muda a todo momento no mundo moderno.
  - Exemplo:
    - Se as leis para recolhimento de impostos mudam, normalmente espera-se que os sistemas sejam rapidamente adaptados à nova legislação.
    - Isso não ocorre, por exemplo, na Física, pois as leis da natureza não mudam de acordo com os caprichos dos homens.



Dificuldades Essenciais (cont.)

- 3) Facilidade de Mudanças
  - Consiste na necessidade de evoluir sempre, incorporando novas funcionalidades.
  - Na verdade, quanto mais bem sucedido for um sistema de software, mais demanda por mudanças ele recebe.



Dificuldades Essenciais (cont.)

- 4) Invisibilidade
  - Devido à sua natureza abstrata, é difícil visualizar o tamanho e consequentemente estimar o esforço de construir um sistema de software.

Dificuldades Essenciais (cont.)

- As Dificuldades 2) Conformidade, 3) Facilidade de Mudanças e 4) Invisibilidade são específicas de sistemas de software, isto é, elas não ocorrem em outros produtos de Engenharia, pelo menos na mesma intensidade.
- Exemplo:
  - Quando a legislação ambiental muda, os fabricantes de automóveis têm anos para se conformar às novas leis.
  - Adicionalmente, carros não são alterados, pelo menos de forma essencial, com novas funcionalidades, após serem vendidos.
  - Por fim, um carro é um produto físico, com peso, altura, largura, assentos, forma geométrica, etc., o que facilita sua avaliação e precificação por consumidores finais.

Dificuldades Acidentais

- O desenvolvimento de software enfrenta também Dificuldades Acidentais.
  - No entanto, elas estão associadas a problemas tecnológicos, que os Engenheiros de Software podem resolver, se devidamente treinados e caso tenham acesso às devidas tecnologias e recursos.
- Exemplo:
  - Um compilador que produz mensagens de erro obscuras.
  - Uma IDE que possui muitos *bugs* e frequentemente sofre travamentos.
  - Um *framework* que não possui documentação.
  - Uma aplicação Web com uma interface pouco intuitiva, etc.

Complexidade do Software – Exemplo 1: Agenda Pessoal

- Objetivo:
  - Software para armazenar o nome de até 50 pessoas.
- Quanto custa para fazer?
- Quanto tempo vai levar para ficar pronto?
- Qual a consequência no caso de defeito?

Complexidade do Software – Exemplo 2: Boeing 777

- Objetivo:
  - Software para controlar todo o hardware do Boeing 777.
- Quanto custa para fazer?
- Quanto tempo vai levar para ficar pronto?
- Qual a consequência no caso de defeito?

Complexidade do Software – Exemplo 2: Boeing 777 (cont.)

- Tamanho
  - Mais de 4 milhões de linhas de código.
  - Linguagem dominante (>99%): Ada.
- Documentação
  - 100 a 10.000 páginas por subsistema.
  - Total de 79 subsistemas integrados.
- Duração
  - 4,5 anos de desenvolvimento.
- Ampla utilização de Engenharia de Software
- Em operação desde 1995
  - Zero acidentes graves até 2006.

Complexidade do Software – Exemplo 3: Google

- Os sistemas do Google somavam 2 bilhões de linhas de código, distribuídas por 9 milhões de arquivos, em janeiro de 2015.
  - Nesta época, cerca de 40 mil solicitações de mudanças de código (*commits*) eram realizadas, em média, por dia, pelos cerca de 25 mil Engenheiros de Software empregados pelo Google nessa época.

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

Complexidade do Software – Exemplo 4: Linux

- O sistema operacional Linux, em sua versão 5.8, de 2020, possuía cerca de mais de 28 milhões de linhas de código em seu *kernel* e contribuições de quase 1.700 engenheiros.

> When `linux-0.01.tar.Z` kernel was released, it consisted of 88 files and 10,239 lines of code and ran on a single hardware architecture, `i386`[8]. Today, the v5.8 release consists of 69,325 files and 28,442,673 lines of code (which corresponds to **110,354,844 tokens**), and it runs on over 30 major hardware architectures[9].