



Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA
Departamento de Ciência da Computação
Tecnólogo em Análise e Desenvolvimento de Sistemas

Evolução de software

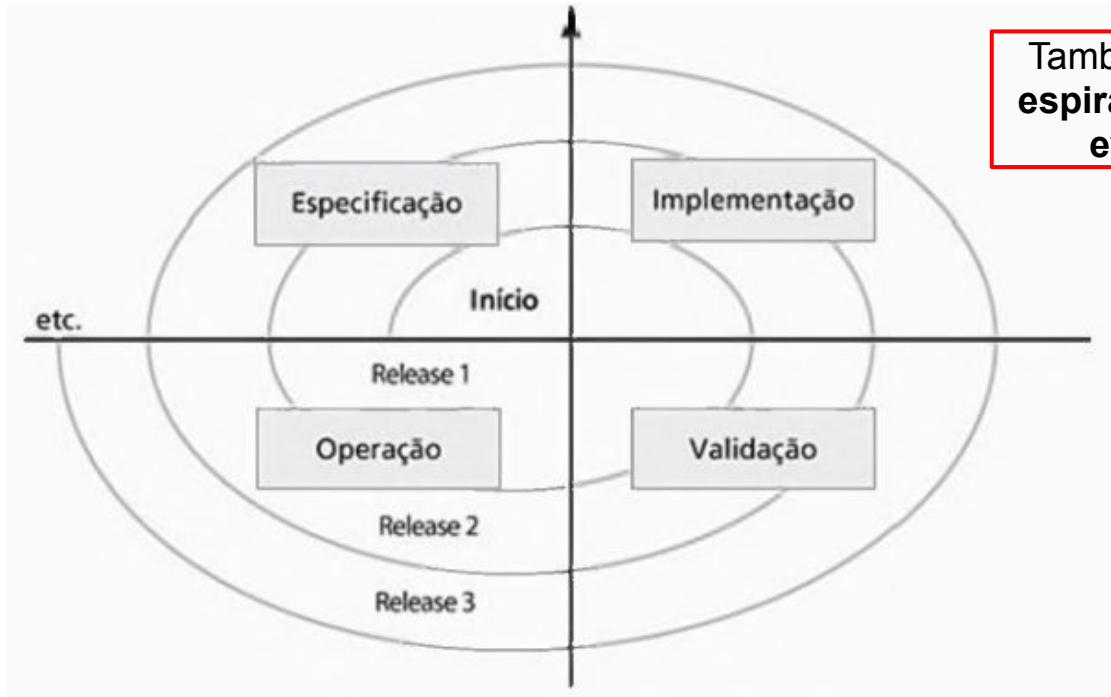
André L. R. Madureira <andre.madureira@ifba.edu.br>
Doutorando em Ciência da Computação (UFBA)
Mestre em Ciência da Computação (UFBA)
Engenheiro da Computação (UFBA)

Evolução de software

- Se um software é utilizado e útil para alguém ou alguma organização, ele estará em constante mudança
 - Requisitos dos sistemas instalados mudam de acordo com os negócios e suas mudanças de ambiente
- Sistemas de software podem ter uma vida útil muito longa
 - **Ex:** sistemas de controle de tráfego aéreo (30 anos)
 - **Razão:** softwares custam caro
- Processo de engenharia de software se repetem em um ciclo

Ciclo de vida de evolução do software

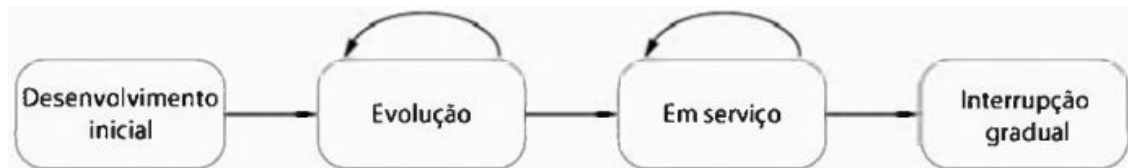
- Processo de engenharia de software se repetem em um ciclo



Também chamado de **modelo espiral do desenvolvimento e evolução de software**

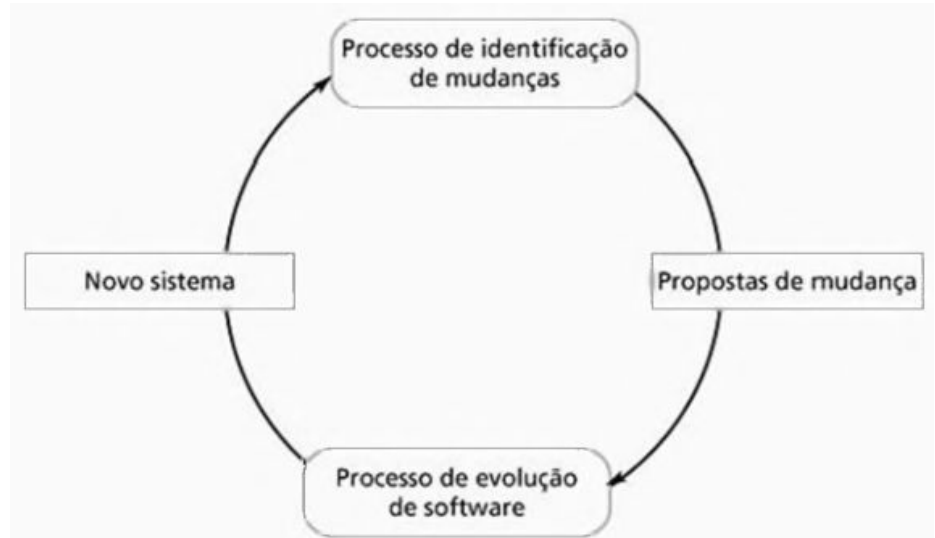
Visão alternativa do ciclo de vida de evolução do software

- Conforme um software é modificado, sua estrutura tende a degradar e as mudanças ficam mais e mais caras
 - Chega um ponto em que o software passa da etapa da evolução para o serviço.
 - **Etapas de serviço:** software ainda é útil e usado, mas apenas pequenas mudanças táticas são feitas
 - Empresa está considerando como o software pode ser substituído

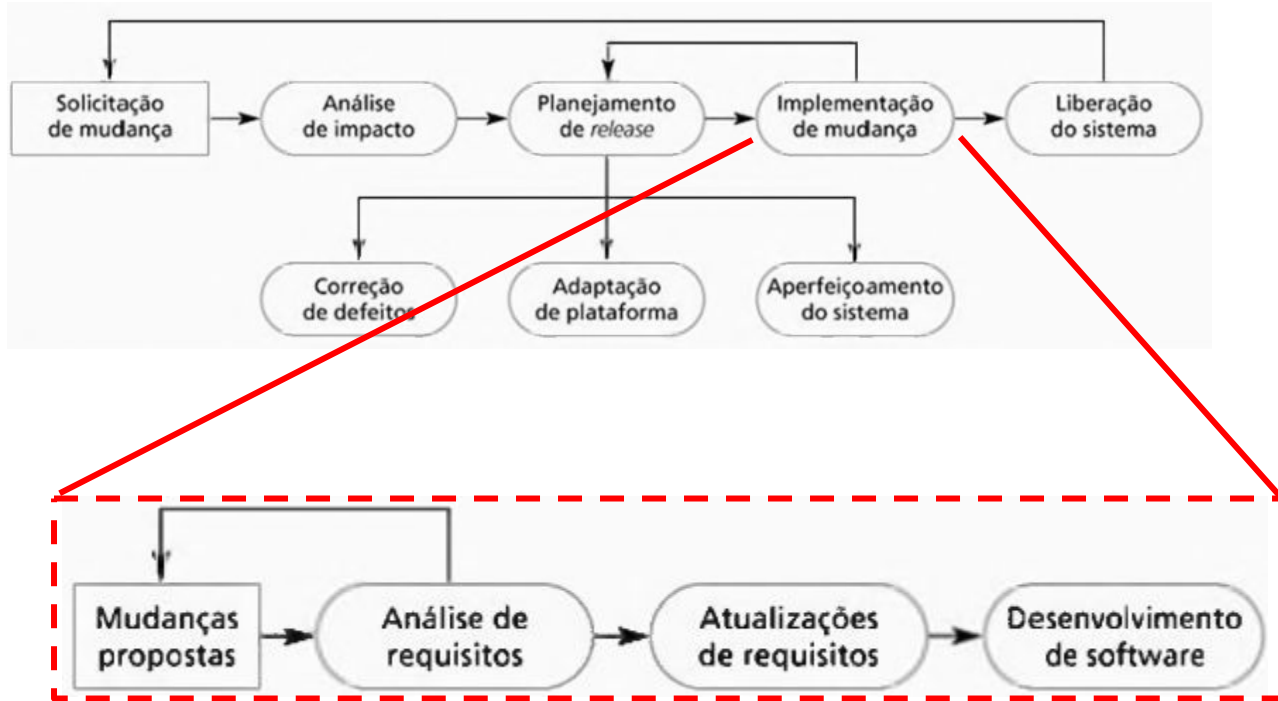


Mudanças de software

- Propostas de mudança podem surgir de necessidades recém descobertas, ou de requisitos que ainda não foram implementados no *release* atual do software
 - Propostas de mudança devem ser relacionadas aos componentes do sistema que necessitam ser modificados para implementar essas propostas

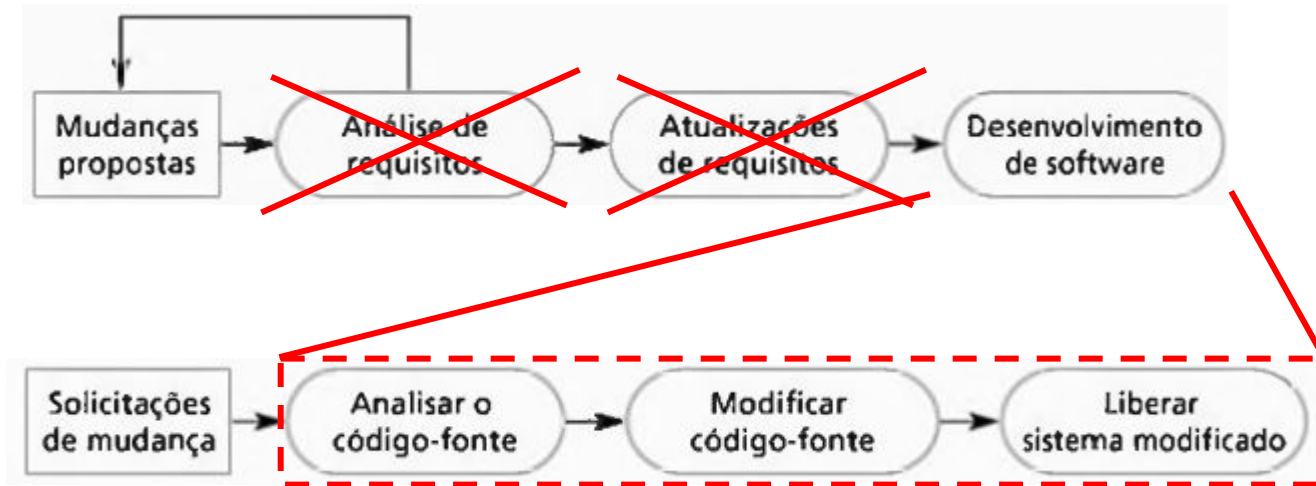


Processo de evolução de software



Correções de emergência em um software

- Quando uma proposta de mudança é relacionada a um problema urgente (**ex:** falha de segurança, defeito grave no sistema, etc), muitas vezes não podemos seguir o fluxo normal de correção abaixo:



Correções de emergência em um software

- Correções de emergência devem ser revistas no futuro para que:
 - Uma solução mais cuidadosa seja desenvolvida em seu lugar
 - Requisitos e documentação sejam atualizados para refletir a correção
 - **Objetivo:** evitar de termos um código implementado que não condiz com os requisitos e documentação atuais do sistema



Dinâmica da evolução de programas

- É o estudo da mudança de sistemas
 - Lehman e Belady (1985) realizaram vários estudos empíricos sobre a mudança de sistemas
 - Eles propuseram as '**Leis de Lehman**'
 - **Leis de Lehman:** leis que provavelmente são verdadeiras para todos os tipos de sistemas de software de grandes organizações (sistemas E-type)
 - **Sistemas E-type:** são aqueles nos quais os requisitos estão mudando constantemente para refletir as necessidades dos negócios

Leis de Lehman

Lei	Descrição
Mudança continua	Um programa usado em um ambiente do mundo real deve necessariamente mudar, ou se torna progressivamente menos útil nesse ambiente.
Aumento da complexidade	Como um programa em evolução muda, sua estrutura tende a tornar-se mais complexa. Recursos extras devem ser dedicados a preservar e simplificar a estrutura.
Evolução de programa de grande porte	A evolução de programa é um processo de autorregulação. Atributos de sistema como tamanho, tempo entre <i>releases</i> e número de erros relatados são aproximadamente invariáveis para cada <i>release</i> do sistema.
Estabilidade organizacional	Ao longo da vida de um programa, sua taxa de desenvolvimento é aproximadamente constante e independente dos recursos destinados ao desenvolvimento do sistema.
Conservação da familiaridade	Durante a vigência de um sistema, a mudança incremental em cada <i>release</i> é aproximadamente constante. Introduzir muitas mudanças em um release implica em mais defeitos.
Crescimento contínuo	A funcionalidade oferecida pelos sistemas tem de aumentar continuamente para manter a satisfação do usuário.
Declínio de qualidade	A qualidade dos sistemas cairá, a menos que eles sejam modificados para refletir mudanças em seu ambiente operacional.
Sistema de <i>feedback</i>	Os processos de evolução incorporam sistemas de <i>feedback</i> multiagentes, <i>multiloop</i> , e você deve tratá-los como sistemas de <i>feedback</i> para alcançar significativa melhoria do produto.

Exercício

Considerando o escopo de evolução de software marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - O fluxo normal para a implementação de uma mudança exige que a proposta da mudança seja feita, seus requisitos sejam analisados e atualizados, para então desenvolver o software com as alterações necessárias. **V**

II - Softwares podem exigir correções de emergência, que ocorrem sem que haja a devida análise e atualização dos requisitos. **V**

III - Uma vez realizada a correção de emergência **não é necessário revisar as alterações** novamente. **F**

IV - Os requisitos e documentação são atualizados **sempre que** correções são realizadas no software. **F**

☐ Somente I, III e IV.

 ☒ Somente I e II.

☐ Somente III e IV.

☐ Somente II.

☐ Nenhuma das alternativas anteriores.

Seguem as assertivas com os ajustes necessários para torná-las VERDADEIRAS:

III - Uma vez realizada a correção de emergência, É NECESSÁRIO revisar as alterações novamente.

IV - Os requisitos e documentação NEM SEMPRE são atualizados quando correções são realizadas no software, pois pode ser necessário realizar correções emergenciais no sistema.

Porque estudar manutenção de software?

- A maior parte do custo de um software (2/3 mais precisamente) está relacionada a manutenção de software que já está em operação
 - Desse custo de manutenção, a maior parte é gasta na implementação de novas funcionalidade do que na correção de *bugs*
 - Isto é, a maior parte do custo da manutenção de software está associada a evolução do sistema para lidar com novos ambientes, bem como novos requisitos

Porque estudar manutenção de software?



Apesar do gráfico ao lado, o custo associado a cada sistema varia muito dos requisitos do sistema e de onde e como ele será usado (domínio de aplicação)

Ex: Em sistemas embutidos de tempo real,
Custos de manutenção =
4 x custos de desenvolvimento



Ex: Já em sistemas de aplicação de negócio (comerciais)
custo de manutenção =
desenvolvimento



Manutenção de Software

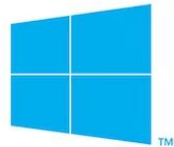
- Processo geral de mudança em um sistema depois que ele é liberado para uso
- Mudanças são implementadas por meio da modificação de componentes do sistema existente e, quando necessário, por meio da adição de novos componentes.
- Classificação de processos de manutenção de software:
 - Correção de defeitos
 - Adaptacao ambiental
 - Adição de funcionalidade

Correção de defeitos

- Ocorre quando há necessidade de corrigir algum defeito no sistema
- Diferentes tipos de erros tem custo para correção diferente:
 - **Erros de requisitos (custo alto)**
 - Requer reprojetar o sistema inteiro (ou de boa parte dele)
 - **Erros de projeto (custo médio)**
 - Requer reescrever vários componentes de programa
 - **Erros de codificação (custo baixo)**
 - Requer apenas uma reimplementação da solução

Adaptação ambiental

- Ocorre quando o sistema deve ser modificado para se adaptar a mudanças em seu ambiente (ex: hardware, sistema operacional, software de apoio)
 - **Ex:** Google Chrome surgiu como aplicação do Windows e, somente depois, sofreu uma **adaptação** para rodar no Linux / Mac OSx



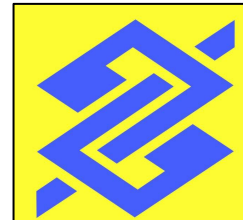
Windows®



Linux

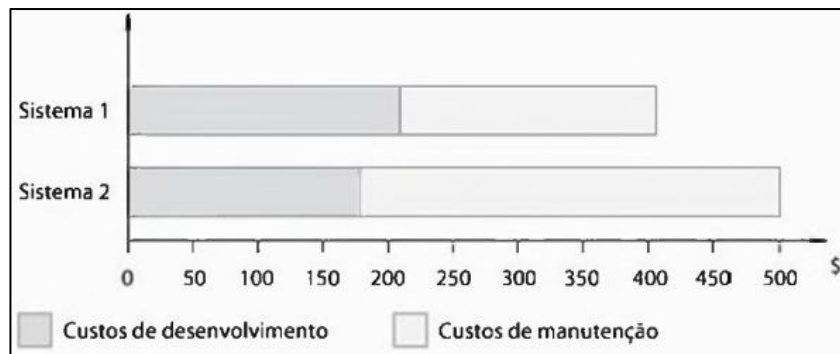
Adição de funcionalidade

- Ocorre quando os requisitos de sistema mudam em resposta às mudanças organizacionais ou de negócios
 - **Ex:** Com o surgimento dos caixas eletrônicos (ATMs), bancos passaram a permitir que seus clientes façam saques e transferências sem a interferência de funcionários do banco (caixas)
- A escala de mudanças necessárias para o software é, frequentemente, muito maior do que para os outros tipos de manutenção



Custos de Manutenção de Software

- Geralmente, vale a pena investir esforços no projeto e implementação de um sistema para redução de custos de mudanças futuras
 - Adicionar uma nova funcionalidade após o sistema estar em operação é caro
 - Necessário tempo para aprender o sistema e analisar o impacto



Gastou-se mais para desenvolver o sistema 1

No entanto, o **custo final do sistema 1 foi menor** do que o do sistema 2

Isto é, o **sistema 1** foi projetado para ser **mais fácil de dar manutenção** (evoluir o sistema)

Fatores que interferem nos custos de Manutenção de Software

- Há muitos fatores que interferem no custo de manutenção de um sistema que está em operação, sendo os principais:
 - **Estabilidade da equipe**
 - **Más práticas de desenvolvimento**
 - **Qualificações de pessoal**
 - **Idade do programa e estrutura**

Fatores que interferem nos custos de Manutenção de Software

- **Estabilidade da equipe**

- Equipes de desenvolvimento mudam frequentemente. Novos engenheiros podem não conhecer o código do sistema.
- Gasta-se um tempo considerável para compreender o sistema antes de dar manutenção nele

- **Más práticas de desenvolvimento**

- Contrato de desenvolvimento firmado com empresa diferente da empresa que irá dar manutenção ao software
- Não há incentivo para a equipe de desenvolvimento escrever um software manutenível

Fatores que interferem nos custos de Manutenção de Software

- **Qualificações de pessoal**

- Sistemas antigos podem ser escritos em linguagens obsoletas de programação (ex: Cobol, Delphi, Fortran)
 - Desenvolvedores precisam aprender a linguagem nova antes de dar manutenção no sistema
- Além disso, o processo de manutenção é visto como um processo menos qualificado do que o desenvolvimento
 - Logo, ele é atribuído a desenvolvedores mais jovens e inexperientes

Fatores que interferem nos custos de Manutenção de Software

- **Idade do programa e estrutura**

- A estrutura de um programa tende a se degradar conforme alterações são feitas no sistema
 - **Consequência:** os sistemas tornam-se mais difíceis de serem entendidos e alterados
- Sistemas antigos não seguem as técnicas modernas de engenharia de software
 - Documentações de sistema inconsistentes ou inexistentes
 - Programas ininteligíveis (**ex:** maioria dos programas assembly)

Exercício

Considerando o escopo de evolução de software marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Os **erros de projeto** são aqueles com maior custo para correção, pois eles requerem a reescrita de vários componentes de programa. **F**

II - A correção de defeitos é parte do processo de manutenção de software. Nessa etapa, erros de requisitos, de projeto e de codificação são corrigidos. **V**

III - O custo associado a correção de defeitos depende do tipo de erro a ser eliminado. **V**

IV - A manutenção de software engloba a modificação e/ou adição de novos componentes ao sistema. **V**

☐ Todas as assertivas são verdadeiras.

☐ Somente II e IV.

☐ Somente II e III.

☐ Somente I, III e IV.

 ☒ Nenhuma das alternativas anteriores.

Seguem as assertivas com os ajustes necessários para torná-las VERDADEIRAS:

I - Os erros de REQUISITOS são aqueles com maior custo para correção, pois eles requerem a reescrita do sistema inteiro, ou de boa parte dele.

Reengenharia de software

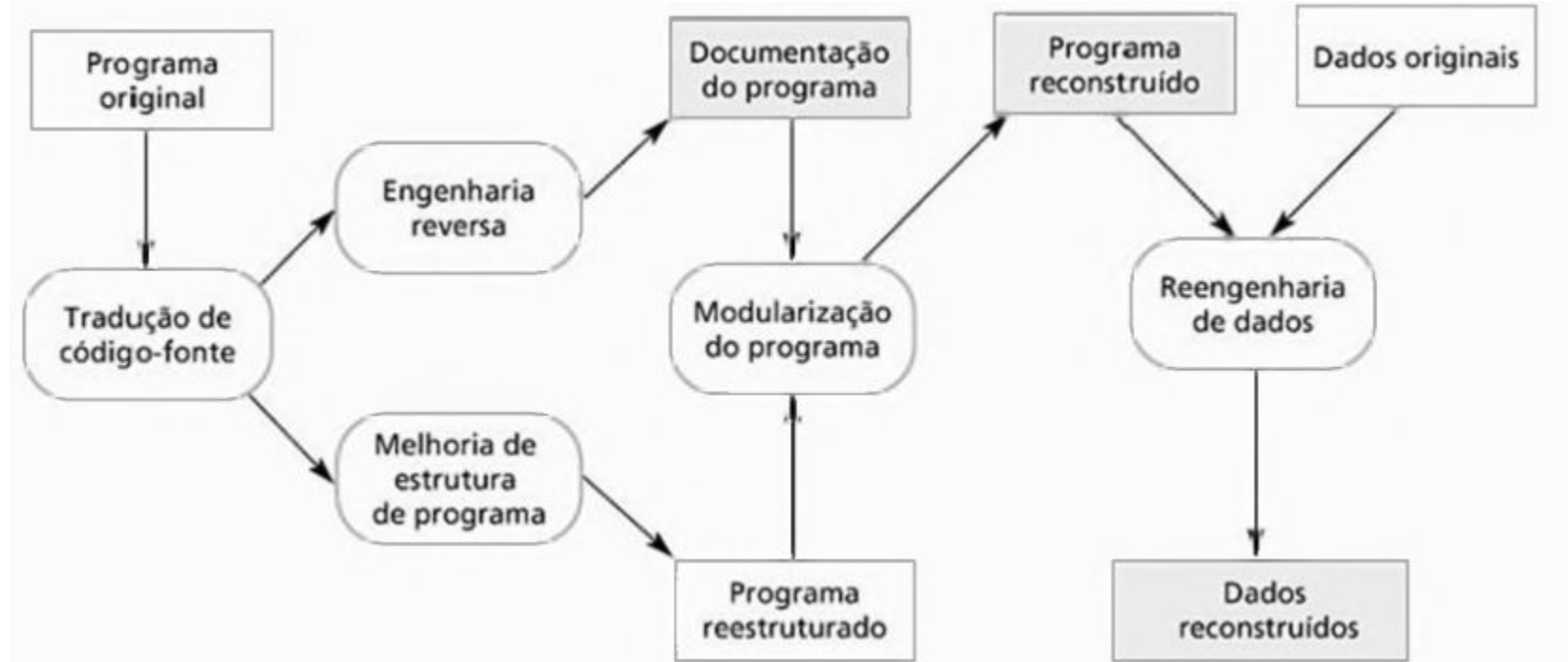
- Surgiu para resolver os problemas de manutenibilidade que mencionamos anteriormente
- Permite melhorar um código existente, em termos de estrutura, inteligibilidade e documentação
- Envolve as tarefas de:
 - Redocumentação de sistema
 - Reconstrução da arquitetura de sistema
 - Mudança de linguagem de programação para uma linguagem moderna
 - Modificações e atualizações da estrutura e dos dados de sistema

A reengenharia de software não altera a funcionalidade do software!
(Se evita grandes mudanças na arquitetura do sistema)

Porque reengenharia de software? Porque não realizar a substituição completa do sistema?

- **Vantagens da reengenharia** (quando comparada a substituição do sistema):
 - **Risco reduzido:** Existe um alto risco em desenvolver um software crítico de negócios do zero
 - **Ex:** erros na especificação, problemas de desenvolvimento, atrasos, custos adicionais
 - **Custo reduzido:** a reengenharia pode ter custo significativamente menor do que o desenvolvimento de um novo software
 - **Ex:** Ulrich (1990) cita um exemplo de um sistema comercial cujos custos de reimplementação foram estimados em **50 milhões** de dólares, sendo que o sistema foi **reconstruído com sucesso por 12 milhões** de dólares

Atividades do processo de reengenharia

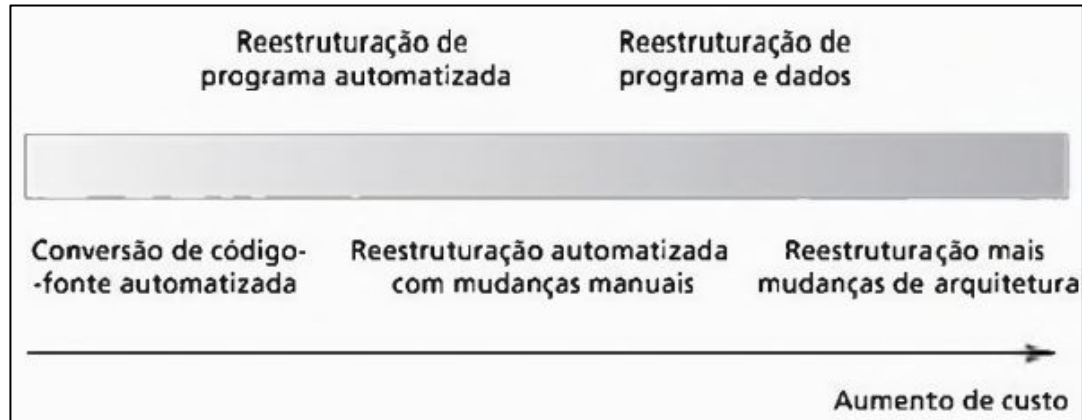


Atividades do processo de reengenharia

- **Tradução de código-fonte:** programa é convertido de uma linguagem antiga para uma linguagem mais moderna ou versão mais atualizada
- **Engenharia reversa:** programa é analisado e as informações são extraídas a partir dele.
- **Melhoria de estrutura de programa:** análise e modificação da estrutura de controle do programa para que ele se torne mais fácil de ler e entender.
- **Modularização de programa:** Partes relacionadas do programa são agrupadas, e redundâncias são removidas.
- **Reengenharia de dados:** dados processados pelo programa são alterados para refletir as mudanças de programa (ex: redefinir estruturas de dados, bancos de dados, etc)

Atividades do processo de reengenharia

- Nem todas as atividades de reengenharia são necessárias para um sistema
 - O custo da reengenharia depende da extensão do trabalho a ser feito
 - Isto é, dependendo do caso, pode ser que substituir o software seja mais viável economicamente (menos custoso)



Manutenção preventiva (por refatoração)

- **Refatoração:** é o processo de fazer melhorias em um programa para diminuir a degradação gradual resultante das mudanças
 - *“Modificar um programa para melhorar sua estrutura e reduzir sua complexidade, tornando-o mais fácil de compreender”*
 - Não deve adicionar funcionalidades. Deve-se concentrar nas melhorias do programa existente.
- **Refatorar não é a mesma coisa que fazer a reengenharia do software**
 - A refatoração é um processo contínuo de melhoria
 - A reengenharia ocorre de maneira pontual, após a criação do sistema

Manutenção preventiva (por refatoração)

- **Exemplos de problemas que a refatoração melhora (ou corrige):**
 - Código duplicado
 - Métodos longos
 - Aglutinação de dados
 - Generalidade especulativa

Código duplicado

- Mesmo código ou um código muito semelhante incluído em diferentes lugares do programa

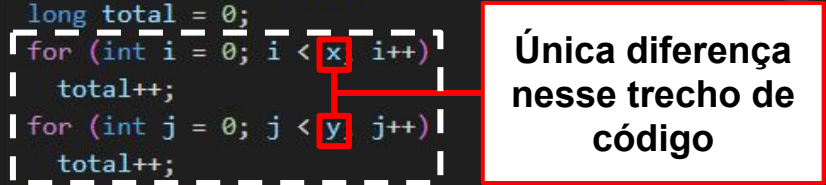
```
function calculateTax(subtotal, country, state, taxrates) {  
  let taxrate;  
  if(country === 'US') {  
    taxrate = taxrates[state];  
  } else {  
    taxrate = taxrates[country];  
  }  
  return subtotal + subtotal * taxrate;  
}  
  
function findTimeZone(country, state, zones) {  
  let timezone;  
  if(country === 'US') {  
    timezone = zones[state];  
  } else {  
    timezone = zones[country];  
  }  
  return timezone;  
}
```

Poucas diferenças
nos dois códigos

Métodos longos

- Métodos longos devem ser reprojutados como uma série de métodos mais curtos, para facilitar a sua manutenção e entendimento

```
1
2 // somar dois numeros
3 // -- Restricao: so podemos somar de 1 em 1
4 long add(int x, int y) {
5     long total = 0;
6     for (int i = 0; i < x; i++)
7         total++;
8     for (int j = 0; j < y; j++)
9         total++;
10    return total;
11 }
12
13
14
15
16
```



Única diferença
nesse trecho de
código

Métodos longos

- Métodos longos devem ser reprojutados como uma série de métodos mais curtos, para facilitar a sua manutenção e entendimento

The diagram illustrates the refactoring of a long function into smaller, more maintainable functions. It shows two versions of code side-by-side, with annotations and arrows indicating the transformation.

Original Code (Left):

```
1 // somar dois numeros
2 // -- Restricao: so podemos somar
3
4 long add(int x, int y) {
5     long total = 0;
6     for (int i = 0; i < x; i++)
7         total++;
8     for (int j = 0; j < y; j++)
9         total++;
10    return total;
11 }
```

Refactored Code (Right):

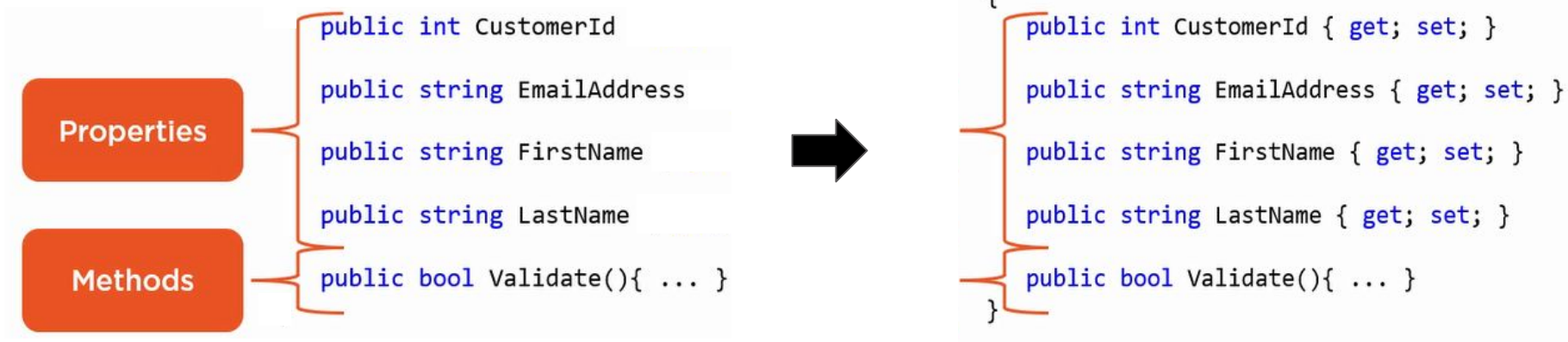
```
18 // somar dois numeros (versao refatorada)
19 // -- Restricao: so podemos somar de 1 em 1
20
21 long add_refatorado(int x, int y) {
22     long total = 0;
23     total = add_parcial(total, x);
24     total = add_parcial(total, y);
25     return total;
26 }
27
28 long add_parcial(long total, int k) {
29     for (int i = 0; i < k; i++)
30         total++;
31     return total;
32 }
```

Annotations:

- A red box highlights the inner loop `for (int j = 0; j < y; j++) total++;` in the original code.
- A white box labeled "Refatorando a função ..." (Refactoring the function ...) has an arrow pointing from the red box to the `add_parcial` function in the refactored code.
- A red box highlights the `for (int i = 0; i < k; i++) total++;` loop in the `add_parcial` function.

Aglutinação de dados

- Unir dados (campos em classes, parâmetros em métodos) que são utilizados em vários lugares de um programa

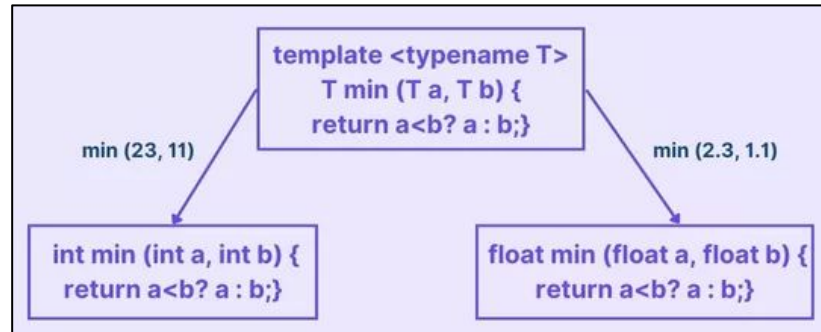


Todas as propriedades e métodos dizem a respeito a uma entidade
“Consumidor”

Aglutinando na
classe
Consumidor ...

Generalidade especulativa

- Tentativa dos desenvolvedores de tornar um programa mais genérico
 - **Objetivo:** atender possíveis necessidades no futuro (*"futurologia especulativa"*)
- Essa generalidade muitas vezes não é necessária
 - O programa pode nunca precisar desses recursos
 - O excesso de generalidade pode tornar o código mais complexo
 - **Ex:** uso excessivo de Templates (Java / C++ / C#)



Exercício

Considerando o escopo de evolução de software marque a alternativa que contém **somente** as assertivas VERDADEIRAS.

I - Realizar a manutenção preventiva de um software inclui a refatoração, que objetiva fazer melhorias para diminuir a degradação gradual de um software. **V**

II - A manutenção preventiva **assim como a reengenharia de software, é um processo contínuo.** **F**

III - **A manutenção preventiva pode adicionar funcionalidade** aos software, caso elas possam diminuir a degradação residual do sistema. **F**

IV - Não há grandes distinções dos objetivos da refatoração em relação aos da reengenharia de software. **V**

☐ Somente I e III.

☐ Somente I e II.

☐ Somente I, II e IV.

☐ Somente III e IV.

☒ Nenhuma das alternativas anteriores.

Seguem as assertivas com os ajustes necessários para torná-las VERDADEIRAS:

II - A manutenção preventiva é um processo contínuo. A reengenharia de software é realizada APÓS A CRIAÇÃO do sistema (de maneira pontual).

III - A manutenção preventiva NÃO pode adicionar funcionalidade aos software.

Referencial Bibliográfico

- SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Addison-Wesley, 2003.
- PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995.
- JUNIOR, H. E. **Engenharia de Software na Prática**. Novatec, 2010.

Obrigado!

- Perguntas?

