



Engenharia de Software

Módulo 02

Prof. Daniel Caixeta



Conteúdo programático

01 Processos de *software*

1.1. Introdução.

1.2. As atividades

1.2.1. A especificação [...].

1.2.2. O projeto e implementação [...].

1.2.3. A validação [...].

1.2.4. A evolução do *software*.

1.2.5. Concluindo [...].

1.3. As características.



02 Modelos de processos

2.1. Conceitos.

2.2. Os tipos de modelos.

2.2.1. Cascata.

2.2.2. Incremental.

2.2.3. Evolucionário. (Prototipação e Espiral).

2.2.4. Concorrente.

2.2.5. Baseado em componentes ou orientado ao reuso.

2.2.6. Métodos formais.

01. Processos de *Software*

1.1. Introdução

1.2. As atividades [...]

1.2.1. A especificação [...].

1.2.2. O projeto e implementação [...].

1.2.3. A validação [...].

1.2.4. A evolução do *software*.

1.2.5. Concluindo [...].

1.3. As características [...]





1.1. Introdução

Conceitos iniciais [...]

1.1. Processos de *software*: Introdução

- Um processo de *software* é um conjunto de atividades relacionadas que levam à produção de um produto de *software*. (SOMMERVILLE, 2018).
- Essas atividades podem envolver desde a criação de um programa executável até aplicações desenvolvidas por meio de extensão e modificação de sistemas já existentes, ou ainda por meio da configuração e integração de componentes que pertencem ao sistema computacional.



- Já para Pressman (2011), processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum produto de trabalho (*work product*).
 - Atividade se refere ao esforço para atingir um objetivo amplo. *E.g.*, comunicar-se com os *stakeholders*.
 - Ação envolve um conjunto de tarefas que resultam num artefato de *software* fundamental. *E.g.*, um projeto de arquitetura para aplicativos *mobiles*.
 - Tarefa se concentra em um objetivo, porém, bem definido, *e.g.*, realizar teste de unidades e produzir resultados tangíveis.
- Uma metodologia (*framework*) de processo estabelece os alicerces para a engenharia de software, por meio da identificação das atividades estruturais aplicáveis a todos os projetos de *software*, independentemente de tamanho ou complexidade.
- Além disso, a metodologia de processo engloba um conjunto de atividades de apoio (*umbrella activities* - abertas) aplicáveis em todo o processo de *software*.

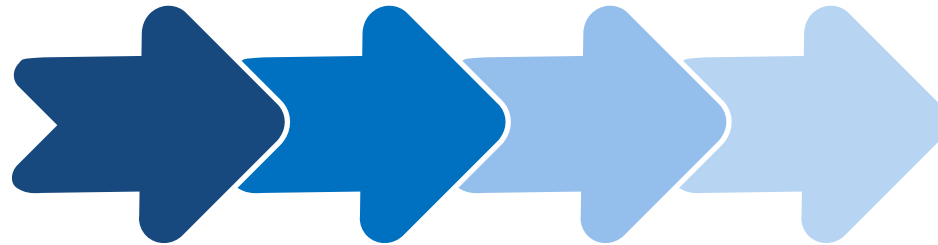


1.2. As atividades [...]

Das especificações às evoluções [...]

1.2. As atividades [...]

- Para Sommerville (2018), existem inúmeros processos que se diferenciam, mas todos incluem essas quatro atividades, consideradas fundamentais para a engenharia de software:



01 Especificação
A funcionalidade e as restrições a seu funcionamento devem ser definidas.

02 Projeto e implementação
O *software* deve ser produzido para atender às especificações.

03 Validação
O *software* deve ser validado para garantir que atenda às demandas do cliente.

04 Evolução
O *software* deve evoluir para atender às necessidades de mudança dos clientes.

1.2.1. A especificação [...]

- Especificação de software ou engenharia de requisitos é:

O processo de compreensão e definição dos serviços requisitados do sistema e identificação de restrições relativas à operação e ao desenvolvimento do sistema. (SOMMERVILLE, 2018).

- É um estágio particularmente crítico do processo de *software*, pois erros nessa fase inevitavelmente geram problemas no projeto e na implementação do sistema. (*ibidem*).
- Existem quatro atividades principais deste processo:
 1. Estudo de viabilidade: Faz-se estimativas acerca da possibilidade de se satisfazerem as necessidades do usuário [...]. O estudo considera se o sistema proposto será rentável a partir do ponto de vista do negócio [...]. O resultado deve informar a decisão de avançar ou não. (*ibidem*).

2. Elicitação e análise de requisitos: É o processo de derivação dos requisitos por meio da observação de sistemas já existentes [...]. Essa parte envolve o desenvolvimento de um ou mais modelos e protótipos, os quais ajudam a entender o sistema a ser especificado. (SOMMERVILLE, 2018).
3. Especificação de requisitos: Existem dois tipos de requisitos que podem ser incluídos nesse documento. Requisitos do usuário - São declarações abstratas dos requisitos do sistema para o cliente e usuário final, e, Requisitos de sistema - São descrições mais detalhadas das funcionalidades a serem desenvolvidas. (*ibidem*).
4. Validação de requisitos: Verifica os requisitos quanto ao realismo, consistência e completude. Durante esse processo, erros são inevitavelmente descobertos, e em seguida, a documentação é corrigida incluindo as correção. (*ibidem*).

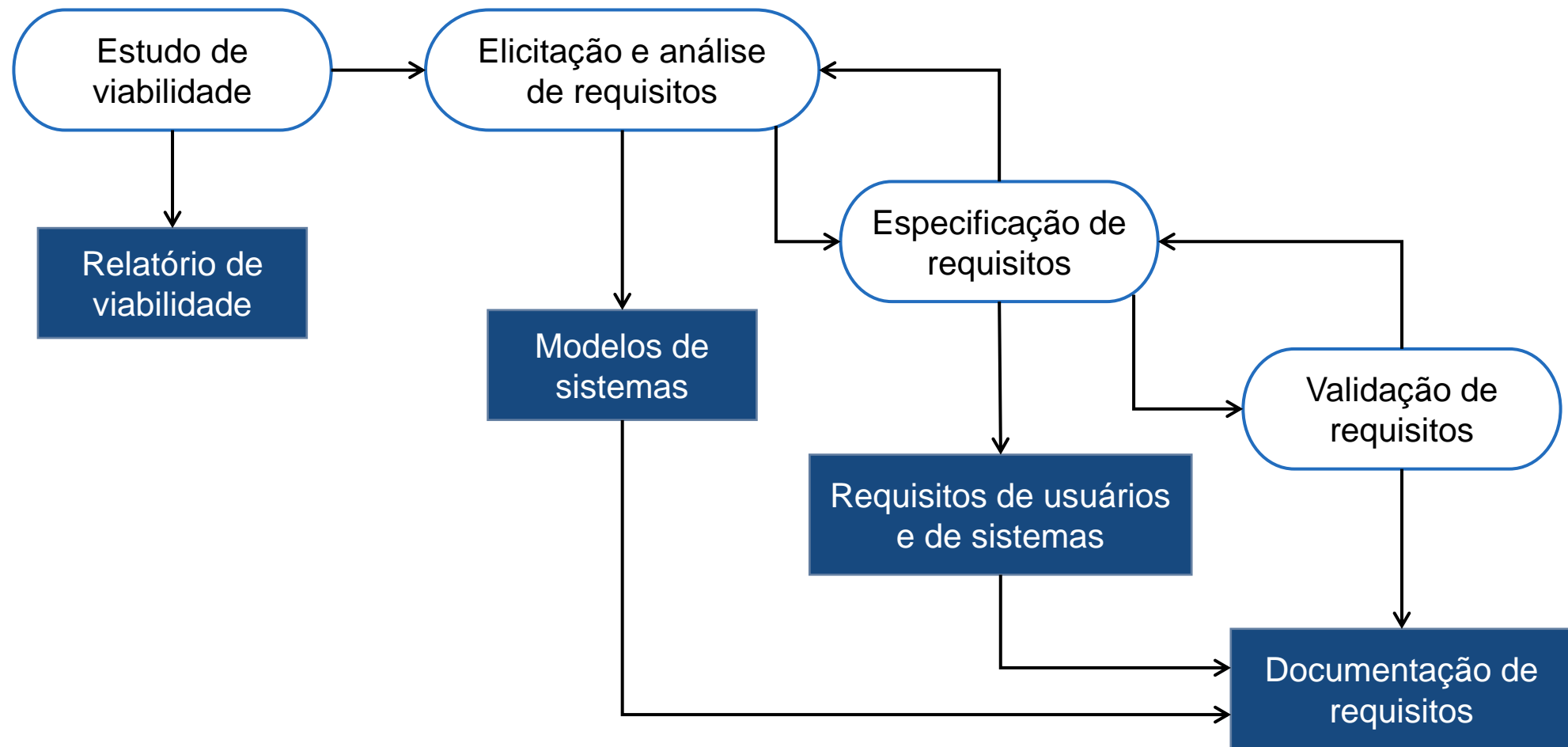


Figura 1. Fluxo dos requisitos da engenharia de processos na especificação de *software* de acordo com suas principais atividades. (SOMMERVILLE, 2018).

1.2.2. O projeto e a implementação [...]

- Estágio de implementação do desenvolvimento de *software* é:

O processo de conversão de uma especificação do sistema em um sistema executável. Sempre envolve processos de projeto e programação de *software*, mas, se for usada uma abordagem incremental para o desenvolvimento, também pode envolver o refinamento da especificação do *software*. (SOMMERVILLE, 2018).

- Já projeto de *software* é:

Uma descrição da estrutura do *software* a ser implementado, dos modelos e estruturas de dados usados pelo sistema, das interfaces entre os componentes do sistema e, às vezes, dos algoritmos usados. (*ibidem*).

- Os projetistas não chegam a um projeto final imediatamente, mas desenvolvem-no de forma iterativa e acrescentando ao longo do desenvolvimento as formalidade e os detalhes, realizando revisões constantes.

- Uma análise importante:

- ✓ A maioria dos *softwares* interage com outros sistemas de *software*, incluindo o sistema operacional, banco de dados, *middleware* e outros aplicativos. Estes formam a 'plataforma de software', o ambiente em que o *software* será executado. (SOMMERVILLE, 2018).
- ✓ Informações sobre essa plataforma são entradas essenciais para o processo de projeto, pois os projetistas devem decidir a melhor forma de integrá-la ao ambiente do *software*. (*ibidem*).
- ✓ A especificação de requisitos é uma descrição da funcionalidade que o *software* deve oferecer, e seus requisitos de desempenho e confiança, e.g., se o sistema for para processamento de dados, a descrição desses dados deve ser incluída na especificação da plataforma. [...].

- Existem quatro atividades principais que podem ser parte do processo de projeto de sistemas de informação:
 1. Projeto de arquitetura: Identifica a estrutura geral do sistema, os componentes principais (subsistemas ou módulos), seus relacionamentos e como eles são distribuídos. (SOMMERVILLE, 2018).
 2. Projeto de interface: Define as interfaces entre os componentes do sistema. (*ibidem*).
 3. Projeto de componente: Toma cada componente do sistema e projeta seu funcionamento. Trata-se de uma simples declaração de funcionalidade, com projetos específicos para cada programador. (*ibidem*).
 4. Projeto de banco de dados: Projeta as estruturas de dados do sistema e como eles devem ser representados em um B.D. (SOMMERVILLE, 2018).

- Estão relacionados com a produção de modelos gráficos que, em muitos casos, geram códigos automaticamente a partir desses modelos. *E.g.*, o Desenvolvimento Dirigido a Modelos (MDD, do inglês *Model-Driven Development*) [...]. (SCHMIDT, 2006 *apud* SOMMERVILLE, 2018), em que os modelos de *software* são criados em diferentes níveis de abstração.
- Em MDD, há maior ênfase nos modelos de arquitetura [...], ou seja, esses são desenvolvidos com o máximo de detalhe para que o sistema executável seja gerado a partir deles e de forma automática.
- Segundo Sommerville (2018), ferramentas de desenvolvimento podem ser usadas para gerar um “esqueleto” de um programa a partir do projeto. Isso inclui o código e a implementação das interfaces e, em muitos casos, o desenvolvedor precisa apenas acrescentar detalhes operacionais de cada componente do programa.

- Geralmente, os programadores fazem testes no código, o que muitas vezes, revela defeitos que devem ser corrigidos. Isso é chamado *debugging*. Testes de defeitos e *Debugging* são processos diferentes.
 - ✓ Testes estabelecem a existência de defeitos;
 - ✓ *Debugging* diz respeito à localização e correção desses defeitos.
- Quando se realiza um processo de *debugging*, o desenvolvedor cria hipóteses sobre o comportamento observável do programa e, em seguida, testa essas hipóteses na esperança de encontrar um defeito que tenha causado uma saída anormal. (SOMMERVILLE, 2018).
- Já o teste das hipóteses pode envolver o rastreo manual do código do programa, bem como exigir novos casos de teste para localização do problema. (*ibidem*).
- Aqui ferramentas interativas de depuração são de grande auxílio para os testes em códigos.

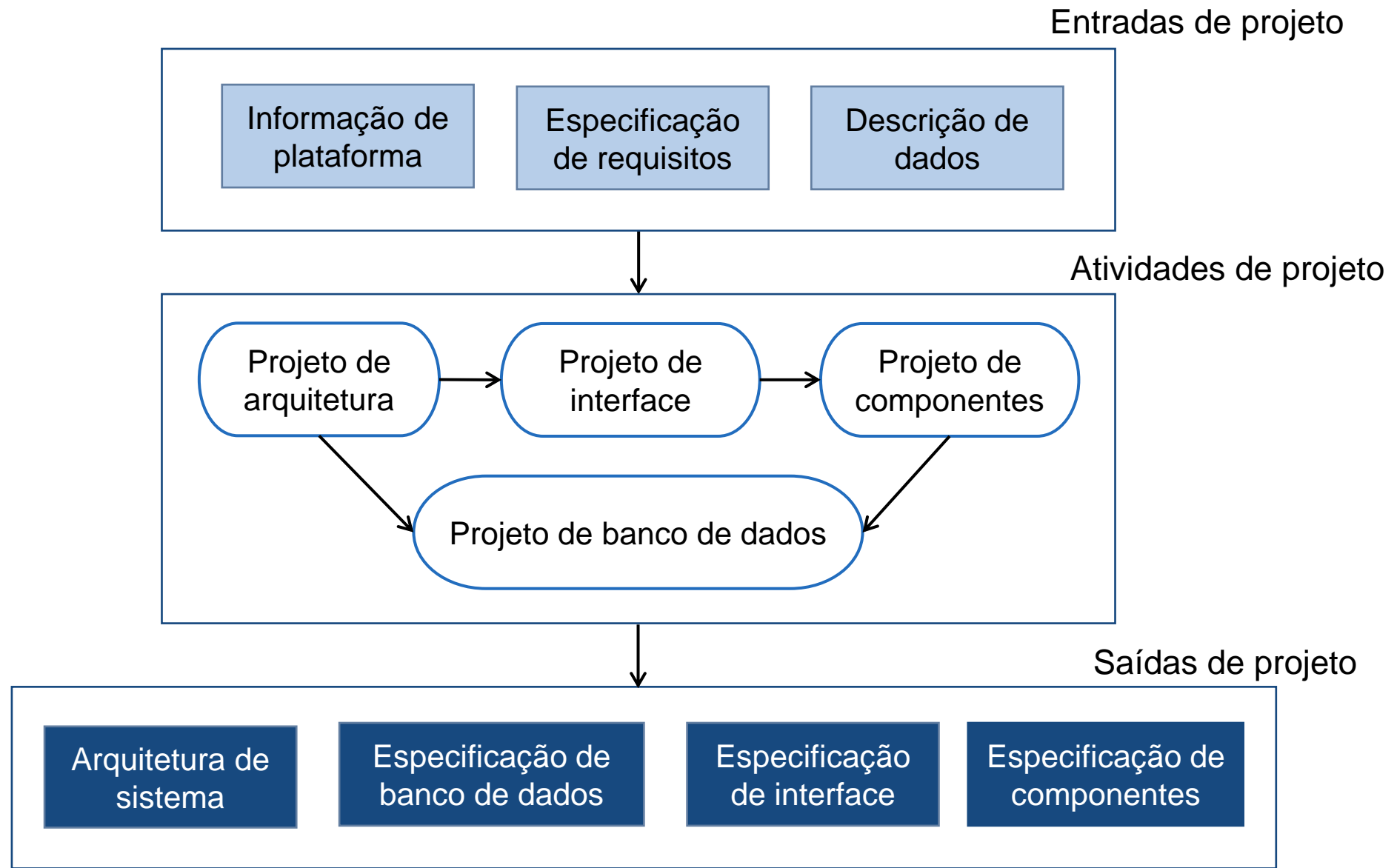


Figura 2. Modelo geral do processo de projetos. (SOMMERVILLE, 2018).

1.2.3. A validação [...]

- Validação de software ou Verificação e Validação (V&V), tem como objetivo mostrar que um *software* se adequa a suas especificações ao mesmo tempo que satisfaz as necessidades do cliente.
- Os estágios desses processos são:
 1. Testes de desenvolvimento: Os componentes do sistema são testados pelas pessoas que o desenvolveram [...]. Esses são normalmente testados como entidades simples (funções, classes de objetos ou agrupamentos). (SOMMERVILLE, 2018).
 2. Testes de sistema: Esse processo procura por erros resultantes das interações inesperadas entre componentes. Também visa demonstrar que o sistema satisfaz os requisitos funcionais e não funcionais. (SOMMERVILLE, 2018).

3. Testes de aceitação: É o estágio final dos testes, antes que o sistema seja aceito para uso operacional. É testado com dados fornecidos pelo cliente (não usa dados dos testes simulados). O teste de aceitação pode revelar erros e omissões na definição dos requisitos do sistema, pois dados reais “provocam” o sistema de formas diferentes dos dados de teste. Os testes de aceitação também podem revelar problemas de requisitos em que os recursos do sistema não atendam às necessidades do usuário [...]. (SOMMERVILLE, 2018 *adaptado* CAIXETA, 2020).

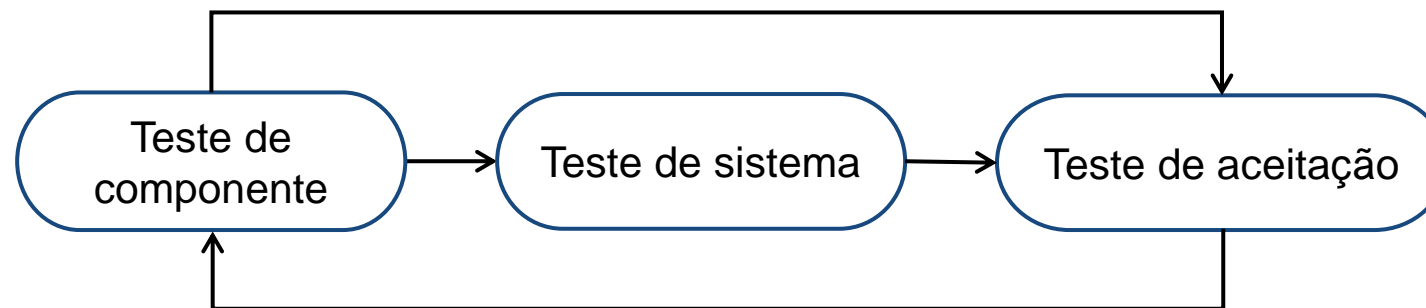


Figura 3. Estágios de testes. (SOMMERVILLE, 2018).

- Dicas importantes:

- Quando a abordagem incremental é adotada, cada incremento deve ser testado enquanto é desenvolvido, sendo que esses testes devem ser baseados nos requisitos para esse incremento.
- Quando o processo de *software* é dirigido a planos, e.g., para o desenvolvimento de sistemas críticos, os testes são formados por um conjunto de planos de testes.

(SOMMERVILLE, 2018 *adaptado* CAIXETA, 2020).

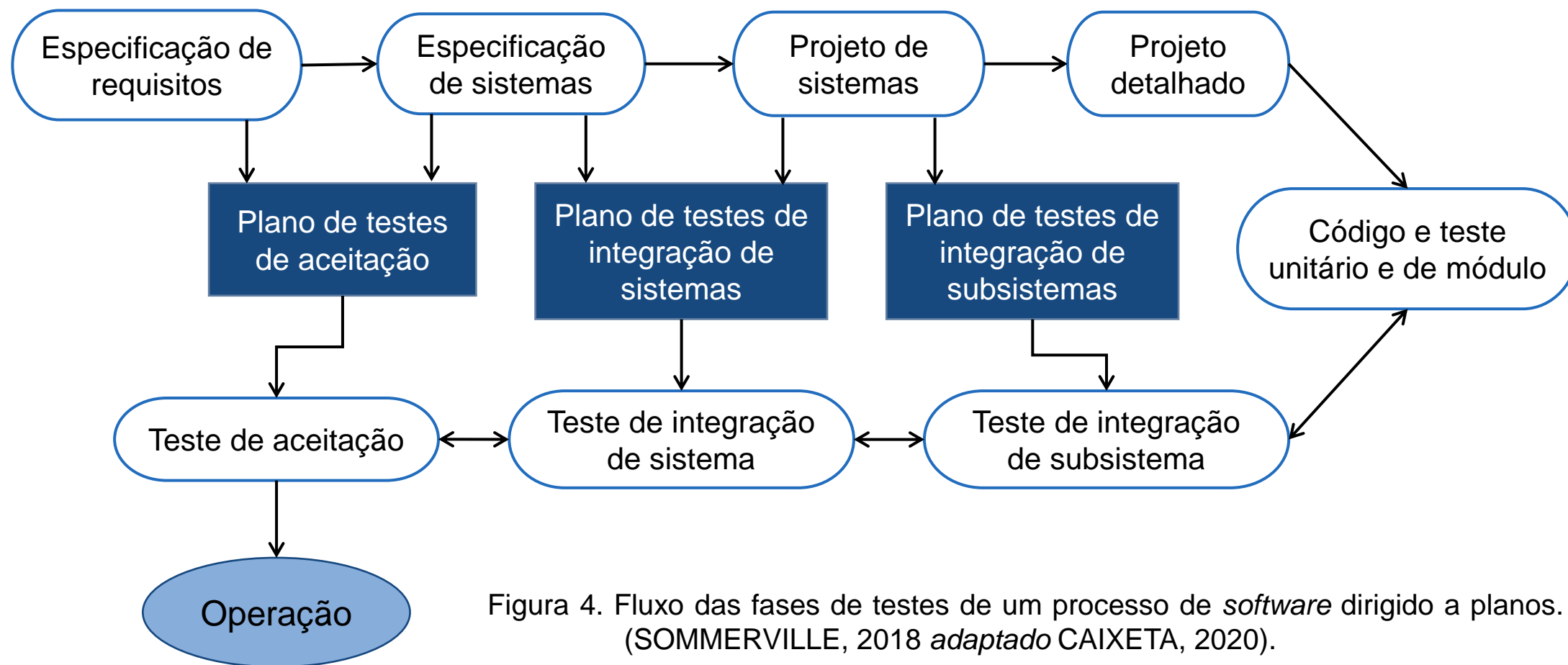


Figura 4. Fluxo das fases de testes de um processo de *software* dirigido a planos. (SOMMERVILLE, 2018 *adaptado* CAIXETA, 2020).

1.2.4. A evolução do *software*

- Para Sommerville (2018), a flexibilidade dos sistemas de *softwares* é uma das principais razões pelas quais vêm sendo cada vez mais incorporados em sistemas grandes e complexos. Uma vez tomada a decisão pela fabricação do *hardware*, torna-se caro fazer alterações em seu projeto. Entretanto, as mudanças no *software* podem ser feitas a qualquer momento durante ou após o desenvolvimento do sistema.
- Mesmo grandes mudanças são muito mais baratas do que as correspondentes alterações no *hardware* do sistema.

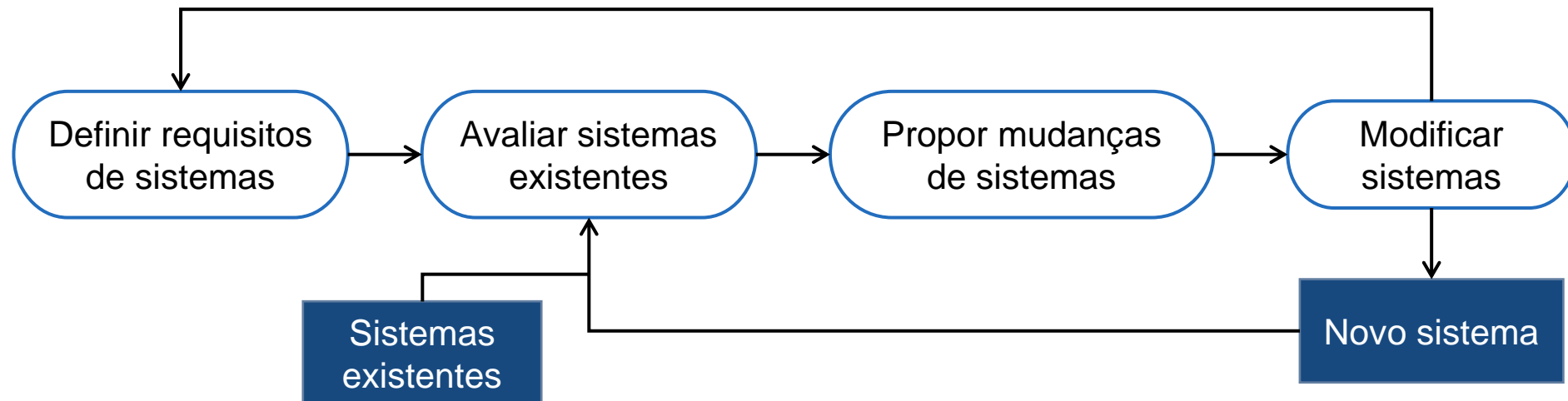


Figura 5. Evolução de um sistema. (SOMMERVILLE, 2018).

1.2.5. Concluindo [...]



- De certa forma, essas atividades fazem parte de todos os processos de *software*.
- Na prática, são atividades complexas entre si, e incluem subatividades como validação de requisitos, projeto de arquitetura, testes unitários, etc.
- Existem também as atividades que dão apoio ao processo, como documentação e gerenciamento de configuração de *software*.

Figura 6. Síntese das principais atividades realizadas no desenvolvimento de *softwares*.

- As descrições dos processos dessas atividades também podem ainda incluir:
 1. Produtos: São os resultados de uma das atividades do processo. *E.g.*, um modelo da arquitetura de *software*, uma interface, um MER, documento de requisitos, diagrama de classes, etc.
 2. Papéis: São as responsabilidades dos *stakeholders* no processo. *E.g.*, Gerente de projeto, Gerente de configuração, Programador, DBA, etc.
 3. Pré e pós-condições: São declarações verdadeiras antes e depois de uma atividade ou da produção de um item ou produto. *E.g.*, para se iniciar o desenvolvimento tornar-se necessária a aprovação pelo cliente dos requisitos funcionais iniciais. (pré-condição).





1.3. As características [...]

Os principais pontos [...]

1.3. As características [...]

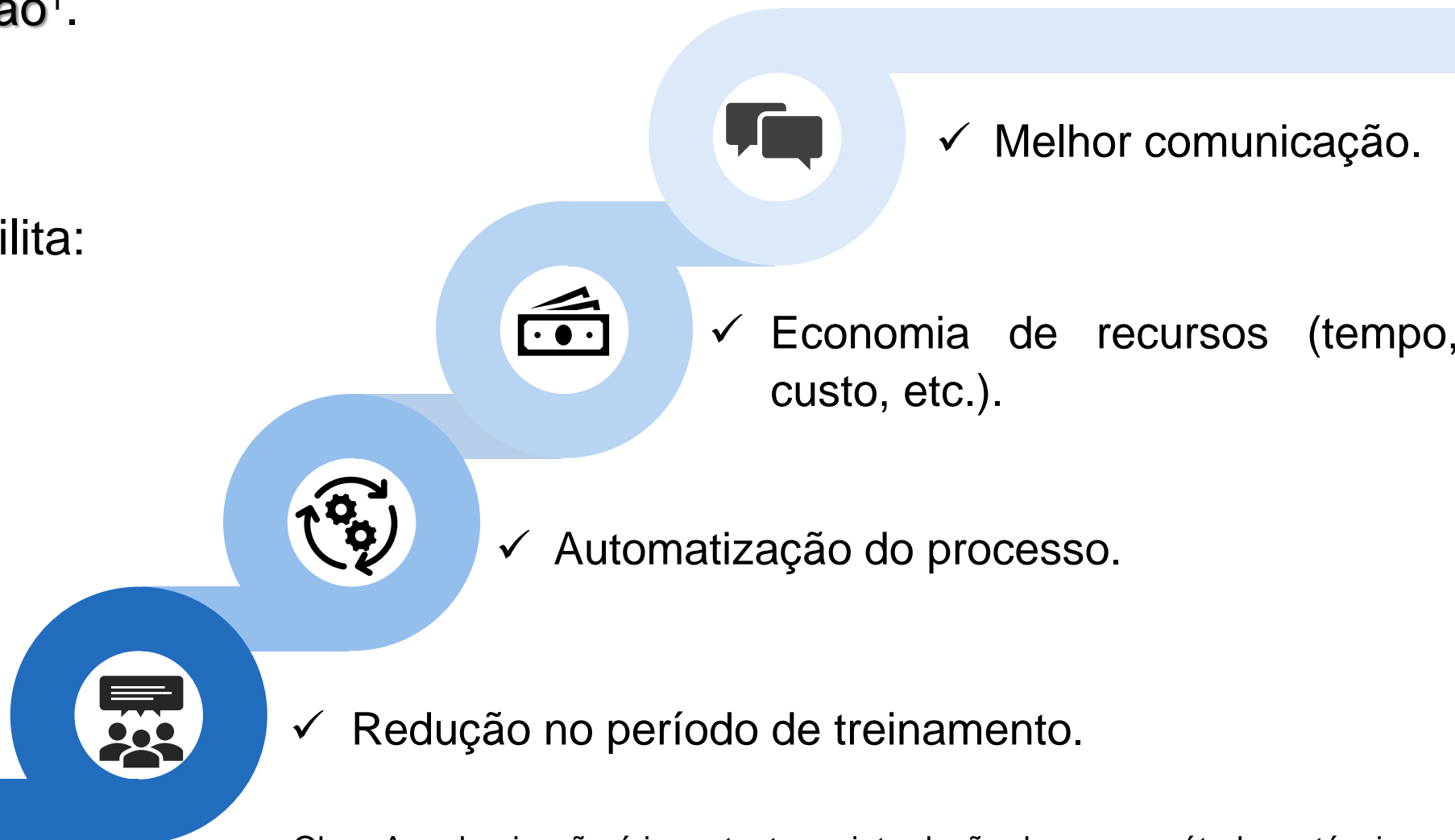
- Os processos de *softwares* são:
 - a. Complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos – Complexidade decisória.
 - b. Não existe um processo ideal, a maioria das organizações desenvolve seus próprios processos – Autoria processual.
 - c. Os processos têm evoluído de maneira a tirarem melhor proveito das capacidades das pessoas, bem como das características específicas do sistema em desenvolvimento – Capacitação profissional e Especificação sistêmica.
 - d. Para sistemas críticos, são necessários processos muito bem estruturado, já para sistemas de negócios, com requisitos que se alteram rapidamente, será mais eficaz processos menos formais e mais flexíveis – Estruturação efetiva.

(SOMMERVILLE, 2018 *adaptado* CAIXETA, 2020)

- Em organizações nas quais os processos são reduzidos, adota-se a padronização¹.



- Isso possibilita:



Obs₁. A padronização é importante na introdução de novos métodos e técnicas, além de boas práticas.

- Segundo Sommerville (2018, *grifo meu*), os processos de *software*, às vezes, são categorizados como:
 - ✓ Processos dirigidos a planos: São aqueles em que todas as atividades são planejadas com antecedência, e o progresso é avaliado por comparação com o planejamento inicial.
 - ✓ Processos ágeis: O planejamento é gradativo, e é mais fácil alterar o processo de maneira a refletir as necessidades de mudança dos clientes.
- De acordo Boehm & Turner (2003 *apud* SOMMERVILLE, 2018), cada abordagem é apropriada para diferentes tipos de *software*, e geralmente, é necessário encontrar um equilíbrio entre os processos dirigidos a planos e os processos ágeis.

02. Modelos de processos

2.1. Conceitos.

2.2. Os tipos de modelos [...].

2.2.1. Cascata.

2.2.2. Incremental.

2.2.3. Evolucionário (Prototipação & Espiral
(Boehm, 1988)).

2.2.4. Concorrente.

2.2.5. Baseado em componentes ou orientado ao
reuso.

2.2.6. Métodos formais.





2.1. Conceitos de Modelos de Processos.

2.1. Modelos de processos [...]

- Conceito:

É uma representação simplificada de um processo de *software*, sendo que, cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele. (SOMMERVILLE, 2018).

Exemplo: Um modelo de atividade do processo pode mostrar as atividades e sua sequência, mas não mostrar os papéis das pessoas envolvidas.

- Para Sommerville (*ibidem*), torna-se necessário observar a execução do sistema na perspectiva de sua arquitetura, ou seja, vemos o *framework* do processo, mas não vemos os detalhes de suas atividades específicas.
- Esses modelos não são descrições definitivas. Pelo contrário, são abstrações que podem ser usadas para explicar diferentes abordagens de desenvolvimento de *software*. (*ibidem*, grifo meu).



2.2. Os tipos de modelos [...]

Segundo Pressman & Sommerville [...]

2.2. Os tipos de modelos (segundo Pressman & Sommerville)

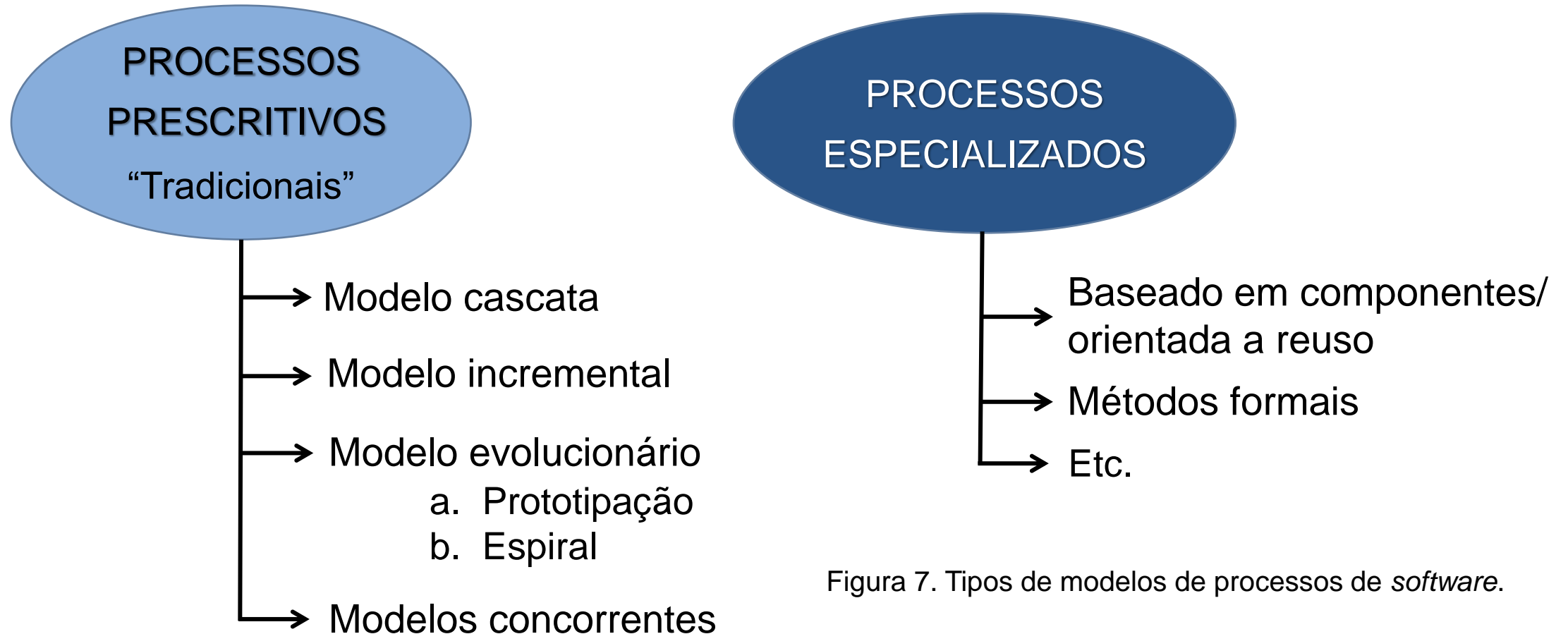
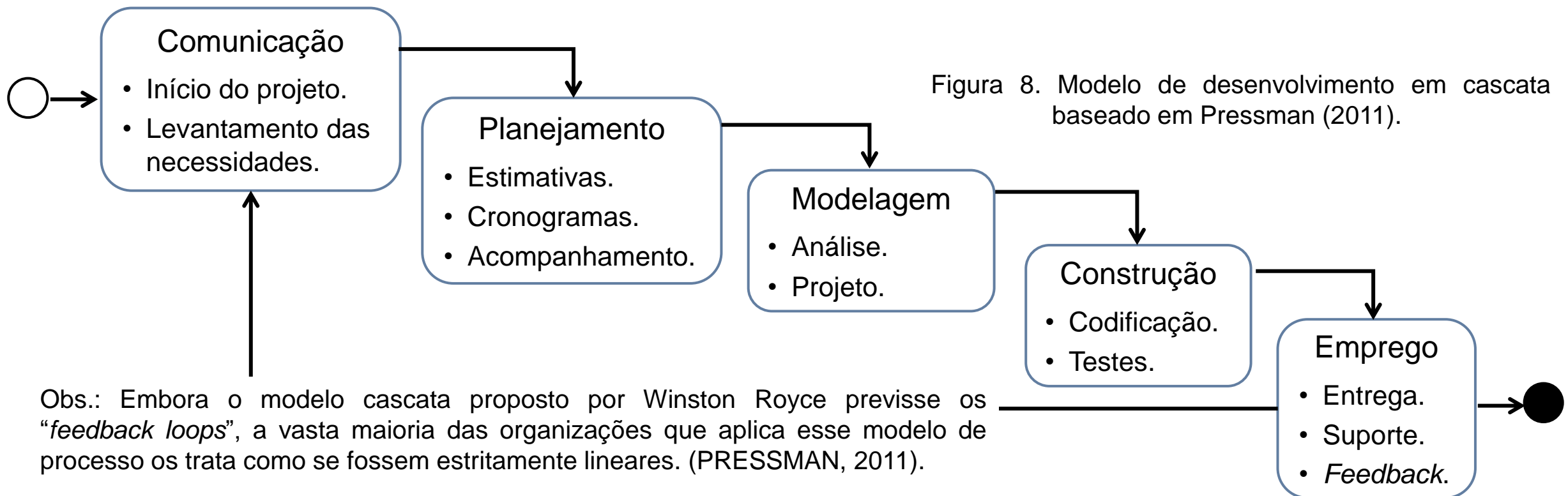


Figura 7. Tipos de modelos de processos de *software*.

2.2.1. Modelo cascata (Ciclo de vida clássico).

- Considera as atividades fundamentais do processo de especificação, desenvolvimento, validação e evolução, e representa cada uma delas como fases distintas, como: especificação de requisitos, projeto de *software*, implementação, teste e assim por diante. (SOMMERVILLE, 2018).



- Desvantagens segundo Pressman (2011)

- i. Projetos reais raramente seguem o fluxo sequencial que o modelo propõe. [...]. Como consequência, mudanças podem provocar confusão à medida que a equipe de projeto prossegue.
- ii. Paciência do cliente. Uma versão operacional do(s) programa(s) não estará disponível antes das etapas finais do projeto. Um erro grave, se não detectado até o programa operacional ser revisto, pode ser desastroso.
- iii. Segundo Bradac (1994, *apud* PRESSMAN, *ibidem*) a natureza linear do ciclo de vida clássico conduz a “estados de bloqueio”, nos quais alguns membros da equipe do projeto têm de aguardar outros completarem tarefas dependentes.

2.2.2. Modelo incremental

- É baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido. (SOMMERVILLE, 2018).

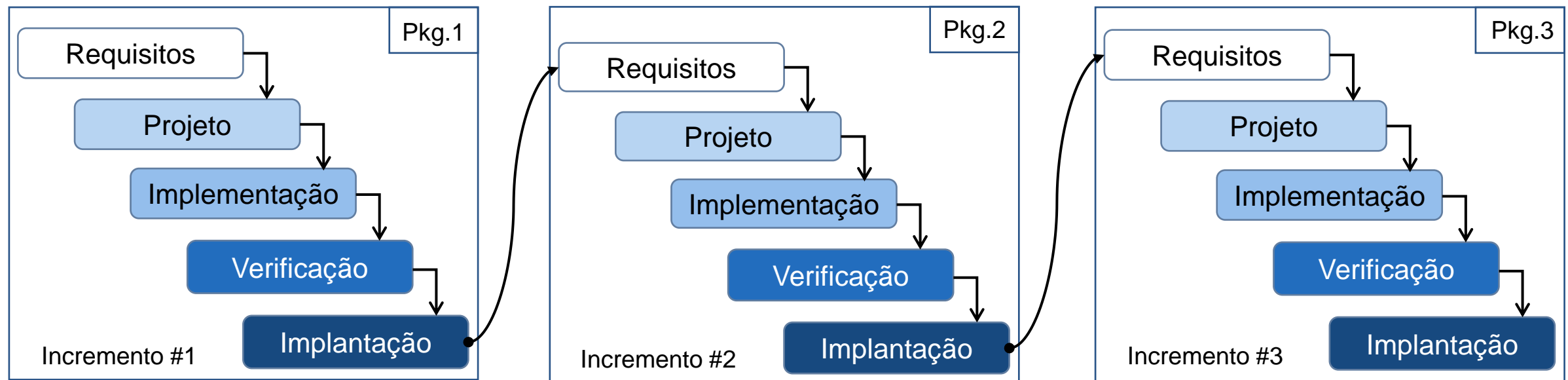


Figura 9. Modelo de desenvolvimento incremental. (CAIXETA, 2020).

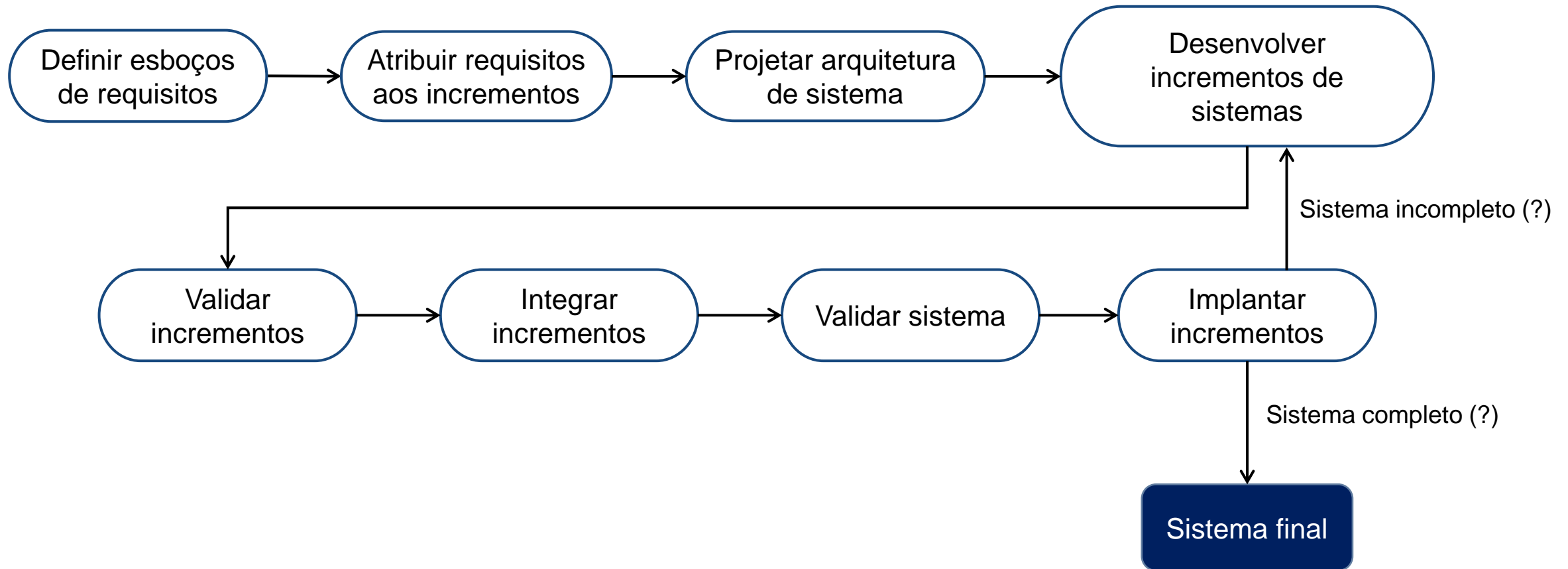


Figura 10. Estágios de testes no modelo incremental. (SOMMERVILLE, 2018).

- De acordo com Sommerville (2018), cada incremento ou versão do sistema incorpora alguma funcionalidade necessária para o cliente.
- Os incrementos iniciais são os que incluem a(s) funcionalidade(s) mais importante(s) ou mais urgente(s). Isso significa que o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado.
- Para Pressman (2011), o modelo incremental tem seu foco voltado para a entrega de um produto operacional com cada incremento.

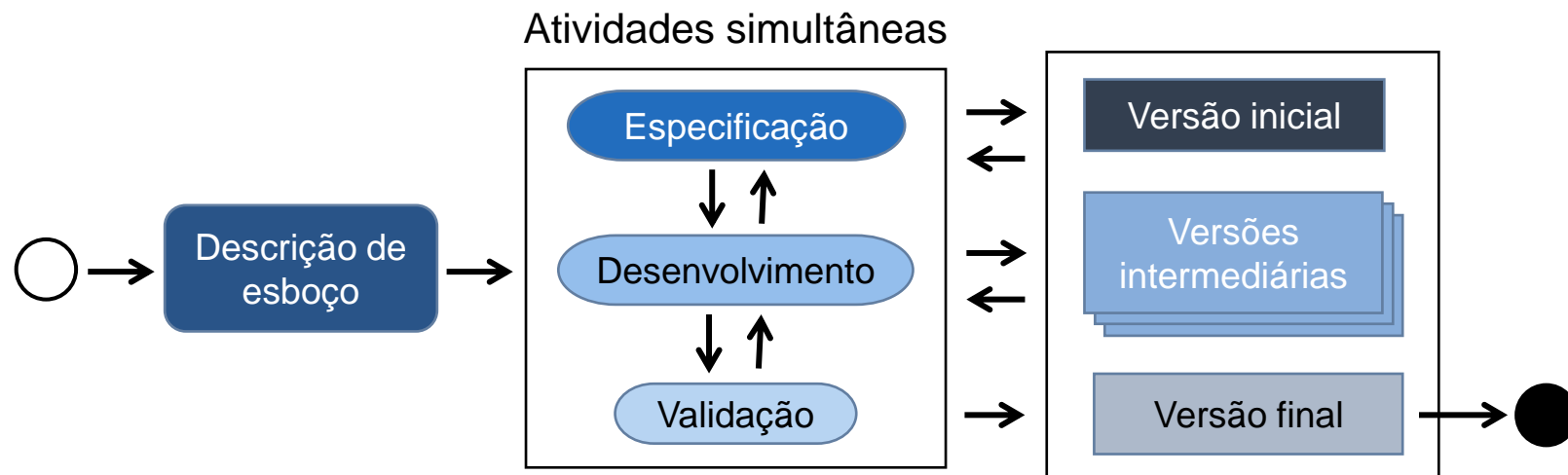


Figura 11. Dinâmica de desenvolvimento no modelo incremental. (CAIXETA, 2020).

- 03 vantagens e 02 desvantagens segundo Sommerville (2018).

As três vantagens:

1. O custo de acomodar as mudanças nos requisitos do cliente é reduzido. A quantidade de análise e documentação a ser refeita é muito menor do que o necessário no modelo em cascata.
2. É mais fácil obter *feedback* dos clientes. Eles podem fazer comentários sobre as demonstrações do *software* e ver o quanto foi implementado. Os clientes têm dificuldade em avaliar a evolução por meio de documentos de projeto.
3. É possível obter entrega e implementação rápida de um *software* útil ao cliente, mesmo se toda a funcionalidade não for incluída. Os clientes podem usar e obter ganhos a partir do *software* inicial.

As duas desvantagens:

1. O processo não é visível. Os gerentes precisam de entregas regulares para mensurar o progresso. Se os sistemas são desenvolvidos com rapidez, não é economicamente viável produzir documentos que reflitam cada uma das versões do sistema.
2. A estrutura do sistema tende a se degradar com a adição dos novos incrementos. A menos que tempo e dinheiro sejam dispendidos em refatoração (*refactoring technique*) para melhoria do *software*. As constantes mudanças tendem a corromper sua estrutura. Incorporar futuras mudanças do *software* torna-se cada vez mais difícil e oneroso.

2.2.3. Modelo evolucionário

- *Software*, assim como todos sistemas complexos, evolui ao longo do tempo. (PRESSMAN, 2011).
- Este modelo produz versões cada vez mais completas do *software* a cada iteração. E obviamente, priorizando a construção de projetos de sistemas que evolua ao longo do tempo.
- São modelos interativos.
- Entre eles, se destacam:

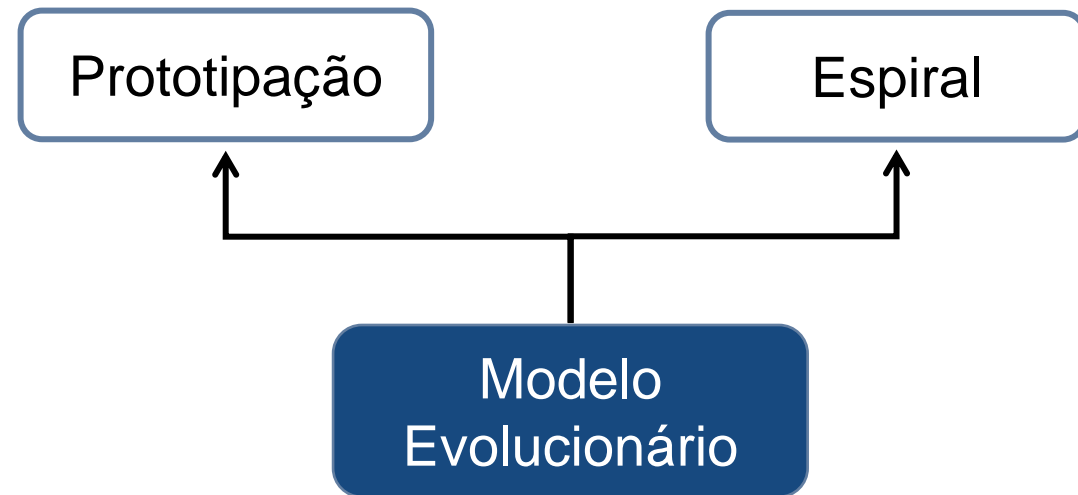


Figura 12. Principais modelos de desenvolvimento evolucionário. (CAIXETA, 2020).

Prototipação

- Um protótipo é uma versão inicial de um sistema de *software*, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções. (SOMMERVILLE, 2018).
- O desenvolvimento rápido e iterativo do protótipo é essencial para que os custos sejam controlados e os *stakeholders* do sistema possam experimentá-lo no início do processo de *software*. (*ibidem*).
- Contribui e antecipa com possíveis mudanças que poderão ser requisitadas. Por exemplo:
 1. Na engenharia de requisitos, um protótipo pode ajudar na elicitação e validação de requisitos de sistema.
 2. No processo de projeto de sistema, um protótipo pode ser usado para estudar soluções específicas do *software* e para apoiar o projeto de interface de usuário.

- Para Sommerville (2018), enquanto o sistema está em projeto, um protótipo pode ser usado para a realização de experimentos visando a verificação da viabilidade da proposta.
- Por exemplo, um projeto de banco de dados pode ser prototipado e testado para verificar se suporta de modo eficiente o acesso aos dados para as consultas mais comuns dos usuários.
- Prototipação também é uma parte essencial na construção de interfaces para os usuários. Devido à natureza dinâmica das interfaces, descrições textuais e diagramas não são bons o suficiente para expressar seus requisitos.



[Veja neste site os principais tipos de prototipação.](#)

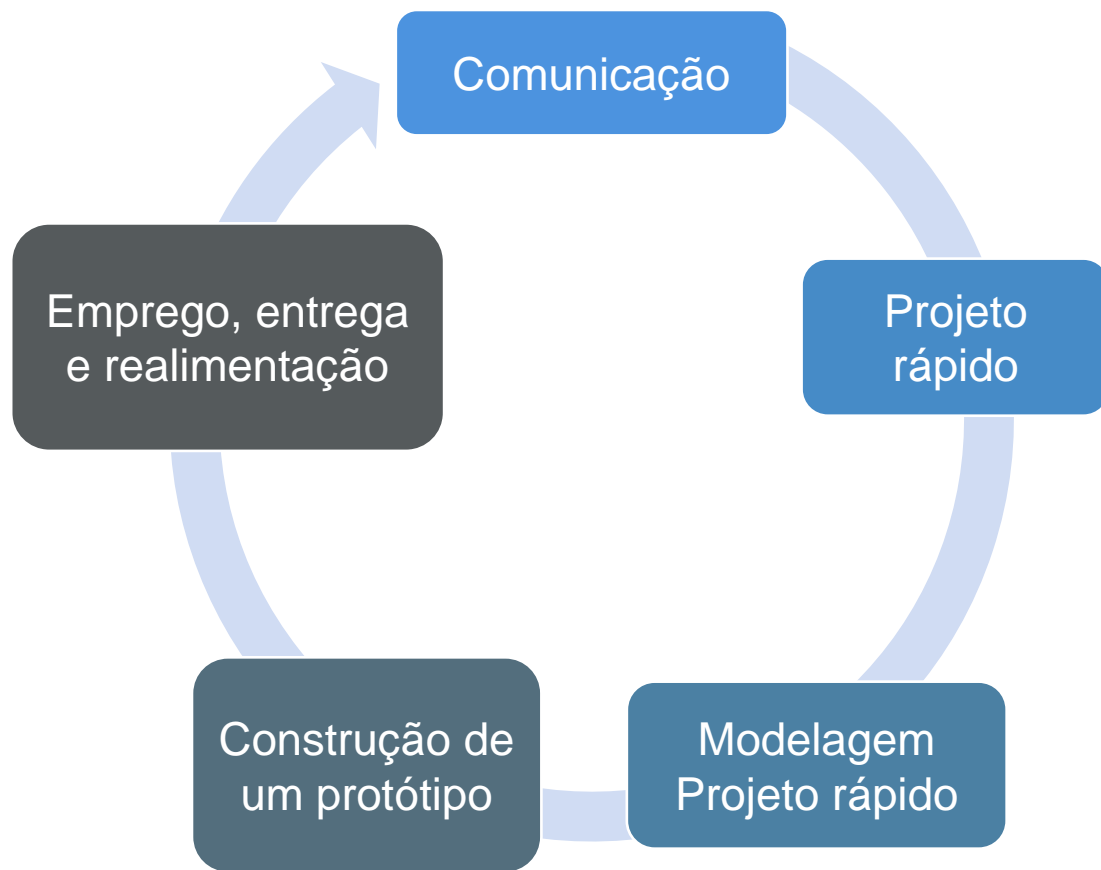


Figura 13. Modelo evolucionário da prototipagem. (CAIXETA, 2020).

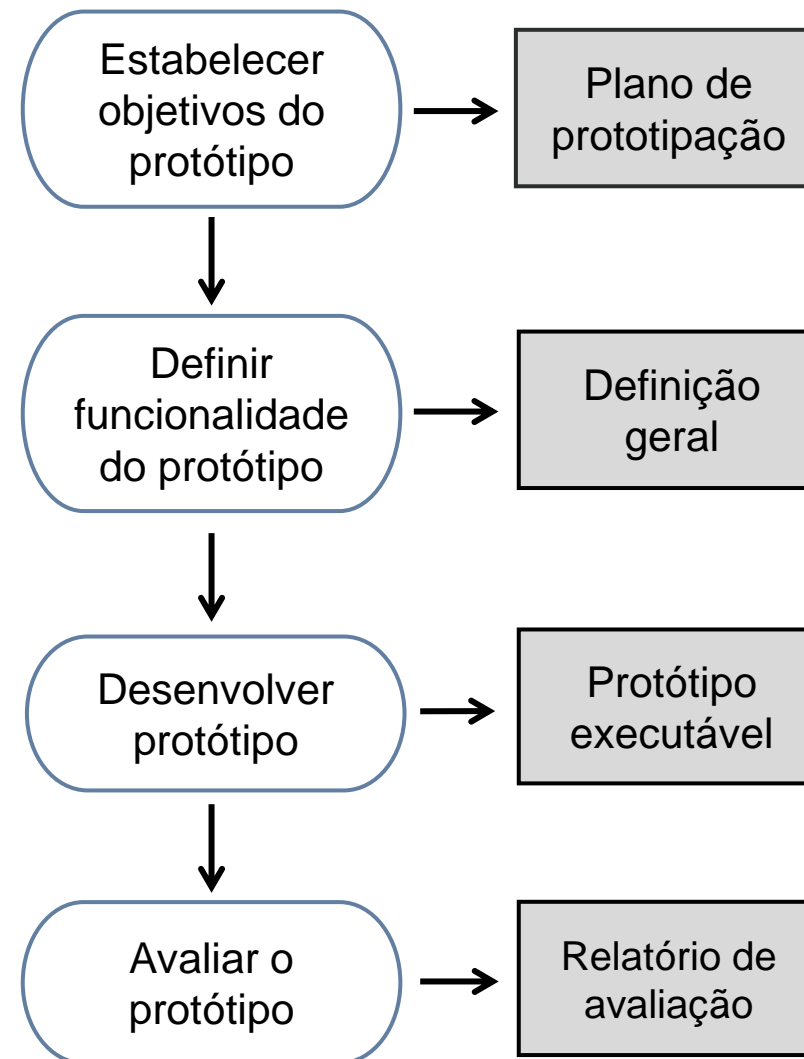


Figura 14. Dinâmica de desenvolvimento no modelo de Prototipação. (CAIXETA, 2020).

- Algumas desvantagens segundo Sommerville (2018).
 1. Mudanças rápidas durante o desenvolvimento inevitavelmente significam que o protótipo não está documentado. A única especificação de projeto é o código do protótipo. Para a manutenção a longo prazo, isso não é bom o suficiente.
 2. As mudanças durante o desenvolvimento do protótipo provavelmente terão degradado a estrutura do sistema. O sistema será difícil e custoso de ser mantido.
- Pressman (2011) apresenta algumas conclusões entre elas:
 - O segredo é definir as regras do jogo logo no início; isso significa que todos os envolvidos devem concordar que o protótipo é construído para servir como um mecanismo para definição de requisitos. Portanto, este poderá descartado, pelo menos em parte.
 - Visa-se a qualidade do *software* na versão final.

Espiral (Boehm, 1988)

- Segundo Sommerville (2018) este *framework* de processo de *software*, foi proposto por Barry Boehm (1988) e é dirigido a fatores de riscos.
- Boehm (1988 *apud* PRESSMAN, 2011) descreve o modelo da seguinte forma:

O modelo espiral de desenvolvimento é um gerador de *modelos de processos* dirigidos a riscos e é utilizado para guiar a engenharia de sistemas intensivos de *software*, que ocorre de forma concorrente e tem múltiplos envolvidos. Possui duas características principais que o distinguem. A primeira consiste em uma abordagem *cíclica* voltada para ampliar, incrementalmente, o grau de definição e a implementação de um sistema, enquanto diminui o grau de risco do mesmo. A segunda característica consiste em uma série de pontos âncora de controle para assegurar o comprometimento de interessados quanto à busca de soluções de sistema que sejam mutuamente satisfatórias e praticáveis.

- No modelo espiral, o *software* será desenvolvido em uma série de versões evolucionárias. (PRESSMAN, 2011).
- Nas primeiras iterações, a versão pode consistir em um modelo ou em um protótipo. Já nas iterações posteriores, são produzidas versões cada vez mais completas do sistema.
- Para Sommerville (2018), a principal diferença entre o modelo espiral e outros modelos é o reconhecimento explícito do risco.

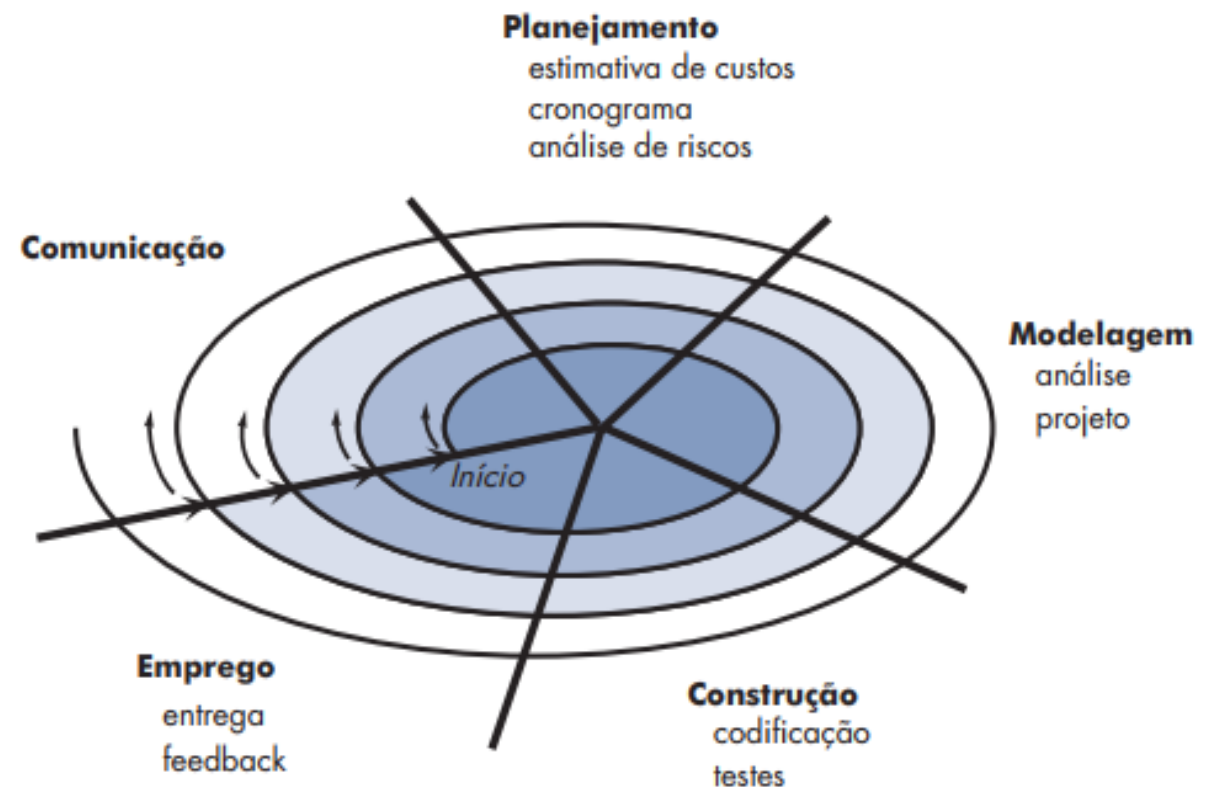
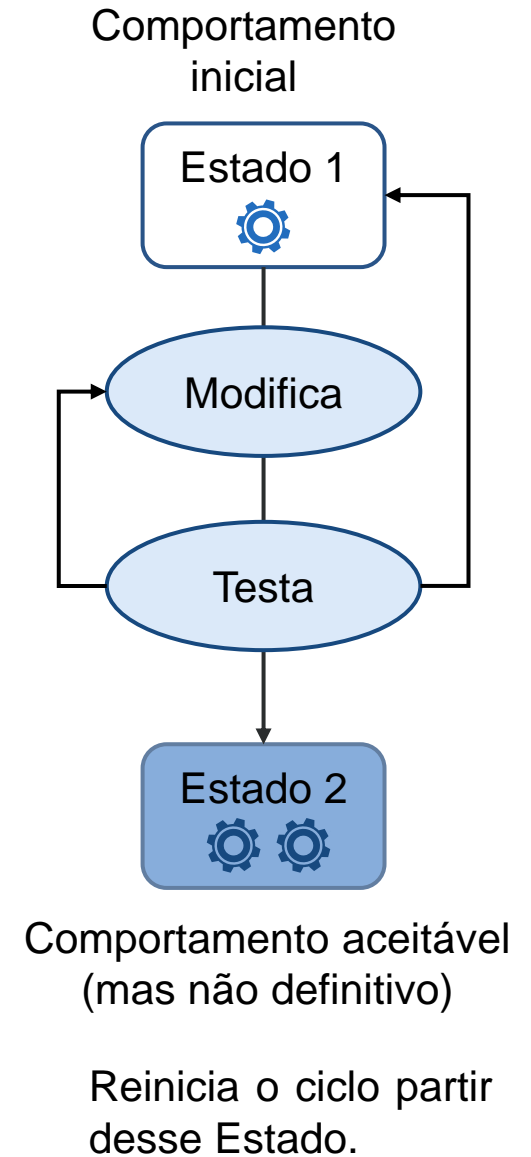


Figura 15. Modelo típico de Espiral baseado na proposta de Boehm (1988). (PRESSMAN, 2011).

2.2.4 Modelos concorrentes

- Também conhecido como engenharia concorrente, possibilita à equipe de *software* representar elementos concorrentes e iterativos de qualquer um dos modelos de processos descritos anteriormente. (PRESSMAN, 2011)¹.
- A modelagem concorrente define uma série de eventos que irão disparar transições de estado para estado² para cada uma das atividades, ações ou tarefas da engenharia de *software*.



1) Neste modelo a análise e o projeto são tarefas complexas que requerem discussão substancial.

2) Um estado é algum modo externamente observável do comportamento.

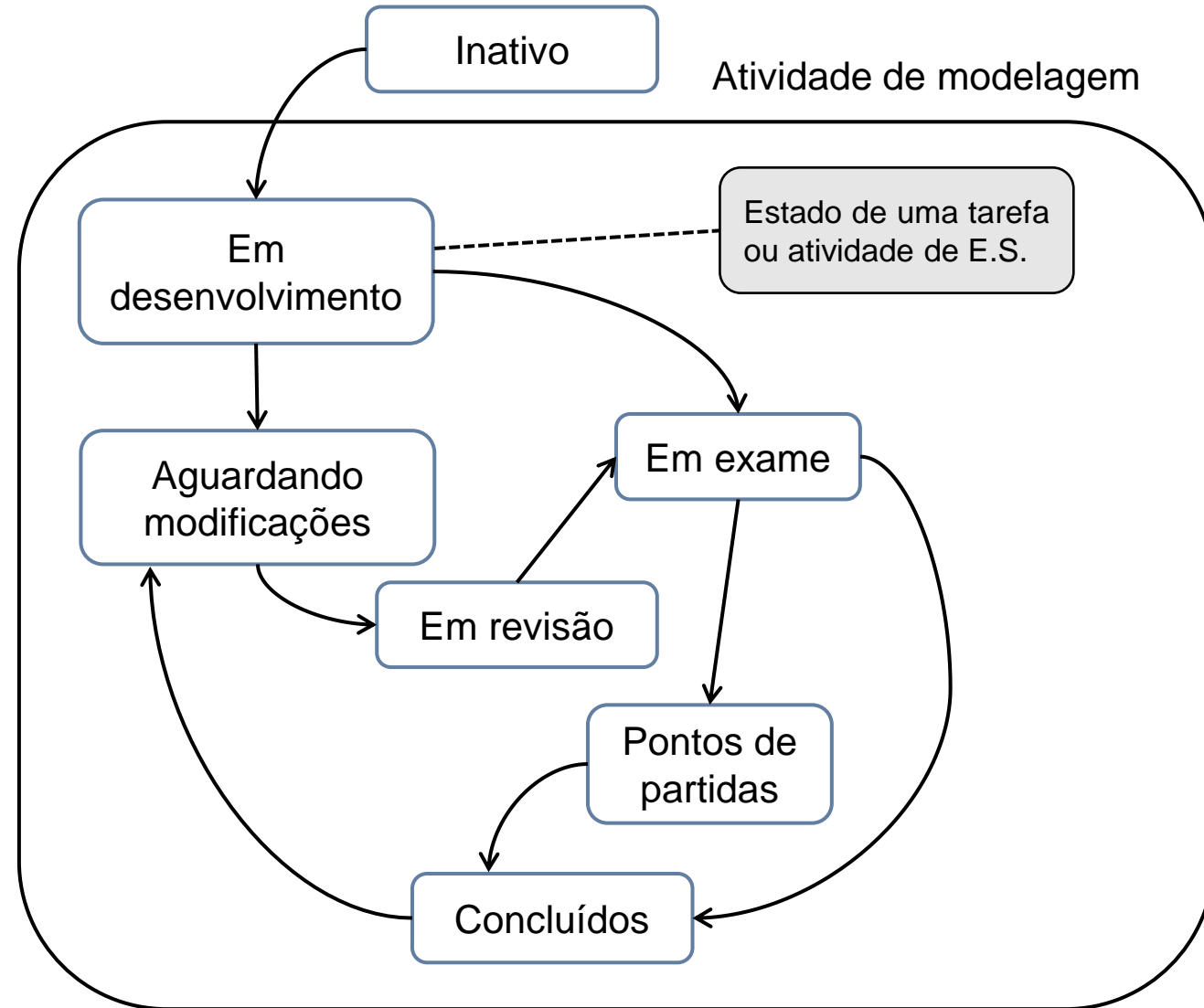


Figura 16. Modelo de processo concorrente.(PRESSMAN, 2011).

2.2.5 Baseado em componente ou orientado ao reuso

- Na maioria dos projetos de *software*, há algum reuso de componentes. Isso acontece muitas vezes informalmente, quando as pessoas envolvidas no projeto sabem de projetos ou códigos semelhantes ao que é exigido. [...]. (SOMMERVILLE, 2018).
- Abordagens orientadas a reuso dependem de uma ampla base de componentes reusáveis de *software* e de um *framework* de integração para a composição desses componentes. (*ibidem*).
- De acordo com Sommerville (*ibidem*), esses componentes são sistemas completos chamados de COTS – *Commercial Off-The-Shelf* (ou melhor dizendo, de prateleira), e são capazes de fornecer as funcionalidades específicas desejadas pelos clientes.

- Para Sommerville (2018), os estágios de processos orientado a reuso são:
 1. Análise de componentes: Dada a especificação de requisitos, é feita uma busca por componentes para implementar essa especificação. Em geral, não há correspondência exata. Torna-se necessária readaptação de componentes. (CAIXETA, 2020).
 2. Modificação de requisitos: Durante esse estágio, os requisitos são analisados usando-se informações sobre os componentes que foram descobertos. Em seguida, estes são modificados. No caso de modificações impossíveis, a atividade de Análise de componentes (item 1) pode ser reinserida na busca por soluções alternativas.
 3. Projeto do sistema com reuso: Durante esse estágio, o *framework* do sistema é projetado ou algo existente é reusado. [...]. Alguns *softwares* novos podem ser necessários, se os componentes reusáveis não estiverem disponíveis.
 4. Desenvolvimento e integração: *Softwares* que não podem ser adquiridos externamente são desenvolvidos, e os componentes e sistemas COTS são integrados para criar o novo sistema.

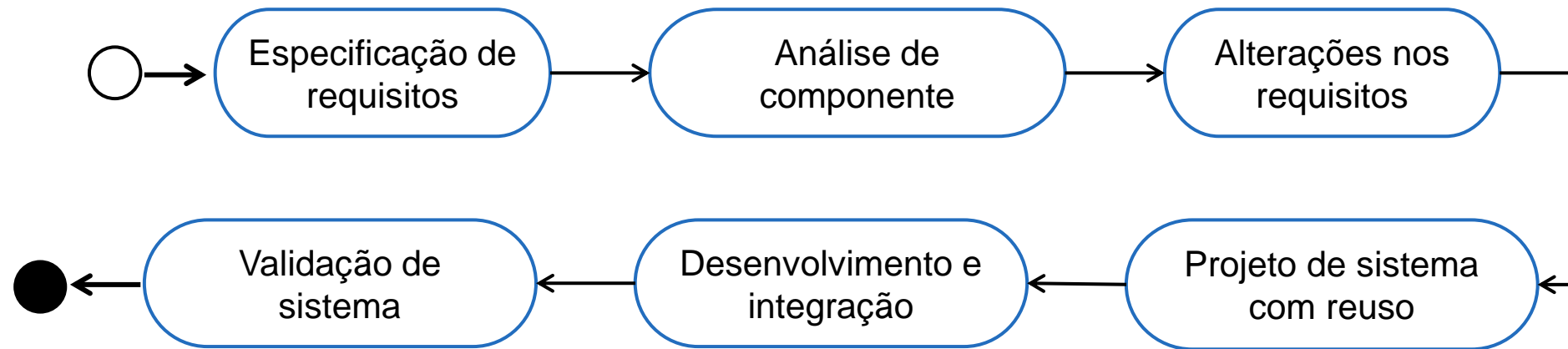


Figura 17. Fluxo de engenharia de *software* orientada a reuso. (SOMMERVILLE, 2018).

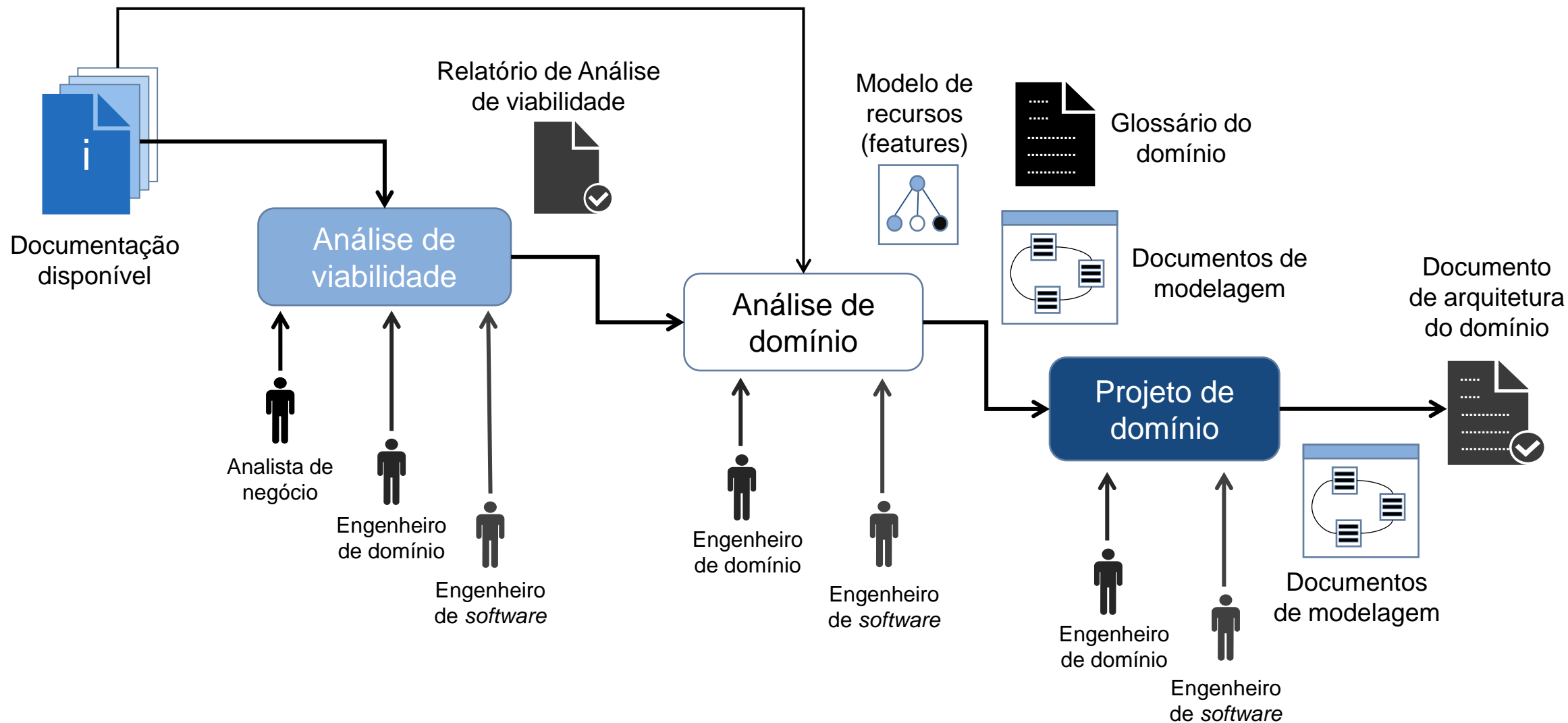


Figura 18. Engenharia de domínio orientada a reuso de componentes.
(Disponível em: <https://www.cin.ufpe.br/~aa2/ABC/engdominio.html>. Acessado em: 17.out.2020)

- Para Sommerville (2018), existem três tipos de componentes de *software* que podem ser usados em um processo orientado a reuso:
 1. Web services desenvolvidos de acordo com os padrões de serviço e que estão disponíveis para invocação remota.
 2. Coleções de objetos que são desenvolvidas como um pacote a ser integrado com um *framework* de componentes, como .NET ou J2EE.
 3. Sistemas de *software* stand-alone configurados para uso em um ambiente particular.

- As vantagens e desvantagens segundo Sommerville (2018).

Vantagens:

1. Redução da quantidade de *software* a ser desenvolvido.
2. Redução de custos e riscos.
3. Geralmente, também proporciona entregas mais rápidas.

Desvantagens:

1. Os compromissos com os requisitos são inevitáveis, e isso pode levar a um sistema que não atende às reais necessidades dos usuários.
2. Além disso, algum controle sobre a evolução do sistema é perdido, pois as novas versões dos componentes reusáveis não estão sob o controle da organização que os está utilizando.

2.2.6 Modelos de métodos formais

- Engloba um conjunto de atividades que conduzem à especificação matemática formal do *software*. (PRESSMAN, 2011).
- Possibilitam especificar, desenvolver e verificar um sistema baseado em computador através da aplicação de uma notação matemática rigorosa. Uma variação dessa abordagem, chamada engenharia de *software* “*cleanroom*” (sala limpa/leve/nítida)¹, é aplicada atualmente por algumas organizações de desenvolvimento de *software*.
- Durante o desenvolvimento, este método oferece mecanismos que eliminam muitos problemas dificilmente resolvidos por outros modelos, e.g., ambiguidade, incompletude e inconsistência, podem ser eliminados através da aplicação de análise matemática.

1) Ref. PRESSMAN, 2011 *apud* MILLS, DYER, LINGER, 1987 & DYER, 1992.

- A utilização deste método serve como base para a verificação da codificação, possibilitando a descoberta e a correção de erros, que de outra forma poderiam passar despercebidos.
- Mas [...], se métodos formais são capazes de demonstrar correção de *software*, por que não são amplamente utilizados?
 - ✓ Consumem tempo e dinheiro.
 - ✓ Poucos desenvolvedores possuem formação e experiência necessárias para aplicação destes métodos, sendo necessário treinamento extensivo.
 - ✓ De difícil uso na comunicação com os clientes, que tecnicamente são despreparados para o entendimento.

- Apesar de tais preocupações, esta abordagem tem conquistado adeptos entre os programadores que precisam desenvolver *software* com fator crítico de segurança, e.g., os desenvolvedores de sistemas de aviação para aeronaves e equipamentos médicos.



Figura 19. Problemas no sistema de *software* do Boeing 737 MAX ocasionou na parada temporária de produção deste modelo de aeronave.



Figura 20. Sistemas de *softwares* embarcados em equipamentos médicos.

BIBLIOGRAFIA

PRESSMAN, R. S. Engenharia de Software: Uma abordagem profissional. 7ª edição. Dados eletrônicos. Porto Alegre: AMGH-McGrawHill, 2011.

SOMMERVILLE, I. Engenharia de Software. 10ª edição. São Paulo: Pearson Prentice Hall, 2018.

WAZLAWICK, R. S. Engenharia de software: conceitos e práticas. Rio de Janeiro: Elsevier, 2013.



Obrigado!

Engenharia de Software