ENGENHARIA DE SOFTWARE

AULA 1

Prof. Alex Mateus Porn

CONVERSA INICIAL

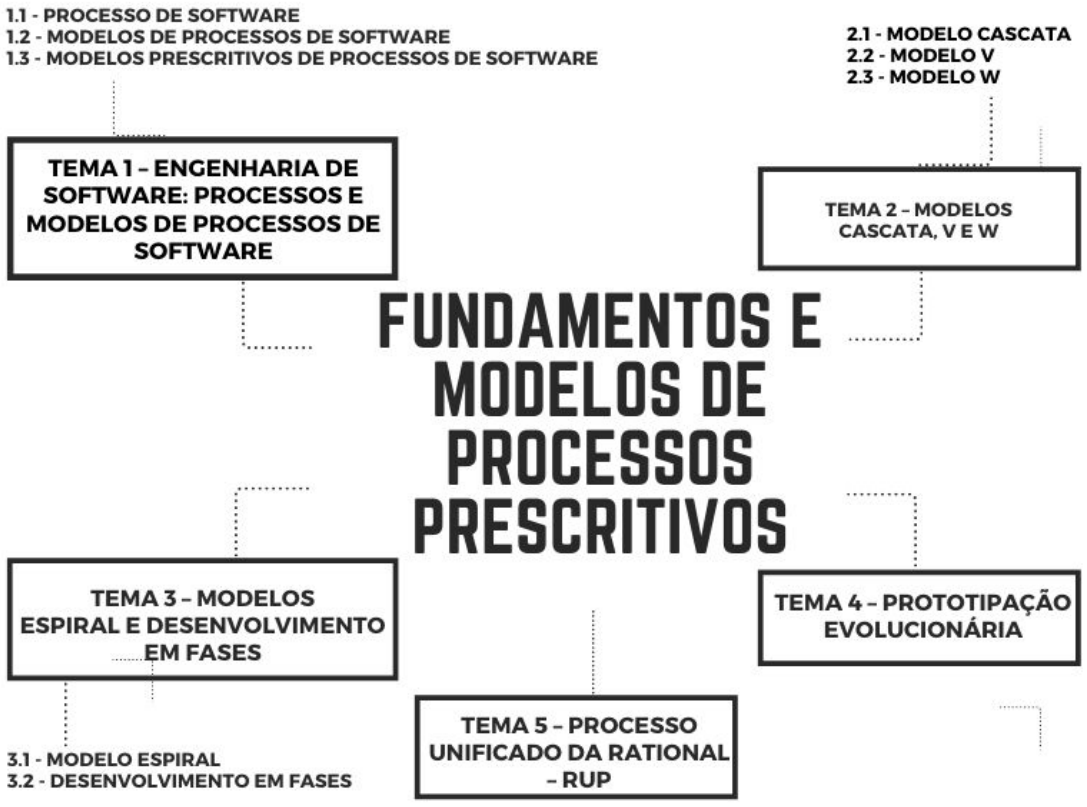
Fundamentos e modelos de processos prescritivos

Nesta aula, abordaremos inicialmente a necessidade da Engenharia de *Software*, em razão da complexidade e da grande diversidade de padrões para o desenvolvimento de *softwares*. Na sequência, serão apresentados conceitos sobre processos de *software* e os principais modelos de processos prescritivos aplicados no desenvolvimento e gerenciamento de projetos de *software*.

Estudaremos em detalhes a estrutura, formas de aplicação e algumas comparações entre os modelos Cascata, V, W, Espiral, Desenvolvimento em Fases e o modelo RUP. Para cada um desses modelos, analisaremos alguns exemplos para nos auxiliar na compreensão de sua aplicação em nossos futuros projetos de *software*.

Ao longo desta aula, serão trabalhados o seguinte conteúdo.

Figura 1 – Conteúdos da aula



Fonte: Porn, 2020.

TEMA 1 – ENGENHARIA DE *SOFTWARE*: PROCESSOS E MODELOS DE PROCESSOS DE *SOFTWARE*

Atualmente, conforme destaca Sommerville (2018, p. 3), existem diferentes tipos de sistemas de *software*, variando de sistemas embarcados simples até sistemas de informação complexos. Podemos, nesse contexto, rapidamente citar diversas áreas que fazem uso constante de sistemas de *software*, como os serviços públicos nacionais, indústrias completamente informatizadas, setores de entretenimento, telefonia, entre outros. Portanto, os sistemas de *software* podem rapidamente se tornar bastante complexos, difíceis de entender e caros de modificar. Diante dessa abordagem, não há notações, métodos ou técnicas universais para o desenvolvimento de uma variedade completamente heterogênea de sistemas de *software*.

Nesse contexto, todas essas aplicações precisam de engenharia de *software*. Conforme destaca Sommerville (2018, p. 5): “A engenharia de *software* se destina a apoiar o desenvolvimento de *software* profissional em vez da programação individual. Ela inclui técnicas que apoiam a especificação, o projeto e a evolução do *software*.”

Ainda nessa abordagem, Sommerville (2018, p. 7) também destaca que a engenharia de *software* é uma disciplina relacionada a todos os aspectos da produção de *software*, desde os estágios iniciais da especificação até a manutenção, depois que o sistema passa a ser usado. Já Wazlawick (2013, p. 4) aborda a engenharia de *software* como o processo de estudar, criar e otimizar os processos de trabalho para os desenvolvedores de *software*. Portanto, assim como não existem métodos e técnicas padronizados para o desenvolvimento de sistemas de *software*, também não existe uma padronização para a engenharia de *software*.

1.1 Processos de *software*

Conforme abordado por Sommerville (2018, p. 29), um processo de *software* refere-se a um conjunto de atividades relacionadas que levam à produção de um sistema de *software*. Nesse contexto, o autor (2018, p. 29) reforça que: “[...] não há um método universal de engenharia de *software* que seja aplicável a todos os tipos diferentes de sistemas de *software* existentes. Desse modo, não existem processos de *software* universalmente aceitos.”

De um modo um pouco mais aprofundado, Wazlawick (2013, p. 11) pontua que há mais coisas envolvidas em um processo de *software*, além do fato de ser um conjunto de atividades, por exemplo, pessoas, recursos, restrições e padrões a serem seguidos. Diante dessa abordagem, Wazlawick (2013, p. 11) compreende um processo de *software* como um conjunto de atividades interdependentes, com responsáveis e com entradas e saídas definidas.

Portanto, como pudemos observar na proposição de Sommerville, não existem processos de *software* universalmente aplicáveis para uma vasta variedade de sistemas de *software*, de modo que o processo utilizado em cada situação depende do tipo de *software* que está sendo desenvolvido. Porém, mesmo existindo muitos processos de *software* diferentes, Sommerville (2018, p. 30) destaca que todos eles devem incluir as quatro atividades fundamentais da engenharia de *software*.

1. **Especificação**: a funcionalidade do *software* e as restrições sobre sua operação devem ser definidas.
2. **Desenvolvimento**: o *software* deve ser produzido para atender à especificação.
3. **Validação**: o *software* deve ser validado para garantir que atenda ao que o cliente deseja.
4. **Evolução**: o *software* deve evoluir para atender às mudanças nas necessidades dos clientes.

1.2 Modelos de processos de *software*

Conforme vimos no tópico anterior, a engenharia de *software* não define métodos ou técnicas que padronizem o desenvolvimento de *software*, porém, de acordo com Wazlawick (2013, p. 21), para que o desenvolvimento de sistemas deixe de ser artesanal e aconteça de forma mais previsível e com maior qualidade, é necessária a compreensão e a aplicação de um processo de desenvolvimento de *software*.

A implementação de um processo de *software* normalmente é estabelecida de acordo com modelos de processos. Para Sommerville (2018, p. 31), um modelo de processo de *software* é uma representação simplificada de um processo de *software* e, muitas vezes, é denominado *ciclo de vida do*

desenvolvimento de software. Os modelos de processos são divididos em duas categorias:

- **modelos de processos prescritivos**: modelos de processos tradicionais, que surgiram com o propósito de organizar e direcionar as atividades inerentes ao desenvolvimento de *software*. Recebem essa denominação, pois, de acordo com Pressman (2011), têm a capacidade de prescrever um conjunto de elementos e atividades próprias de um processo de desenvolvimento de *software*, como atividades metodológicas, ações de engenharia de *software*, tarefas, produtos de trabalho, garantia de qualidade e mecanismos de controle de mudanças para cada projeto;
- **modelos de processos ágeis**: são modelos que se referem a uma abordagem contrária aos modelos de processos prescritivos. Conforme Pressman (2011), os modelos ágeis possuem iterações curtas, em que o resultado é medido por meio do produto pronto, ao contrário dos modelos prescritivos, em que o desenvolvimento é divido em etapas bem definidas.

Nesta aula, conheceremos quais são os modelos de processos prescritivos, detalhando o modelo cascata, modelos V e W, modelos espiral e desenvolvimento em fases, modelo prototipação evolucionária e RUP. Quanto aos modelos de processos ágeis, estudaremos em detalhes futuramente.

1.3 Modelos prescritivos de processos de *software*

Como estudamos no tópico anterior, os modelos prescritivos de processos de desenvolvimento de *software*, são os tradicionais modelos criados com objetivos específicos para presumir o desenvolvimento de *software*. Sommerville (2018, p. 31) destaca os seguintes principais modelos prescritivos.

- **Modelo em cascata**: representa as atividades fundamentais do processo, como especificação, desenvolvimento, validação e evolução.
- **Desenvolvimento incremental**: intercala as atividades de especificação, desenvolvimento e validação. O modelo recebe esse nome, de modo que o sistema é desenvolvido como uma série de versões (incrementos) com cada uma delas acrescentando funcionalidades à versão anterior.
- **Integração e configuração**: baseia-se na disponibilidade de componentes ou sistemas reutilizáveis. O processo de desenvolvimento

se concentra na configuração desses componentes para que sejam utilizados em um novo contexto e na integração deles em um sistema.

Além dos principais modelos prescritivos de processos de desenvolvimento de *software* citados por Sommerville, Wazlawick (2013, p. 22) apresenta uma lista de vários outros modelos, conforme podemos ver na sequência.

- **Espiral**: tem como característica principal a realização de ciclos de prototipação para a redução de riscos de projeto.
- ***Sashimi***: introduz a noção de que as fases do modelo cascata não são estanques, mas que em determinado momento ele pode estar simultaneamente em mais de uma dessas fases.
- **Modelos V e W**: Focam no teste durante o desenvolvimento de *software* e indicam que este deve ser uma preocupação constante, e não apenas uma etapa colocada ao final do processo.
- **Modelo orientado a cronograma**: tem foco prioritariamente no desenvolvimento das funcionalidades mais importantes, deixando as demais para o final. Caso ocorram atrasos e o prazo for rígido, ao menos as funcionalidades mais críticas serão entregues no prazo.
- **Entrega em estágios**: tem como prioridade planejar e entregar partes prontas do sistema antes do final do projeto.
- **Prototipação evolucionária**: propõe o uso de protótipos para ajudar na compreensão da arquitetura, da interface e dos requisitos do sistema, quando não é possível conhecer bem esses aspectos *a priori*.
- **Entrega evolucionária**: combina a prototipação evolucionária com a entrega em estágios, mostrando que é possível fazer um planejamento adaptativo em que, a cada nova iteração, o gerente de projeto decide se vai acomodar as requisições de mudança que surgiram ao longo do projeto ou manter-se fiel ao planejamento inicial.
- **Modelos orientados a ferramentas**: família de modelos cuja única semelhança costuma consistir no fato de que eles são indicados para uso com ferramentas específicas.
- **Linhas de produto de *software***: tipo de processo que se aplica somente no desenvolvimento de uma família de produtos semelhantes. Desse

modo, é possível que a reusabilidade de componentes seja efetivamente planejada.

Dada a grande variedade existente de modelos de processos de desenvolvimento de *software*, ou ciclos de vida, Wazlawick (2013, p. 22) afirma que “alguns modelos de processos vêm caindo em desuso, enquanto outros evoluem a partir desses. O engenheiro de *software* deve escolher o modelo que for mais adequado para sua equipe e ao projeto que ele vai desenvolver.”

Portanto, podemos observar que a escolha de um modelo de processo de desenvolvimento de *software* pelo engenheiro de *software* depende da sua experiência no uso do modelo escolhido, de acordo com o tipo da aplicação em desenvolvimento.

TEMA 2 – MODELOS CASCATA, V E W

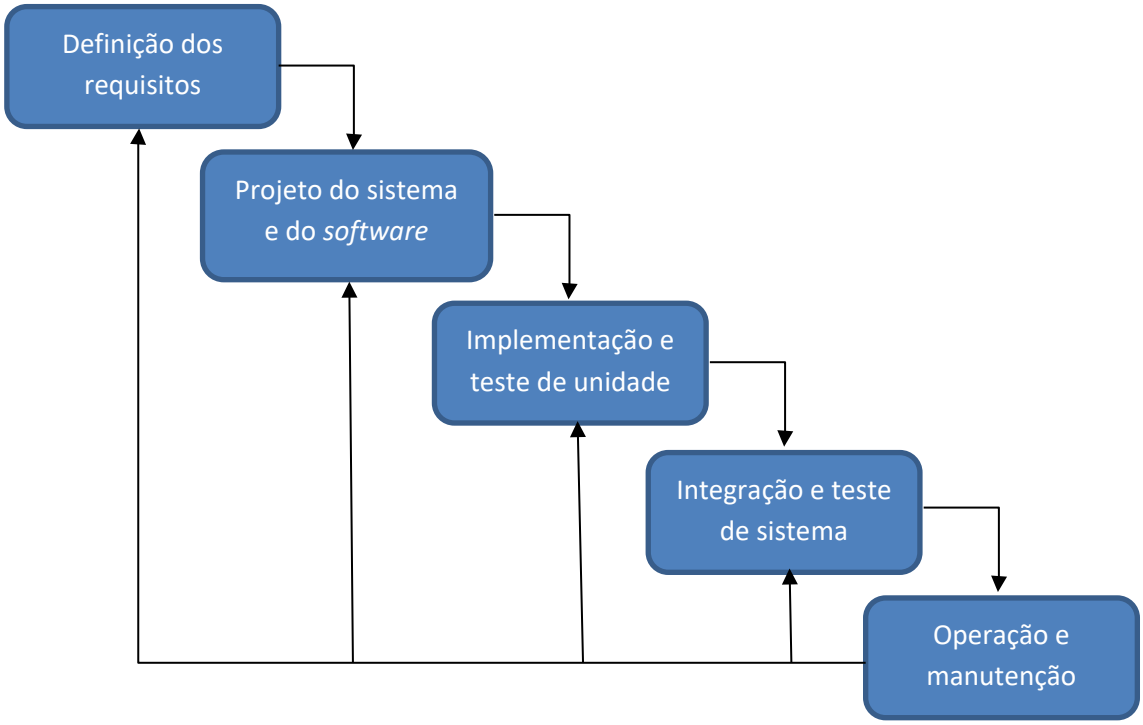
Conforme veremos neste tópico, o modelo cascata é um dos primeiros modelos de processos criado para o desenvolvimento de *software*, sendo aprimorado pelos modelos V e W em razão de alguns problemas identificados durante sua aplicação, conforme veremos na sequência.

2.1 Modelo cascata

O Modelo cascata foi originalmente desenvolvido na década de 1970 por Winston W. Royce (1970), sendo um dos primeiros modelos de processos criado, derivado dos modelos utilizados na engenharia de grandes sistemas militares.

Conforme exposto em Sommerville (2018) e Wazlawick (2013), o modelo cascata apresenta o processo de desenvolvimento de *software* como uma série de fases. A Figura 2 apresenta a especificação do modelo cascata.

Figura 2 – Modelo cascata



Fonte: Com base em Sommerville, 2018, p. 33.

Cada uma das fases do modelo cascata, conforme visto na Figura 2 anterior, refere-se às atividades fundamentais do desenvolvimento de *software*. De acordo com Wazlawick (2013, p. 25), o modelo prevê uma atividade de revisão ao final de cada fase para que se avalie se o projeto pode passar à fase seguinte. Caso na revisão perceba-se que o projeto não está pronto para passar para a próxima fase, ele permanece na fase em questão.

Na sequência, veremos a especificação de cada fase do modelo cascata, conforme apresentado por Sommerville (2018, p. 33).

- **Definição dos requisitos**: os serviços, as restrições e as metas do sistema são estabelecidos por meio de consultas aos usuários. Depois, eles são definidos em detalhes e servem como uma especificação do sistema.
- **Projeto do sistema e do *software***: o processo de projeto do sistema reparte os requisitos entre requisitos de sistema de *hardware* e de *software*, e estabelece uma arquitetura global do sistema. O projeto de *software* envolve a identificação e a descrição das abstrações fundamentais do sistema de *software* e seus relacionamentos.
- **Implementação e teste de unidade**: o projeto do *software* é realizado como um conjunto de programas ou unidades de programa. O teste de

unidade envolve a verificação de cada unidade, conferindo se satisfazem a sua especificação.

- **Integração e teste de sistema**: as unidades do programa ou os programas são integrados e testados como um sistema completo a fim de garantir que os requisitos de *software* tenham sido cumpridos. Após os testes, o sistema de *software* é entregue ao cliente.
- **Operação e manutenção**: normalmente, esta é a fase mais longa do processo. O sistema é instalado e colocado em uso. A manutenção envolve corrigir os erros que não foram descobertos nas primeiras fases do processo, melhorar a implementação das unidades do sistema e aperfeiçoar os serviços do sistema à medida que novos requisitos são descobertos.

O Quadro 1 apresenta as vantagens e desvantagens do modelo cascata segundo Wazlawick (2013, p. 26-28).

Quadro 1 – Vantagens e desvantagens do modelo cascata

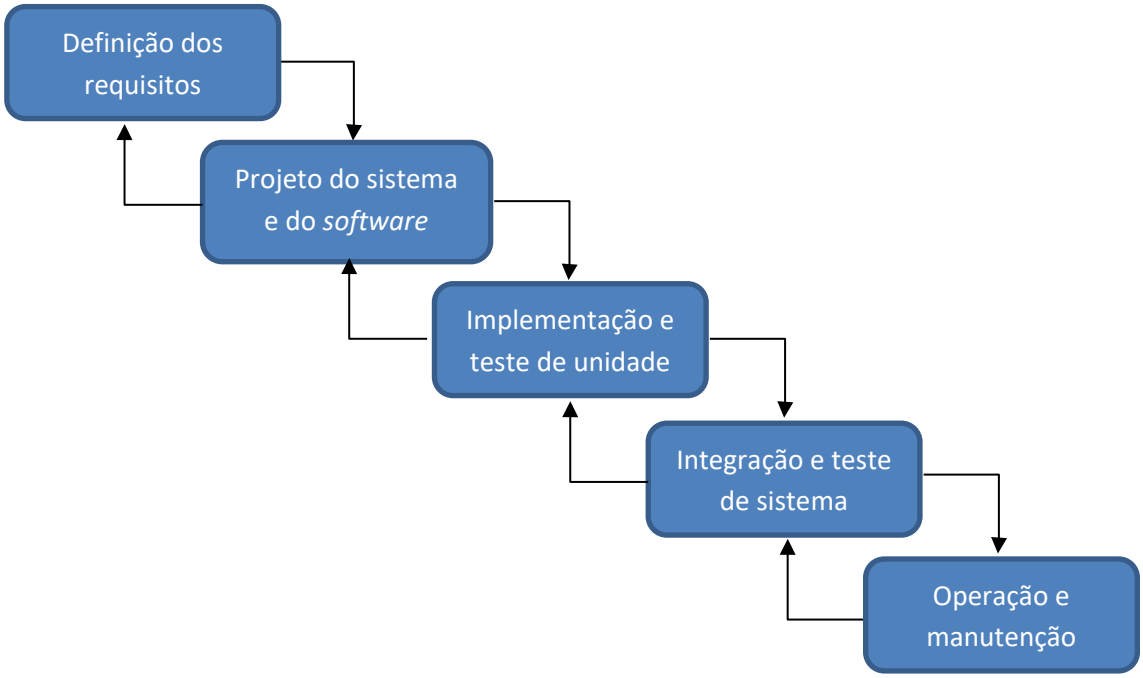
Vantagens	Desvantagens
Fases bem definidas ajudam a detectar erros mais cedo.	Não produz resultados tangíveis até a fase de codificação.
Procura promover a estabilidade dos requisitos.	É difícil estabelecer requisitos completos antes de começar a codificar.
Funciona bem quando os requisitos são conhecidos e estáveis.	Desenvolvedores sempre reclamam que os usuários não sabem expressar aquilo de que precisam.
É adequado para equipes tecnicamente fracas ou inexperientes.	Não há flexibilidade com requisitos.

Fonte: Wazlawick, 2013, p. 26-28.

É possível observar que o modelo cascata é sequencial, possibilitando retornar às fases anteriores somente após a fase de operação e manutenção. Isso o torna um modelo impraticável, visto que muitos problemas descobertos em fases mais avançadas não necessariamente foram originados na fase imediatamente anterior.

Tentando solucionar esse problema, Royce (1970) propôs o modelo de cascata dupla, de modo que problemas encontrados em uma fase podem ser resolvidos retornando-se à fase anterior para efetuar as correções, seguindo um modelo conforme apresentado na Figura 3.

Figura 3 – Modelo cascata dupla



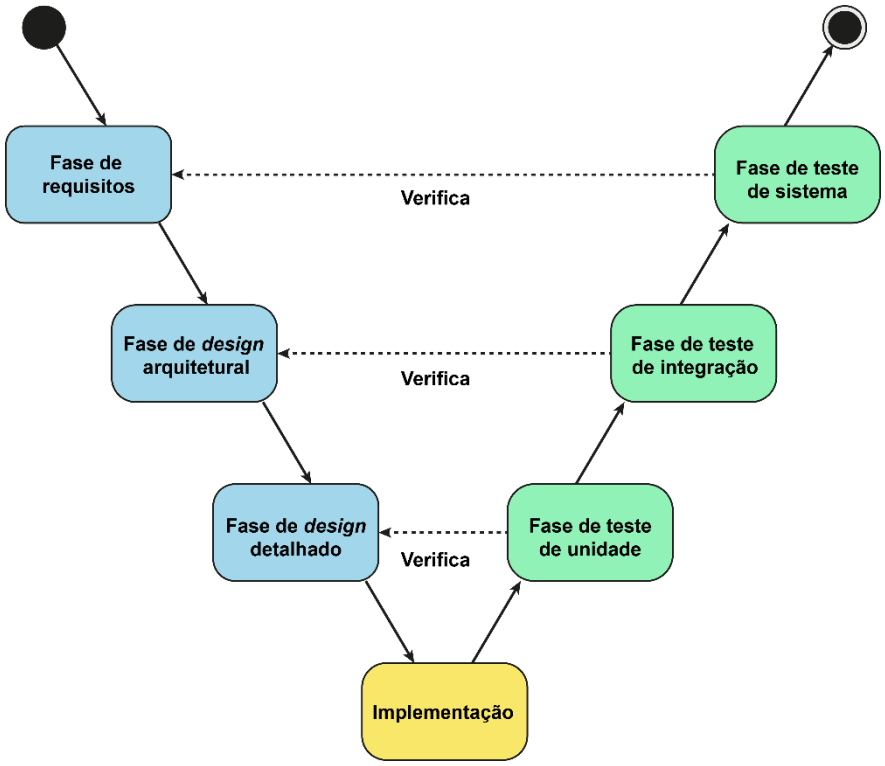
Fonte: Com base em Royce, 1970).

Conforme observado na Figura 3, no modelo cascata dupla é possível retornar às fases anteriores. Porém, caso o problema não tenha sido originado na fase imediatamente anterior, pode gerar uma dificuldade maior na resolução do problema, acarretando possivelmente atrasos no *software*.

2.2 Modelo V

Com o intuito de amenizar as dificuldades impostas pelo modelo cascata, o modelo V originou-se com base naquele, prevendo uma fase de validação e uma de verificação para cada fase de construção do sistema. A Figura 4 apresenta a especificação do modelo V.

Figura 4 – Modelo V



Fonte: Porn, 2020.

De acordo com o Modelo V apresentado na Figura 4 anterior e, conforme exposto por Wazlawick (2013, p. 31-32), cada fase do modelo representa:

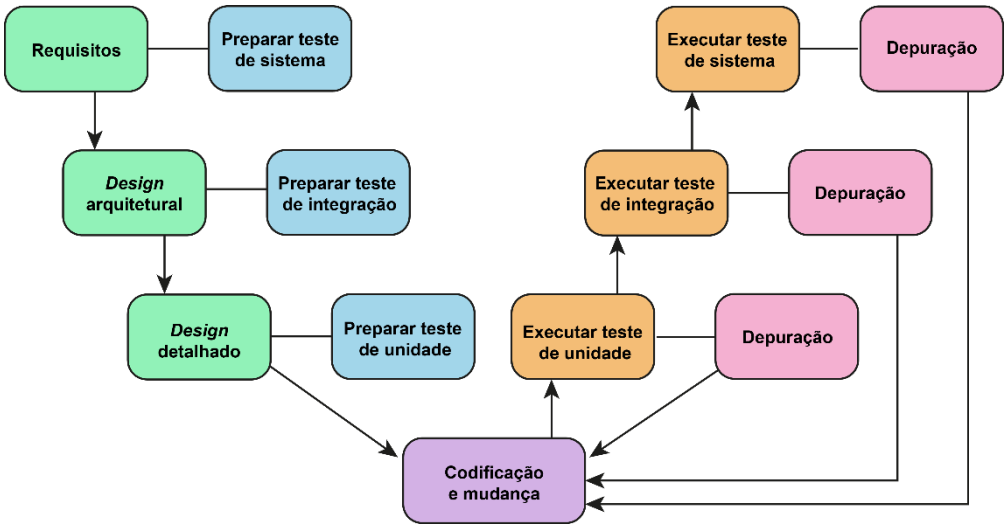
- **fase de requisitos** – a equipe e o cliente eliciam e elaboram o documento de requisitos, que é aprovado conjuntamente, na forma de um contrato de desenvolvimento;
- **fase de *design* arquitetural** – a equipe organiza os requisitos em unidades funcionais coesas, definindo como as diferentes partes arquiteturais do sistema vão se interconectar e colaborar;
- **fase de *design* detalhado** – a equipe vai aprofundar a descrição das partes do sistema e tomar decisões sobre como elas serão implementadas;
- **fase de implementação** – o *software* é implementado de acordo com a especificação detalhada;
- **Fase de teste de unidade** – tem como objetivo verificar se todas as unidades se comportam como especificado na fase de *design* detalhado. O projeto só segue em frente se todas as unidades passam em todos os testes;

- **fase de teste de integração** – tem como objetivo verificar se o sistema se comporta conforme a especificação do *design* arquitetural;
- **fase de teste de sistema** – verifica se o sistema satisfaz os requisitos especificados.

2.3 Modelo W

Com o objetivo de focar ainda mais na fase de testes do sistema, o Modelo W foi proposto como uma variação ao modelo V. Conforme Wazlawick (2013, p. 32), a principal variação para o modelo V está na observação de que há uma divisão muito estrita entre as atividades construtivas do lado esquerdo do V e as atividades de teste no lado direito. A Figura 5 apresenta o modelo de processos de desenvolvimento de *software* W.

Figura 5 – Modelo W



Fonte: Wazlawick, 2013, p. 33.

Com relação ao modelo W, Wazlawick (2013, p. 32-33), faz-se três importantes apontamentos:

- a fase de requisitos diz respeito ao fato de eles serem ou não testáveis. Apenas requisitos que possam ser testados são aceitáveis ao final dessa fase;
- na fase de *design* arquitetural, arquiteturas simples devem ser fáceis de testar, caso contrário, talvez a arquitetura seja demasiadamente complexa e necessite ser refatorada;

- na fase de projeto detalhado, a mesma questão se coloca em relação às unidades. Unidades coesas são mais fáceis de testar.

Com base na análise do Modelo W, podemos concordar com a afirmação de Wazlawick (2013, p. 33) de que esse modelo incorpora o teste nas atividades de desenvolvimento desde seu início, e não apenas nas fases finais. Essa característica também é fortemente utilizada pelos métodos ágeis, conforme veremos futuramente, os quais preconizam que o caso de teste deve ser produzido antes do código que será testado.

TEMA 3 – MODELOS ESPIRAL E DESENVOLVIMENTO EM FASES

Como vimos no tópico anterior, ao longo do tempo o modelo cascata sofreu variações com a intenção de minimizar os problemas identificados durante o desenvolvimento de projetos de *software*. Uma dessas variações foi a proposta de Royce (1970), para desenvolver o projeto pelo menos duas vezes, para que tudo o que foi aprendido na primeira vez pudesse ser utilizado na segunda. Nesse contexto, conforme Wazlawick (2013, p. 36), o modelo espiral é uma forma de realizar essas iterações de forma mais organizada, iniciando com pequenos protótipos e avançando para projetos cada vez maiores. Desse modo, o modelo espiral atende de certa forma às recomendações de Royce (1970), conforme veremos em mais detalhes no tópico 3.1 adiante.

Já em relação ao modelo de desenvolvimento em fases, ele surgiu com o propósito principal de resolver problemas de prazos de entrega, com a proposta de realizar as entregas com base em funcionalidades que fossem sendo finalizadas, denominadas *versões do sistema*. Veremos esse modelo em mais detalhes no tópico 3.2 adiante.

3.1 Modelo espiral

O modelo espiral é definido conforme Wazlawick (2013, p. 35):

> [...] como sendo fortemente orientado a riscos, sendo proposto por Boehm em 1986, como uma proposta de apresentar a ideia em ciclos iterativos. O projeto é dividido em subprojetos, cada qual abordando um ou mais elementos de alto risco, até que todos os riscos identificados tenham sido tratados.

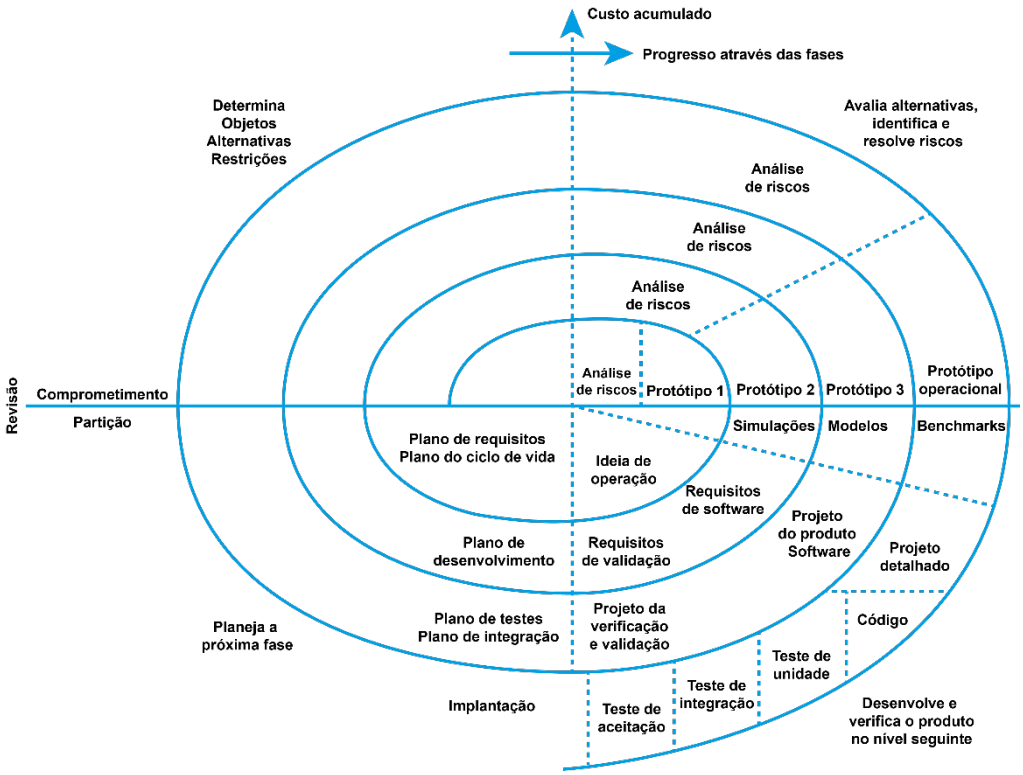
Nessa linha, Sommerville (2011) afirma que: “[...] o modelo espiral combina prevenção e tolerância a mudanças, assume que mudanças são um

resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua redução.”

De um modo mais otimista, Pressman (2011) coloca que o modelo espiral é uma abordagem realista do desenvolvimento de sistemas e *softwares* de grande porte.

Seguindo essas premissas, Wazlawick (2013, p. 36) reforça que a ideia dos ciclos do modelo espiral é que o projeto inicie com miniprojetos, abordando os principais riscos, e então seja expandido por meio da construção de protótipos, testes e replanejamento, de forma a abarcar os riscos identificados. Após ter adquirido um conhecimento mais completo dos potenciais problemas com o sistema, a equipe passará a desenvolver um ciclo final semelhante ao do modelo cascata. A Figura 6 apresenta a estrutura do modelo espiral.

Figura 6 – Modelo de processos espiral



Fonte: o autor.

De acordo com Pfleeger (2004, p. 46), o modelo espiral começa com os requisitos e um projeto inicial para o desenvolvimento do *software*, incluindo orçamento, restrições e alternativas para o pessoal, o projeto e o ambiente de desenvolvimento, inserindo em seguida uma etapa para avaliar os riscos e protótipos alternativos antes de ser produzido um documento em alto nível de

como o sistema deverá funcionar. Desse modo, cada uma das iterações produz os seguintes resultados:

- **primeira iteração**: concepção das operações;
- **segunda iteração**: os requisitos do produto de *software*;
- **terceira iteração**: o desenvolvimento do sistema;
- **quarta iteração**: os testes.

Segundo Wazlawick (2013, p. 37), uma das vantagens do modelo espiral é que as primeiras iterações são as mais baratas do ponto de vista de investimento de tempo e recursos, como também resolvem os maiores problemas do projeto. Ainda conforme Wazlawick, esse modelo não provê a equipe com indicações claras sobre a quantidade de trabalho esperada a cada ciclo, o que pode tornar o tempo de desenvolvimento nas primeiras fases bastante imprevisível.

3.2 Desenvolvimento em fases

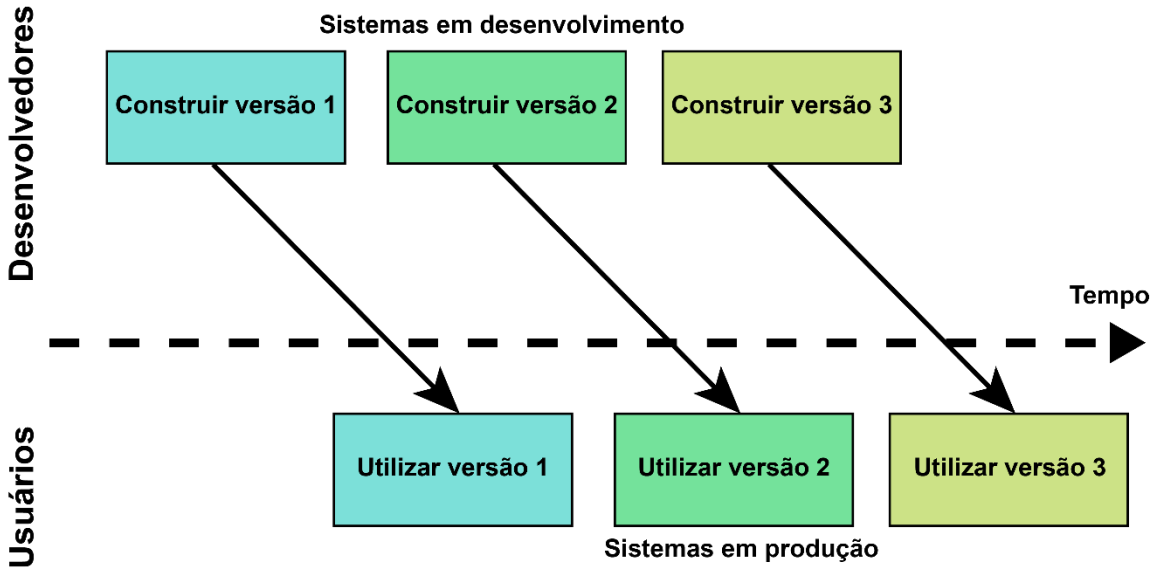
O modelo de desenvolvimento em fases, ou entrega em estágios, conforme abordado em Wazlawick (2013), tem como premissa diminuir o tempo de desenvolvimento e entrega do produto de *software* ao cliente. Conforme especificado por Pfleeger (2004, p. 45): “no modelo de desenvolvimento em fases, o sistema é projetado de modo que possa ser entregue em partes, possibilitando aos usuários dispor de alguns recursos enquanto o restante do sistema está sendo desenvolvido.”

Nesse modelo de processo, conforme Pfleeger (2004), existem dois sistemas em paralelo:

- **sistema em produção**: é o sistema que está sendo atualmente utilizado pelo cliente;
- **sistema em desenvolvimento**: é a versão seguinte que está sendo desenvolvida para substituir o sistema em produção.

A Figura 7 apresenta o modelo de desenvolvimento em fases.

Figura 7 – Modelo de desenvolvimento em fases



Fonte: Pfleeger, 2004, p. 45.

De acordo com Pfleeger (2004, p. 45), as duas principais abordagens para o desenvolvimento de *software* em fases são:

- **desenvolvimento incremental**: o sistema é dividido por funcionalidades e cada versão entregue corresponde a uma ou um conjunto dessas funcionalidades;
- **desenvolvimento iterativo**: a primeira versão entregue corresponde ao sistema completo, porém, com funcionalidades limitadas. Cada versão posterior corresponde a mudanças realizadas nas funcionalidades limitadas.

Para exemplificar os dois modos de desenvolvimento em fases, vamos analisar e compreender o exemplo proposto por Pfleeger (2004, p. 46):

O projeto a ser desenvolvido refere-se a um *software* de processamento de textos. Esse *software* deve oferecer os seguintes recursos:

- Criar textos;
- Organizar textos (recortar e colar);
- Formatar textos.

1. **Desenvolvimento no modelo incremental**:
   a. **versão 1**: função de criação de textos;
   b. **versão 2**: criação e organização de textos;

c. **versão 3**: criação, organização e formatação de textos.

2. **Desenvolvimento no modelo iterativo**:
   a. **versão 1**: formas primitivas das três funções. Podemos criar textos, mas não podemos recortar ou colar. Podemos formatar o texto, mas não podemos mudar a cor e tamanho da fonte;
   b. **versão 2**: as funções para recortar e colar são implementadas;
   c. **versão 3**: as funções para mudar a cor e tamanho da fonte são implementadas.

Segundo Pfleeger (2004, p. 46), muitas organizações utilizam uma combinação dos dois modos de desenvolvimento (incremental e iterativo), dado que uma nova versão pode trazer novos recursos, mas a funcionalidade existente na versão anterior pode também ser aprimorada.

TEMA 4 – PROTOTIPAÇÃO EVOLUCIONÁRIA

Nos tópicos anteriores, quando estudamos os modelos de processos de desenvolvimento de *software* cascata e espiral, pudemos observar que ambos os modelos de processos podem ser incrementados com prototipação para melhorar a compreensão do produto final tanto pelos desenvolvedores como pelos clientes. Segundo Pfleeger (2004, p. 42), a prototipação, ou conforme definido por Wazlawick (2013, p. 37) de prototipação evolucionária, não precisa ser somente uma atividade para incrementar outros modelos de processos, de modo que ela própria pode ser a base de um modelo de processo efetivo. Seguindo essa abordagem, Pfleeger (2004, p. 42) ainda afirma que:

> O modelo de prototipação permite que todo o sistema, ou parte dele, seja construído rapidamente para que dúvidas sejam entendidas ou esclarecidas. Esse modelo tem o mesmo objetivo de um protótipo de engenharia, quando os requisitos ou o projeto necessitam de investigações repetidas para garantir que o desenvolvedor, usuário e cliente cheguem a um consenso sobre o que é necessário e o que é proposto.

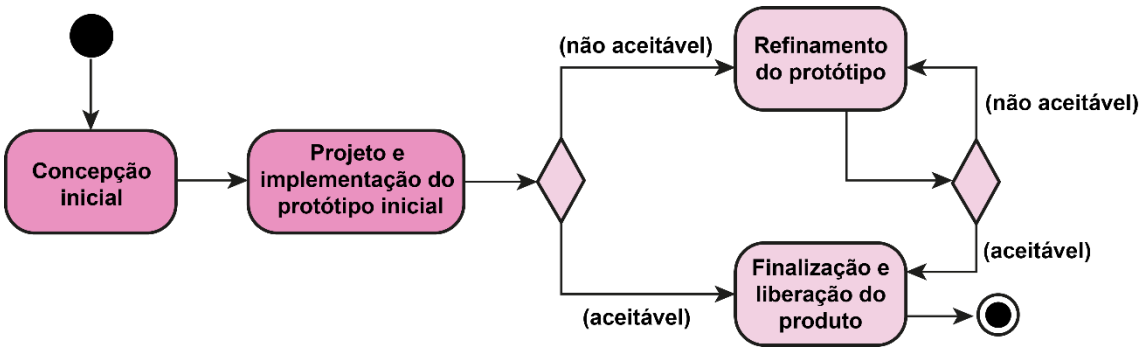
Nesse contexto, Wazlawick (2013, p. 37) apresenta dois possíveis métodos para aplicação do modelo de processos de prototipação:

- ***Throw-away*** – também denominado *descartável*, consiste na construção de protótipos que são usados unicamente para estudar aspectos do sistema, entender melhor seus requisitos e reduzir riscos;

- ***Cornerstone*** – também denominado *fundamental*, consiste na construção de protótipos para compreender todos os aspectos da abordagem *throw-away* e, para serem parte do sistema final, de modo que o protótipo vai evoluindo até se tornar um sistema que possa ser entregue.

A Figura 8 apresenta o modelo de prototipação evolucionária abordada em Wazlawick (2013, p. 37).

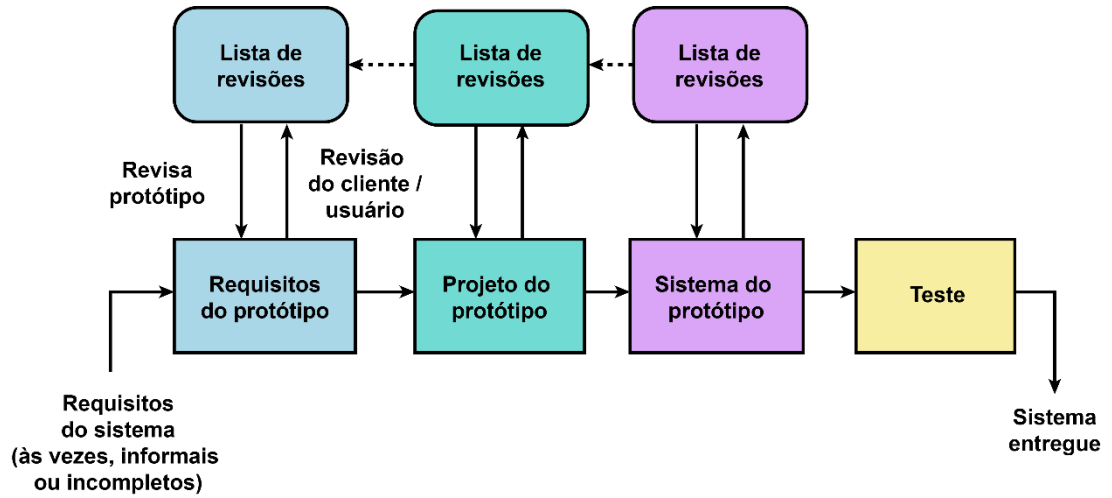
Figura 8 – Modelo de prototipação evolucionária



Fonte: Wazlawick, 2013, p. 37.

Já na concepção de Pfleeger (2004, p. 43), a prototipação segue um modelo orientado por revisões em cada uma das etapas, conforme pode ser observado na Figura 9.

Figura 9 – Modelo de prototipação



Fonte: Pfleeger, 2004, p. 43.

Fazendo uma analogia entre o modelo proposto em Wazlawick (2013) e o modelo proposto em Pfleeger (2004), podemos fazer a seguinte análise:

- a fase de concepção inicial de Wazlawick corresponde à fase de requisitos do protótipo de Pfleeger;
- a fase de projeto e implementação do protótipo inicial de Wazlawick corresponde à fase de projeto do protótipo de Pfleeger;
- a fase de refinamento do protótipo de Wazlawick corresponde a cada uma das fases de lista de revisões de Pfleeger;
- a fase de finalização e liberação do produto de Wazlawick corresponde à fase de Sistema do Protótipo de Pfleeger;
- já a última fase de testes de Pfleeger corresponde às diversas iterações possíveis na fase de refinamento do protótipo de Wazlawick.

Para um maior esclarecimento do modelo de processos de prototipação evolucionária, vamos visualizar e analisar o exemplo apresentado por Pfleeger (2004, p. 43):

> O desenvolvimento do sistema pode começar com um conjunto simples de requisitos fornecidos pelos clientes e usuários. Examinam-se as telas, tabelas, relatórios e outras saídas do sistema, diretamente utilizadas pelos clientes e usuários. Assim que os clientes e usuários decidem o que realmente querem, os requisitos são revisados. Uma vez que haja consenso de como deveriam ser os requisitos, o desenvolvedor se volta para as atividades de requisitos, a fim de reconsiderar e alterar a especificação. Finalmente o sistema é codificado, e as alternativas, discutidas, de novo com uma possível iteração entre requisitos e projeto.

Analisando o exemplo apresentado por Pfleeger (2004, p. 43), percebemos que ele se encaixa perfeitamente no modelo de prototipação apresentado na Figura 8, de Wazlawick (2013), diferenciando-se apenas que, enquanto no modelo apresentado por Pfleeger as revisões acontecem desde a fase de elicitação dos requisitos, no modelo de Wazlawick as revisões e possíveis alterações ocorrerão somente após a implementação da primeira versão do protótipo.

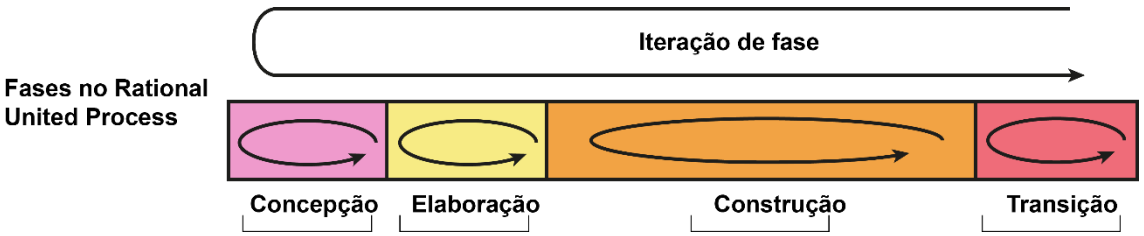
TEMA 5 – PROCESSO UNIFICADO DA RATIONAL (RUP)

Como vimos nesta aula, os modelos de processos estudados apresentam uma visão única do processo. Contrária a essa metodologia, o *Rational Unified Process* (RUP), ou Processo Unificado da Rational (empresa em que o RUP foi criado), é um modelo híbrido de processo que conforme Sommerville (2018, p. 32) aborda quatro fases no processo de *software*:

- **concepção** – fase em que é elaborado um plano de negócios para o sistema, com o objetivo de identificar as entidades externas e os requisitos do sistema a fim de avaliar a contribuição do sistema para o negócio;
- **elaboração** – fase em que são desenvolvidos os requisitos e a arquitetura do sistema;
- **Construção** – implementação e testes do sistema;
- **Transição** – implantação do sistema em ambiente real.

O RUP, de acordo com Sommerville (2018, p. 32), reúne os elementos de todos os modelos de processos que estudamos nesta aula, apoiando inclusive a prototipação e a entrega incremental do *software*. A Figura 10 apresenta o modelo de processos RUP.

Figura 10 – Modelo de processos RUP



Fonte: o autor.

Observando o modelo RUP apresentado na Figura 10 anterior, podemos notar que cada fase pode ser realizada seguindo o modo iterativo do modelo de processos de entrega em fases, com os resultados desenvolvidos no modo incremental. Também é possível observar que as quatro fases juntas podem ser aplicadas de modo incremental ou seguir a metodologia proposta no modelo espiral.

Para cada uma das quatro fases, de acordo com Wazlawick (2013), o modelo RUP apresenta um conjunto de seis atividades lógicas de projeto e três de apoio, denominadas disciplinas.

1. **Disciplinas de projeto**
   a. Modelagem de negócios: visa estudar e compreender a empresa e seus processos para descrever as regras de negócio;
   b. Requisitos: tem como objetivo a elicitação dos requisitos e também o *design* da interface do sistema;
   c. Análise e *design*: tem como foco o detalhamento dos requisitos para elaboração da modelagem do sistema;

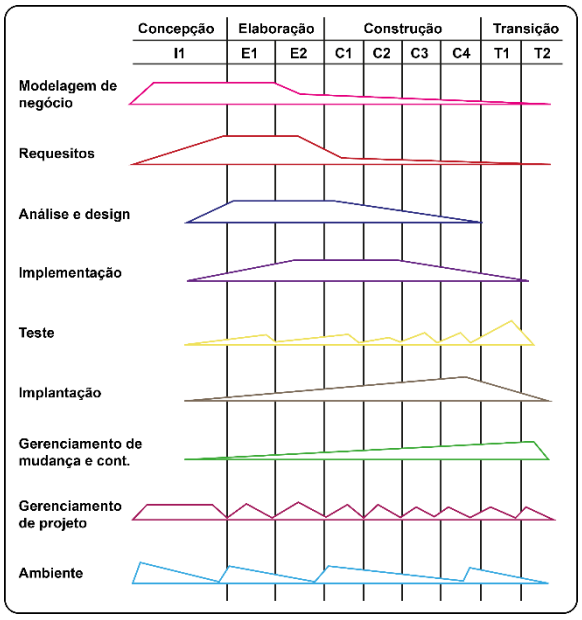
d. Implementação: foca no desenvolvimento do sistema e testes de unidade;
e. Teste: exclui os testes de unidade que já foram realizados na disciplina de implementação e foca nos testes de integração e interação entre os componentes;
f. Implantação: produção de versões do sistema para serem entregues aos clientes.

2. Disciplinas de apoio

a. Ambiente: aborda o ambiente de desenvolvimento do sistema, focando na configuração do processo a ser usado para desenvolver o projeto. Essa disciplina deve também tratar das ferramentas de apoio necessárias para que a equipe tenha sucesso no projeto;
b. Gerenciamento de mudança e configuração: tem como objetivo manter a integridade do conjunto de artefatos produzidos ao longo do projeto;
c. Gerência de projeto: tem como objetivo planejar o projeto como um todo, planejar cada iteração individualmente, gerenciar os riscos do projeto e monitorar o progresso.

Na Figura 11 podemos observar a ênfase de cada uma dessas disciplinas em cada uma das quatros fases do modelo RUP.

Figura 11 – Disciplinas do modelo RUP



Fonte: Wazlawick, 2013, p. 87.

FINALIZANDO

Nesta aula, abordamos de modo geral os principais modelos de processos prescritivos de *software*, em que pudemos compreender as formas de aplicação de cada modelo, como as vantagens e desvantagens de cada um.

Aprendemos que, mesmo sendo um dos primeiros modelos de processos de *software*, o modelo cascata foi muito utilizado no decorrer dos anos, inclusive sendo aplicado nos dias atuais e, mesmo apresentando diversas dificuldades, serviu como base para o surgimento dos modelos posteriores.

Também pudemos compreender que, com o surgimento de vários modelos de processos de desenvolvimento de *software*, a engenharia de *software* não define um modelo como um padrão a ser adotado, ficando a critério da equipe de desenvolvimento escolher o modelo mais adequado para a aplicação a ser desenvolvida.

A cada novo modelo de processo criado, várias dificuldades identificadas em modelos anteriores foram corrigidas. Porém, a forma de aplicação de cada modelo, causando certa demora em todo o processo, sempre foi uma das principais reclamações. Nesse contexto, começaram a surgir modelos de processos inovadores, identificados como métodos ágeis de desenvolvimento de *software*, os quais abordaremos futuramente.

Portanto, aprenderemos no futuro sobre os avanços no gerenciamento de projetos de *software* com o surgimento dos métodos ágeis, os quais não necessariamente substituem os modelos de processos prescritivos, mas sim, muitas vezes, os complementam.

REFERÊNCIAS

PFLEEGER, S. L. **Engenharia de *Software***: teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, R. S. **Engenharia de *Software***: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

ROYCE, W. W. Managing the Development of Large *Software* Systems. ***Proceedings of* IEEE WESCON**, 1970.

SOMMERVILLE, I. **Engenharia de *Software***. 10 ed. São Paulo: Pearson Education do Brasil, 2018.

SOMMERVILLE, I. **Engenharia de *Software***. 9. ed. São Paulo: Pearson, 2011.

WAZLAWICK, R. S. **Engenharia de *Software***: conceitos e práticas. São Paulo: Elsevier, 2013.