Fundamentos Engenharia de Software

UNIDADE 01

Modelos de Processos de desenvolvimento de Software

Esta Unidade tem início com uma breve introdução dos principais conceitos que envolvem a engenharia de software. Na sequência, apresenta e discute os modelos de processo de desenvolvimento de software. Sobre os modelos, será apresentada a sua evolução, funcionamento e as situações mais propícias de aplicação. Basicamente, os modelos de processos de desenvolvimento de software são divididos prescritivos e ágeis. Os prescritivos foram os primeiros modelos aplicados no desenvolvimento de software e tinham como característica principal determinar de forma mais rígida como as atividades deveriam ser realizadas. Com a evolução e a identificação das deficiências dos modelos

prescritivos, surgiram os modelos/métodos ágeis, que trouxeram mais flexibilidade em como as atividades são organizadas para o desenvolvimento de um produto de software.

Fundamentos da engenharia de software

Durante o processo de desenvolvimento do *software*, as equipes buscam atingir objetivos que são cruciais para o sucesso do projeto. Eles normalmente são: desenvolver o produto correto, ou seja, que resolva o problema do cliente; desenvolver de forma rápida, no menor tempo possível; desenvolver um produto barato, portanto, com o menor custo possível, e assim tornar-se competitivo. No entanto, tais objetivos somente serão alcançados com a aplicação dos conceitos da engenharia de *software*.

O *software* hoje é onipresente, também chamado de ubíquo. Ou seja, ele está presente em todas as atividades diárias (profissionais ou pessoais) e, muitas vezes, nem percebemos a sua existência. O seu uso já foi incorporado como algo natural em nossas vidas. Porém, esta presença em tantos ambientes distintos trouxe um ambiente de desenvolvimento muito mais complexo do que há décadas.

Ambientes mais complexos demandam tecnologias, processos e composições de equipes mais complexas também. O trabalho entre pessoas distribuídas geograficamente é uma realidade comum no desenvolvimento de *software*. Portanto, cada vez mais, precisamos dos métodos, dos processos e das ferramentas propostos pela engenharia de *software* para entregar um produto de *software* com qualidade.

Mas o que é qualidade de *software?*

Há diversas definições e pontos de vistas distintos relacionados a este conceito. Uma dessas definições diz:

> *Uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam (PRESSMAN, 2016)*

A base para a engenharia de software é a camada de processos. O processo de engenharia de software é a liga que mantém as camadas de tecnologias coesas e possibilita o desenvolvimento de software com qualidade. Assim, as palavras-chave na engenharia de software são: qualidade e processos, ambas estão intrinsicamente relacionadas.

Na literatura, encontramos várias definições para a área de engenharia de software. Uma das definições mais claras e objetiva expõe que a

> *engenharia de software é o processo de estudar, criar e otimizar os processos de trabalhos para os desenvolvedores de software" (WAZLAWICK, 2019)*

Áreas do conhecimento

O SWEBOK (guia de referência sobre conhecimentos de engenharia de *software*), versão 2013, classifica a engenharia de *software* em 15 áreas do conhecimento.

1. Requisitos de *software*.
2. Projeto de *software*.
3. Construção de *software*.
4. Teste de *software*.
5. Manutenção de *software*.
6. Gerência de configuração de *software*.
7. Gerência da engenharia de *software*.
8. Processos de engenharia de *software*.
9. Ferramentas e métodos da engenharia de *software*.
10. Qualidade de *software*.
11. Práticas profissionais em engenharia de *software*.
12. Economia da engenharia de *software*.

13. Fundamentos de computação.

14. Fundamentos de matemática.

15. Fundamentos de engenharia.

Por se tratar de um grande conjunto de áreas, os assuntos tratados nesta disciplina estão mais relacionados às seguintes áreas: processos de engenharia de *software*, gerência da engenharia de *software*, teste de software e qualidade de *software*. Os demais assuntos são tratados de forma distribuída, em outras disciplinas.

A seguir, serão apresentados alguns dos modelos de processo de desenvolvimento de *software* mais populares das últimas décadas na indústria do desenvolvimento.

Modelos de processos de desenvolvimento de software

Conceitos-chave

Antes de discutirmos a respeito dos modelos de processo de desenvolvimento de *software*, é importante entendermos conceitos-chave, tais como o ciclo de vida e de desenvolvimento do *software*.

O **ciclo de vida de um *software*** é semelhante ao ciclo de vida que encontramos na natureza. Ele representa o conjunto de **transformações** que certos indivíduos de uma espécie podem passar. O *software* também passa por um conjunto de transformações, o que é chamado de **ciclo de vida do *software*.** Independentemente do seu tamanho e da sua complexidade, o *software* sempre terá o ciclo de vida ilustrado a seguir, na Figura 1.

Figura 1: Ciclo de vida do *software*

Representação das principais etapas (transformações) do *software* ao longo da sua existência. Fonte: Autor (2021).

Ou seja, todo projeto de *software* tem início com o seu estudo de **viabilidade**. Sendo viável, parte-se para o **desenvolvimento**, os produtos acabados são **implantados**, ao longo do tempo ocorrem **manutenções** neles, até que um dia são **descontinuados**.

Muitas vezes, a descontinuação ocorre pela necessidade de atualização para uma nova tecnologia.

No entanto, antes de avançarmos, é importante entendermos outro conceito-chave: **ciclo de desenvolvimento do *software***. No desenvolvimento do *software*, cada projeto pode estabelecer um conjunto de atividades e a sua sequência, o que chamamos de **processo**. Processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum produto de trabalho. Com isso, cada projeto pode ter um processo diferente, de acordo com o seu contexto.

Aqui, é importante chamarmos a atenção para a diferença entre estes dois conceitos: ciclo de vida do software e ciclo de desenvolvimento do software. O ciclo de vida do *software* é sempre o mesmo, mas o ciclo de desenvolvimento do *software* pode mudar para cada projeto, dependendo das características e do seu contexto.

Modelos de processos de desenvolvimento

Há várias **maneiras de organizar** um processo de desenvolvimento de *software* para evitar o caos. Portanto, conheceremos alguns dos **principais modelos de processos** propostos e aplicados no mercado.

É importante ressaltar que, independentemente do tamanho e da complexidade de um projeto de *software*, todos comportarão minimamente as seguintes atividades de desenvolvimento:

- Especificação de requisitos (entender o negócio e as necessidades do cliente).
- Análise e projeto (planejar e projetar *o software*).
- Construção (implementar o *software*).
- Testes (testar o *software*).
- Implantação (implantar o *software* desenvolvido).

Os modelos de processo normalmente ajudam as equipes a organizarem as suas atividades de maneira a atingir o sucesso no desenvolvimento do *software*. O modelo mais adequado é aquele que alinha as características do projeto com as características

do modelo. E, com certeza, adotar um modelo de processo de *software*, seja qual for, é muito melhor do que não ter nenhum.

Por volta dos anos 70, surgiram modelos de processo de *software* mais tradicionais, conhecidos como prescritivos, que contribuíram para trazer ordem ao caos existente no desenvolvimento de *software*. No entanto, tais modelos apresentavam limitações. Com o objetivo de suprir essas limitações, foram propostos os modelos conhecidos por ágeis.

Na sequência, será exposto o conceito de cada um desses modelos (prescritivos e ágeis) e identificar as situações favoráveis e os pontos de atenção de cada modelo.

Modelos prescritivos

Os modelos prescritivos se baseiam em uma descrição de como as atividades devem ser feitas. São diversos os modelos propostos e que evoluíram com este objetivo. Nesta Unidade, conheceremos alguns dos mais utilizados ao longo da história da engenharia de *software*.

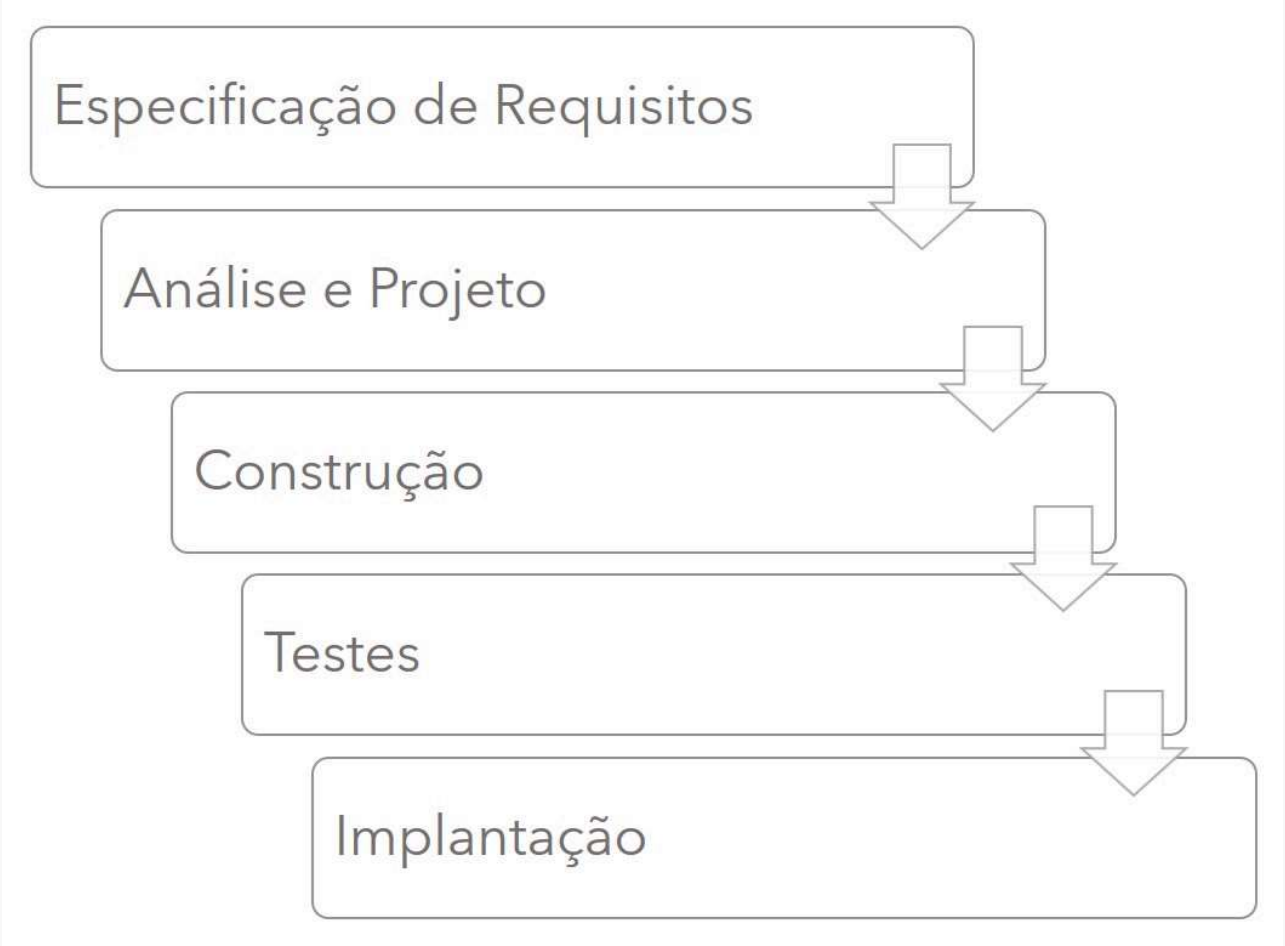
A seguir, estão apresentados alguns detalhes referentes aos modelos: cascata, incremental, evolucionário (prototipação e espiral) e unificado.

MODELO CASCATA

O modelo cascata segue uma abordagem **sequencial** e **sistemática**. Inicia com levantamento de requisitos e avança para as demais atividades quando se entende que a atual está completa.

A Figura 2, apresentada a seguir, representa o ciclo de desenvolvimento de um *software* proposto no modelo cascata.

Figura 2: Modelo cascata



Ciclo de desenvolvimento no modelo cascata. Fonte: Autor (2021).

Situações favoráveis para adoção do modelo cascata:

- Os requisitos de software estão bem compreendidos.
- Os requisitos de software são estáveis, ou seja, quando não mudam.
- O prazo para uso da solução não é urgente.

Isso significa que o modelo pode funcionar melhor em um contexto em que o domínio da aplicação e as suas regras de negócio são conhecidos pela equipe e que as mudanças de requisitos não são frequentes. Ou, ainda, quando não há uma urgência de prazo para a sua implantação. No entanto, sabe-se que estas situações praticamente não existem. Muitas vezes, o conhecimento do domínio inicia com o desenvolvimento do projeto, as mudanças acabam sendo frequentes, principalmente em cenários competitivos ou quando envolvem legislações. E, por fim, não existir urgência de prazo pode significar que o projeto não é relevante.

Pontos de atenção na adoção do modelo cascata:

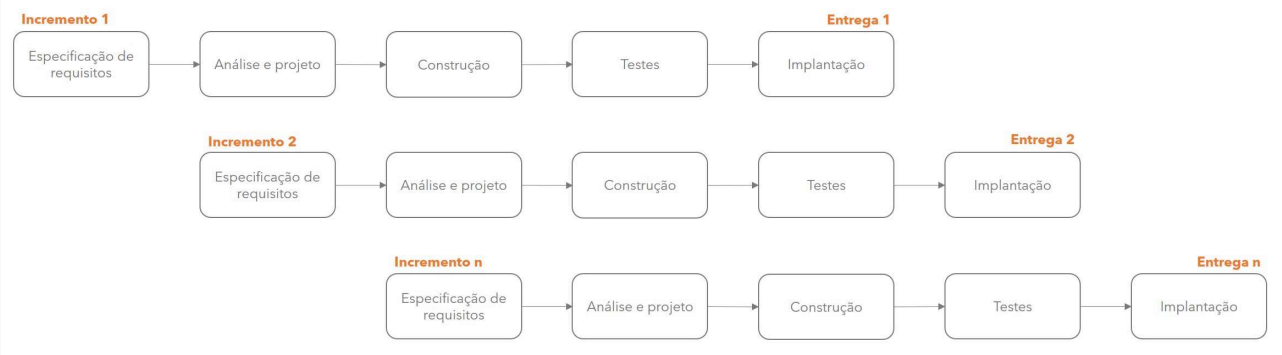
- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe.
- Dificilmente, o cliente consegue estabelecer explicitamente todas as necessidades no início do projeto.
- O cliente precisa ter paciência para obter uma versão testável, uma vez que uma versão operacional fica disponível apenas muito próximo do final do projeto.

MODELO INCREMENTAL

O modelo incremental tem início com um subconjunto simples de requisitos, e incrementalmente realizam-se entregas de versões até o sistema todo estar implementado. O modelo combina elementos dos fluxos de processos lineares e paralelos.

A Figura 3, apresentada a seguir, representa o ciclo de desenvolvimento de um *software* proposto no modelo incremental.

Figura 3: Modelo incremental



Ciclo de desenvolvimento no modelo incremental. Fonte: Autor (2021).

Quando se usa um modelo incremental, normalmente, o primeiro incremento é um produto essencial. Ou seja, os requisitos básicos são atendidos, porém, muitos recursos complementares ou mais avançados ainda não são entregues. No planejamento do incremento, já se considera a modificação do produto essencial para se adequar às necessidades.

Situações favoráveis para adoção do modelo incremental:

- Os requisitos de software estão bem compreendidos.
- Os requisitos de software são estáveis, ou seja, quando não mudam.
- O prazo para uso da solução não é urgente.

Pontos de atenção na adoção do modelo incremental:

- A funcionalidade entregue para validação pode ainda não estar completa nos primeiros incrementos.
- Podem ocorrer diversas modificações em torno da mesma funcionalidade.

MODELOS EVOLUCIONÁRIOS (PROTOTIPAÇÃO E ESPIRAL)

Os modelos evolucionários surgiram da observação de que a dinâmica de *software* muda frequentemente (concorrência, legislação, negócios etc.), tornando inviável seguir um planejamento em linha reta, conforme proposto nos modelos anteriores.

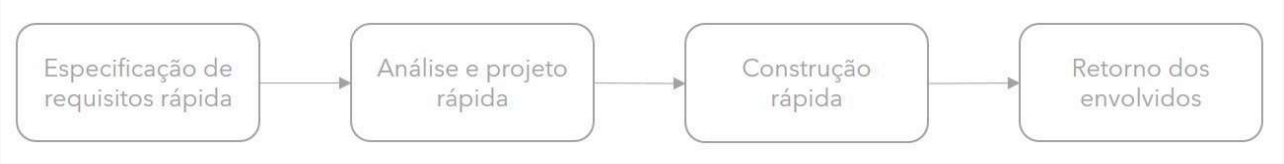
Em diversas situações, faz-se necessário um modelo de processo que tenha sido projetado especificamente para desenvolver um produto que evolua ao longo do tempo. Ou seja, um modelo que possibilite desenvolver versões cada mais completas do *software*. A seguir, estão apresentados dois tipos de modelos evolucionários que apresentam estas características: a prototipação e o espiral.

PROTOTIPAÇÃO

A prototipação pode ser utilizada como um modelo de processo isolado, mas é mais comum que seja aplicada em conjunto com outro modelo de processo citado nesta Unidade, ou ainda, outros modelos de processo existentes.

A Figura 4, apresentada a seguir, representa o ciclo de desenvolvimento de um *software* proposto no modelo prototipação.

Figura 4: Modelo evolucionário: prototipação



Ciclo de desenvolvimento no modelo prototipação. Fonte: Autor (2021).

Normalmente, a execução do processo tem início com uma reunião para entender os requisitos gerais. O sistema é analisado e projetado de acordo com os requisitos gerais. Do projeto se constroem os protótipos, principalmente as interfaces e os *layouts*. Os envolvidos fornecem um retorno (*feedback*), que servirá para aprimorar os requisitos. Dessa maneira, o protótipo atua como um mecanismo para identificar os requisitos do *software*. Posteriormente, seguem as etapas para o desenvolvimento do produto definitivo.

Situações favoráveis para adoção do modelo prototipação:

- Quando os requisitos ainda não estão claros.
- O protótipo é construído de forma rápida e ajuda a entender o que deve ser desenvolvido.
- O cliente tem uma ideia prévia do que será o sistema.

Pontos de atenção na adoção do modelo prototipação:

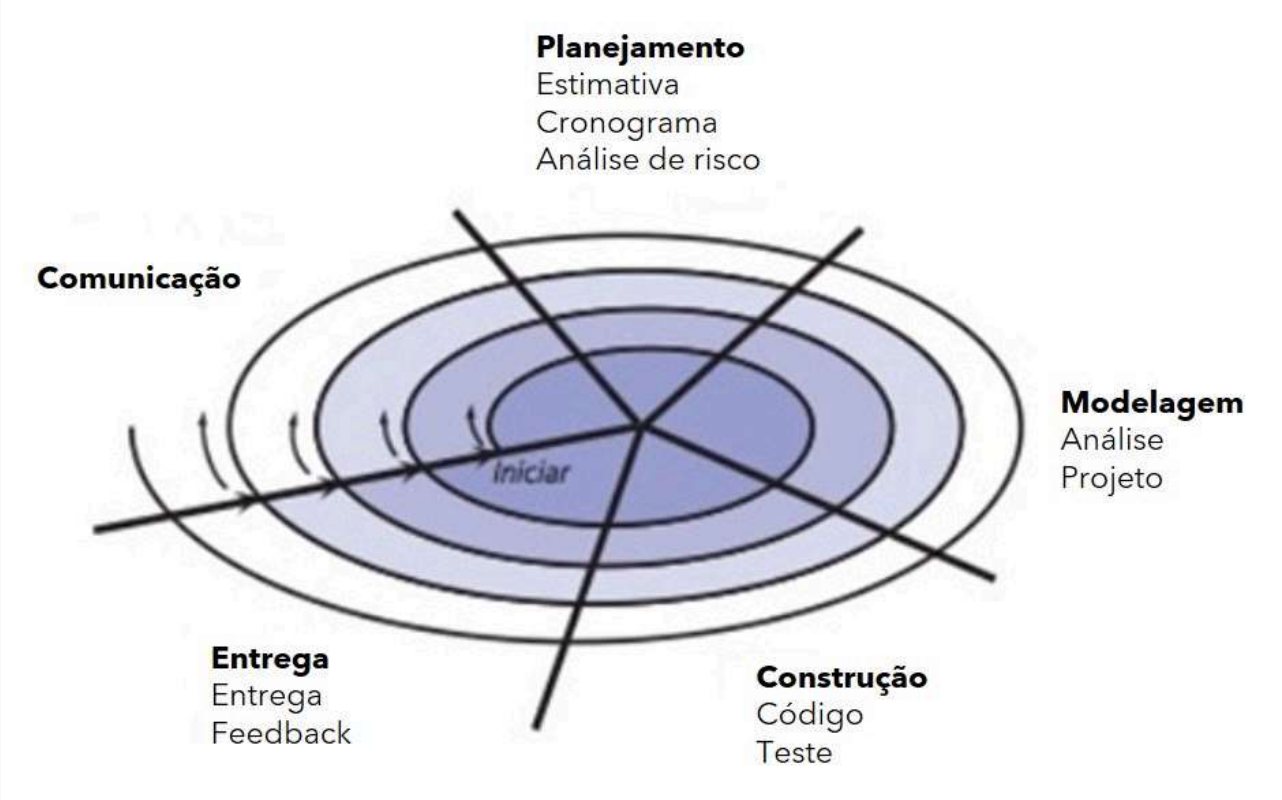
- As versões iniciais são descartáveis.
- O cliente, muitas vezes, não compreende que o protótipo não é a versão final e que posteriormente o produto deverá ser reconstruído.
- O desenvolvedor pode se acomodar com os resultados provisórios e torná-los definitivos.

ESPIRAL

O modelo espiral é uma combinação de prototipação (iterativo) e etapas de processos lineares (ex.: cascata).

Analise a Figura 5, apresentada a seguir, para entender o funcionamento do modelo.

Figura 5: Modelo evolucionário: espiral



Ciclo de desenvolvimento no modelo prototipação. Fonte: PRESSMAN, R. S. ; MAXIM, B. R. (2016, p. 48).

Iniciamos pelo centro da espiral e prosseguimos no sentido horário. A primeira atividade é a especificação do *software.* As próximas passagens podem ser usadas para desenvolver um protótipo e, assim, sucessivamente, evoluímos para versões cada vez mais sofisticadas do *software*. Cada passagem resulta em ajustes no planejamento.

Situações favoráveis para a adoção do modelo espiral:

- Quando os requisitos ainda não estão claros.
- Sistemas de larga escala.
- Combina abordagem sistemática, sugerida pelo modelo cascata, mas incorpora métodos iterativos (prototipação).

Pontos de atenção na adoção do modelo espiral:

- É necessária boa experiência da equipe para controlar a evolução do projeto.
- Não indicado para projetos menores.

MODELO UNIFICADO

O último modelo a ser apresentado nesta Unidade é o unificado, um dos últimos modelos prescritivos a serem propostos antes da expansão dos modelos ágeis.

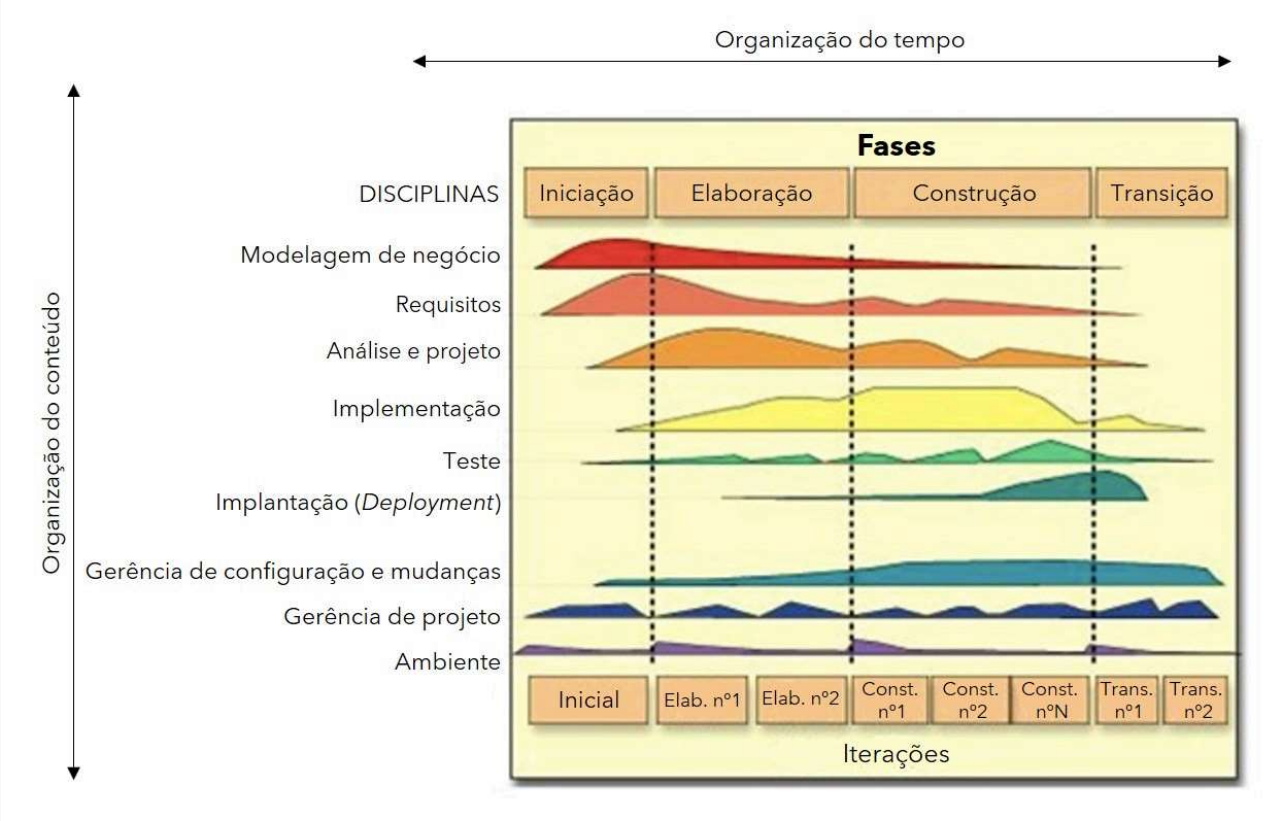
Na tentativa de usar os melhores recursos e características dos modelos prescritivos, mas também incorporando princípios ágeis, foi proposto o modelo unificado, por Jacobson, Rumbaugh e Booch (os três amigos, “pais” da UML), o qual usa a abordagem da orientação a objetos e notação *Unified Modeling Language* (UML) para mostrar o processo em ação.

As principais características deste modelo são o desenvolvimento iterativo e incremental.

Com base nesta proposta, a IBM criou o *Rational Unified Process* (**RUP**). Ele não é apenas um modelo, mas também um produto de *software*. Toda a sua metodologia é apoiada por diversas ferramentas de desenvolvimento integradas e vendidas.

A Figura 6, apresentada a seguir, representa as fases propostas no modelo, a intensidade de certas áreas do conhecimento (chamadas de disciplina) dada em cada fase e as iterações que podem ocorrer.

Figura 6: Modelo unificado



Ciclo de desenvolvimento no RUP (baseado no modelo unificado). Fonte: Adaptado de KRUCHTEN, 2001.

Ele é considerado um modelo pesado e preferencialmente aplicável a grandes equipes de desenvolvimento e a grandes projetos, porém, o fato de ser amplamente customizável torna possível que seja adaptado para projetos de qualquer escala.

Modelos ágeis

Nesta seção, estão apresentadas brevemente algumas das informações mais relevantes sobre os métodos ágeis. Por se tratar de um assunto vasto, você poderá se aprofundar em disciplinas posteriores ou ainda buscar atividades de extensão.

Os métodos ágeis surgiram para suprir a necessidade de mudanças rápidas no processo de desenvolvimento de *software*, necessidade que não era atendida integralmente nos modelos prescritivos.

Os métodos ágeis seguem uma filosofia que foca principalmente nos resultados. O modelo de entrega é baseado em ciclos iterativos (integração entre as fases) e incrementais (entregas parciais).

Até a década de 90, as diretrizes eram:

- Um planejamento cuidadoso do projeto.
- Um processo de *software* rigoroso e controlado.

O termo método ágil surgiu durante a formação do manifesto ágil, em 2001: um grupo de 17 pessoas se reuniu para discutir sobre uma nova abordagem para a gestão de projetos de software. Ao final dessa reunião, todas as pessoas presentes assinaram o manifesto ágil.

Neste manifesto (https://agilemanifesto.org/?utm_source=angelopublio.com.br) são apresentados os valores e os princípios dos métodos ágeis. Os quatro valores principais destacados no modelo são:

---

1. **Indivíduos e interações mais do que** processos e ferramentas.
2. ***Software* em funcionamento mais do que** documentação abrangente.
3. **Colaboração com o cliente mais do que** negociação de contrato.
4. **Responder a mudanças mais do que** seguir um plano.

Os conceitos em verde negrito não anulam a existência dos conceitos em verde que não estão em negrito, apenas reforçam onde as ações devem ser focadas para obter os resultados esperados.

Ao longo do tempo, foram desenvolvidos diversos modelos de desenvolvimento de *software*, considerados ágeis:

- *EXtreme Programming (XP).*
- *Scrum.*
- *Feature-Driven Development (FDD).*
- *Dynamic Systems Development Method (DSDM).*
- *Crystal clear.*
- *Adaptative Software Development (ASD).*
- *Kanban.*

Muitos deles foram propostos muito antes do manifesto ágil. Os problemas nos modelos prescritivos já haviam sido identificados por volta da década de 80. A partir de então, as pessoas envolvidas com o processo de desenvolvimento de *software* iniciaram propostas de modelos que pudessem trazer mais qualidade para o processo de desenvolvimento de *software*. O manifesto ágil acabou sendo um momento para criar um marco que definiu uma nova era de modelos para desenvolvimento de *software.*

A seguir, serão apresentadas algumas das principais características dos modelos ágeis mais adotados pela indústria do *software.*

EXTREME PROGRAMMING (XP)

Os primeiros trabalhos relacionados ao XP tiveram início nos anos 80, bem antes do manifesto ágil (2001). Os valores principais do XP são: comunicação, simplicidade e *feedback* (retorno).

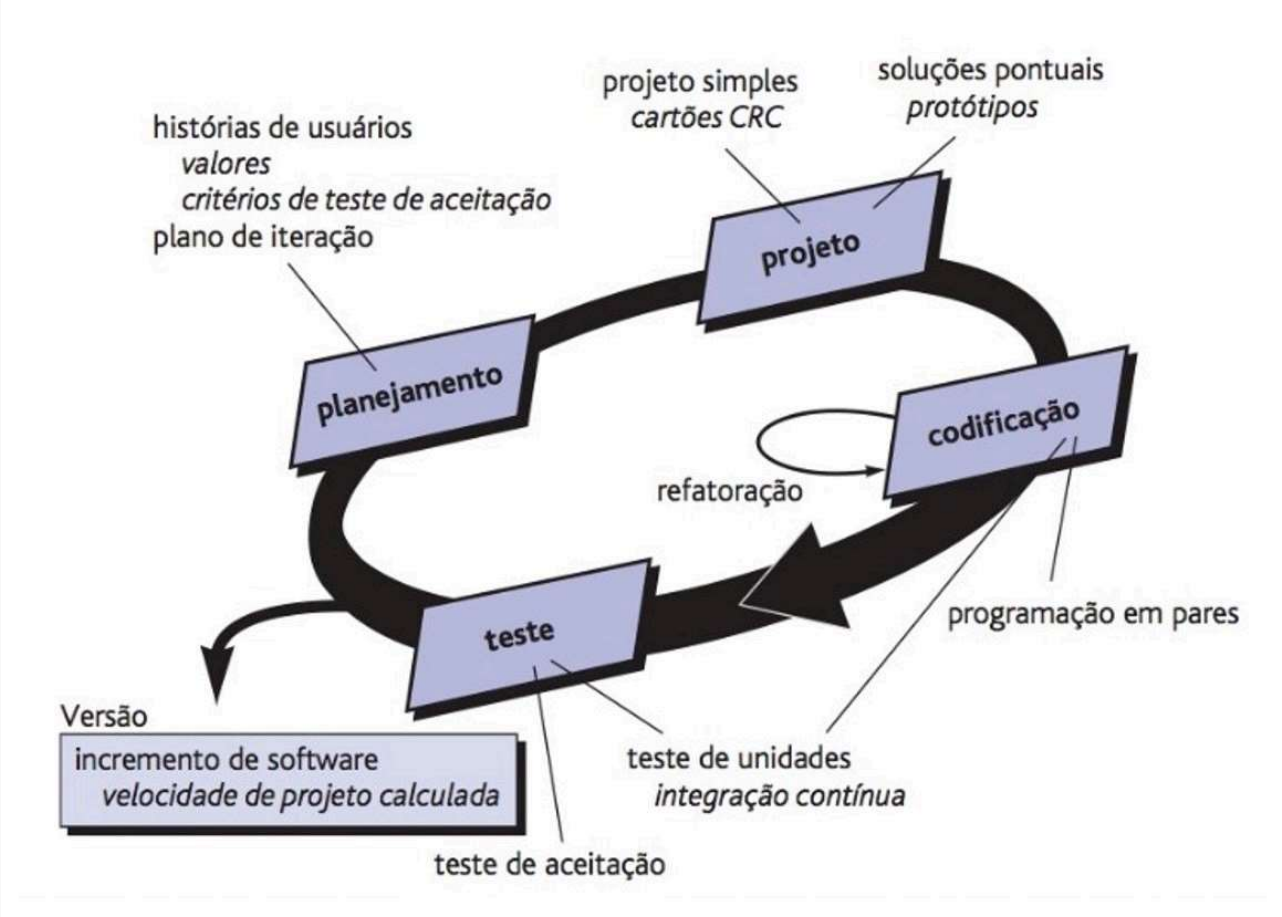
Neste modelo, são projetadas apenas as necessidades imediatas, sem considerar as futuras. Portanto, isso é importante muita disciplina. O XP preconiza:

- Ciclos curtos para dar previsibilidade e redução de incertezas/riscos.
- Simplicidade e melhorias constantes de código (*refactoring*) para facilitar a mudança.

- Testes automatizados e integração contínua para aumentar a confiança.

A Figura 7 apresenta o processo XP e destaca alguns conceitos e tarefas-chave associados às atividades.

Figura 7: Modelo XP



Representação das principais atividades e práticas propostas no modelo XP. Fonte: PRESSMAN, R. S.; MAXIM, B. R. (2016, p. 72).

O objetivo da atividade de **planejamento** é entender o ambiente de negócio do *software*, o que conduzirá para a criação de um conjunto de histórias. Cada história é estimada e priorizada e, se necessário, desmembrada em outras histórias para reduzir a granularidade. Clientes e membros da equipe trabalham em conjunto nesta atividade.

A atividade de **projeto** segue o princípio *Keep It Simple* (KIS), ou seja, em tradução livre, “mantenha a simplicidade”. Nesta atividade, é encorajado o uso dos cartões Classe-Responsabilidade-Colaborador (CRC), mecanismo que ajuda a pensar sobre o *software*, considerando o paradigma da orientação a objetos. Os protótipos são encorajados para reduzir riscos de estimativas com falhas. O projeto é refeito continuamente conforme o andamento do desenvolvimento e, portanto, poucos artefatos formais são produzidos, pois o projeto é visto como algo transitório.

A atividade de **codificação** não começa exatamente com a criação dos códigos, mas com o desenvolvimento de testes de unidades que exercitarão as histórias identificadas. Depois da criação dos testes, o desenvolvedor se dedicará à implementação do que foi aprovado no teste. Outro conceito-chave nesta atividade é a programação em duplas. Esta prática amplia a possibilidade de resolução de problemas em tempo real, aumentando a qualidade do *software*, uma vez que duas cabeças pensam melhor do que uma, certo?

Por fim, a atividade de **testes** inicia antes da codificação e é finalizada após ela. Como a codificação é derivada dos testes criados, após a codificação, os testes estão praticamente prontos para a automatização. A automatização facilita a execução diária dos testes de integração e de validação e, consequentemente, a identificação cedo dos problemas.

Situações favoráveis para a adoção do modelo XP:

- Embora alguns digam que é mais adequado para projetos menores, há registros de sucesso em grandes projetos.
- Time entrosado e cliente próximo.
- Abertura para mudança e retrabalho (desapego de protótipos, códigos ruins).

Pontos de atenção na adoção do modelo XP:

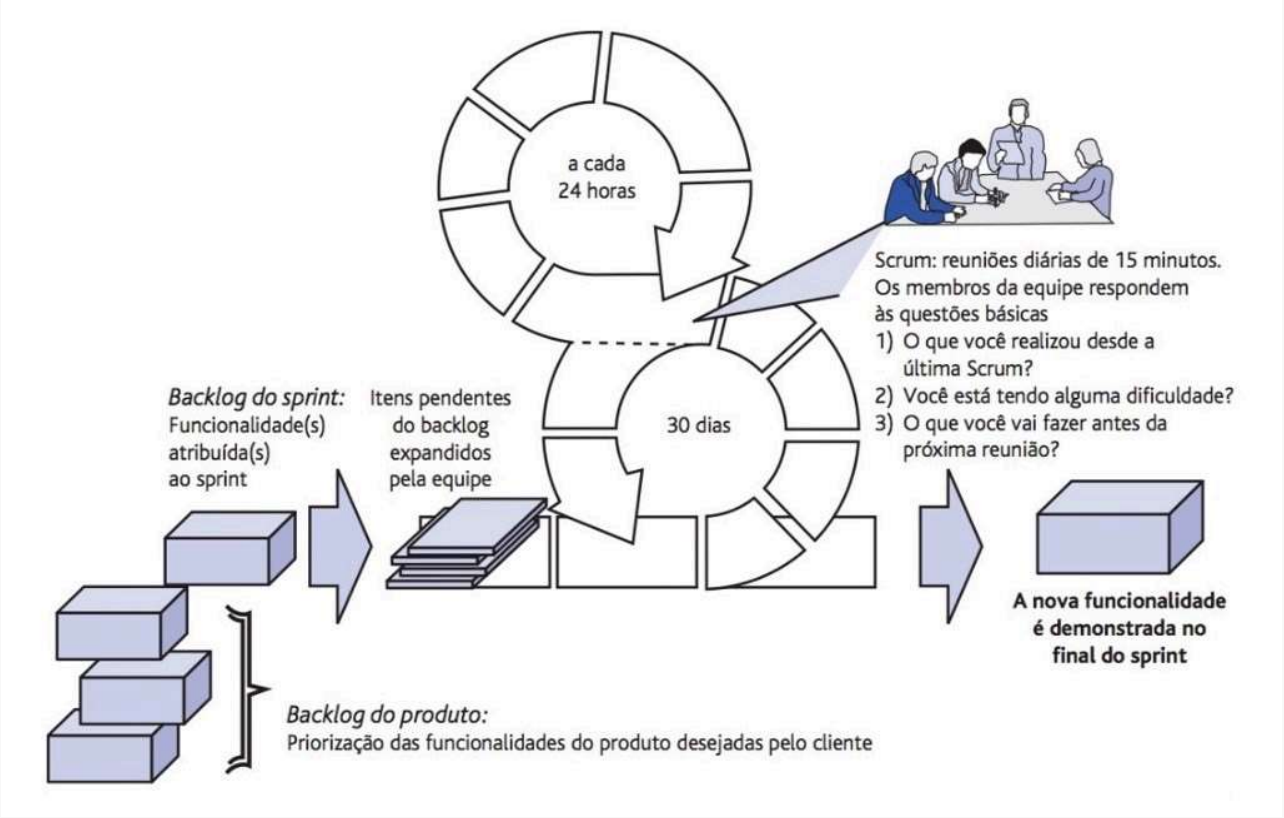
- Proximidade do cliente pode gerar solicitações informais, colocando em risco o planejamento de escopo/prazo e custo.
- Método de levantamento de requisitos superficial, por meio de histórias, o que pode gerar omissões e inconsistências.

SCRUM

O modelo *scrum* foi desenvolvido a partir dos anos 90. Os principais conceitos utilizados neste modelo são: ***backlog, sprints, scrum master*** e ***product owner***.

A Figura 8 apresenta o fluxo geral do modelo scrum, apresentando como estes conceitos estão relacionados.

Figura 8: Modelo *scrum*



Representação das principais atividades e práticas propostas no modelo *scrum*. Fonte: PRESSMAN, R. S. ; MAXIM, B. R. (2016, p. 78).

O ***backlog*** do produto representa uma lista de requisitos priorizados que fornecem valor ao cliente. Novos itens podem ser adicionados a qualquer momento.
As ***sprints*** são compostas de itens para atender aos requisitos existentes no *backlog*, os quais serão implementadas em um ciclo de no máximo 30 dias. As alterações não são introduzidas durante a execução das *sprints*.

As reuniões são diárias e curtas, de aproximadamente 15 min., para responder três perguntas:

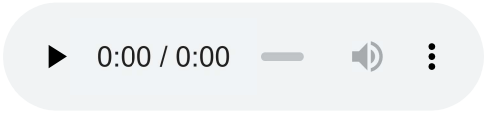
- O que você fez desde a última reunião?
- Quais dificuldades encontrou?
- O que planeja realizar até a próxima reunião?

O líder da equipe que conduz a reunião é chamado ***scrum master***. Outro papel muito importante no método *scrum* é o ***Product Owner*** (PO), que define as histórias (que serão desmembradas em funcionalidades) e prioriza o *backlog* de um produto de *software*.

Ao final da *sprint*, uma nova funcionalidade é entregue. Vários ciclos com este deverão ser executados para atender aos itens que compõem o *backlog* do produto.

PODCAST

Neste *podcast*, é apresentada uma conversa com especialistas da área de engenharia de *software*, os quais contam como a área surgiu e como está sendo a sua evolução ao longo dos últimos anos.



Trilha do *podcast* por ©Jahzzar/freemusicarchive.org

Nesta Unidade, houve uma introdução aos principais conceitos da engenharia de *software* e exposição sobre os modelos de processo de desenvolvimento de *software*. Entendemos que os modelos foram evoluindo ao longo do tempo. Os primeiros modelos, conhecidos como prescritivos, tinham um caráter mais rigoroso na determinação de como as atividades de desenvolvimento deveriam ser executadas. No início dos anos 2000, os modelos ágeis ganharam força: eles vieram para suprir as deficiências dos modelos prescritivos, trazendo maior capacidade para acomodar as mudanças, que são naturais ao longo do desenvolvimento do *software*. É importante entender os motivos que embasaram o surgimento de cada modelo, as situações em que a sua aplicação é mais favorável e os pontos de atenção. Assim, somos capazes de analisar as características do contexto em que estamos inseridos e escolher qual pode nos ajudar a desenvolver *softwares* com mais qualidade. Lembre-se: **processo** e **qualidade** são as palavras-chave da engenharia de *software*.

Referências

BECK, K. *et al.* ***Manifesto for Agile Software Development***. 2001. Disponível em: https://agilemanifesto.org/?utm_source=angelopublio.com.br. Acesso em: 21 jan. 2021.

PRESSMAN, R. S. ; MAXIM, B. R. **Engenharia de *software*:** uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788580555349/. Acesso em: 21 jan. 2021

SBROCCO, J. H. T. C.; MACEDO, P. C. **Metodologias ágeis:** engenharia de *software* sob medida. São Paulo: Editora Saraiva, 2012.

WAZLAWICK, R. S. **Engenharia de *software*: conceitos e práticas.** 2. ed. Rio de Janeiro: Elsevier, 2019.