Índice

[Metodologias 2](#_Toc66986400)

[Escolhas 2](#_Toc66986401)

[Metodologia Ágil 2](#_Toc66986402)

[Metodologia Cascata 3](#_Toc66986403)

[Perfil do cliente – Cascata 3](#_Toc66986404)

[Processo de Software 4](#_Toc66986405)

[Entendo as desvantagens 4](#_Toc66986406)

[Ciclo de Vida (1) 5](#_Toc66986407)

[Ciclo de Vida (2) 6](#_Toc66986408)

[Construção de Software 7](#_Toc66986409)

[Padrão de Projeto 7](#_Toc66986410)

[Padrão de Código 7](#_Toc66986411)

[Componentização (DRY) 7](#_Toc66986412)

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versão | Responsável | Motivo |
|  | 1 | R. Groff | Criação do documento |

# Metodologias

## Escolhas

A metodologia está profundamente conectada com o processo de desenvolvimento de software. Existem diversas opções no mercado e seus impactos nos resultados precisam ser observados para diminuir os efeitos de uma escolha.

São elas:

* Metodologia Ágil;
* Metodologia Cascata;

## Metodologia Ágil

Como o nome informa, é voltada para **velocidade** na entrega de requisitos.

|  |  |
| --- | --- |
| Vantagens | Desvantagens |
| * Foco em errar rápido, consertar rápido e com custo baixo; * Entregas feitas com maior  velocidade, pois dispenda projeto ou padrões; * Cliente poderá participar mais intensamente do projeto, pois a agilidade na entrega requer seu feedback para a próxima entrega; * De custo mais barato, pois não requer diversas etapas (documentação, projeto, testes unitários e de integração) já que tudo vai mudar de acordo com a revisão do cliente numa posterior refatoração; * Perfeito para clientes que não sabem muito bem o que precisam, porém querem ver algum protótipo funcional antes de investirem substancialmente; | * Falta de documentação; * BBM pois reside na refatoração como princípio, e nunca no projeto como um todo – cada pedaço do projeto tende a ter padrão diferente; * Maior quantidade de bugs pela quantidade de refatorações; * Depende da interpretação dos requisitos pelo dev – quanto mais junior, maior será o retrabalho; * Curva alta de aprendizado de um novo integrante pela falta de documentação / padrões; * Clientes não engajados podem ser um risco ao projeto, esperando muito do software e recebendo “pouco”; * Não recomendado para projetos de longa duração; |

## Metodologia Cascata

Esta metodologia de construção é voltada para **segurança** e conferência passo a passo na entrega de requisitos.

|  |  |
| --- | --- |
| Vantagens | Desvantagens |
| * Feito para projetos longos e de continuo atendimento; * Ideal para times grandes e projetos de alta complexidade; * Foco em entregar o que foi pedido sem erros; * Documentação primeiro, sempre em constante melhoria; * Papel de um projetista elimina a interpretação do desenvolvedor; * Todos participam do processo de avaliação do projeto, do tester ao projetista – diminuindo erros óbvios de entendimento dos requisitos; * Curva rápida de aprendizado para novos membros da equipe; * Devs juniors conseguem atender a demanda de entrega de código; | * Entregas requerem um **projeto de software** para construção posterior, e então as sprints são maiores; * Menor foco em programação e sim em aderência ao projeto e a seguir os padrões; * Cliente mais formal e atento pelo seu **alto investimento** – sujeito àprocessos na justiça; |

## Perfil do cliente – Cascata

O projeto atual de software requer longa manutenção e, por lidar com um sistema financeiro, precisa de segurança e previsibilidade. Por estas características, a **metodologia de cascata** é o ideal neste cenário.

# Processo de Software

## Entendo as desvantagens

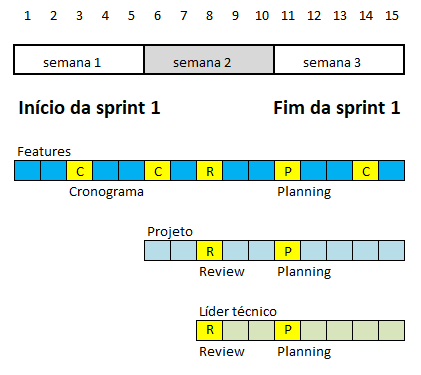
Conhecendo todas as desvantagens do modelo cascata escolhido, precisamos diminuir estas características negativas com algumas ações consistentes nas equipes.

Para isso, veremos todos os itens novamente, em mais detalhes:

* **Projeto de Software.**
  + Apesar deste item estar na coluna de desvantagem , é de sua natureza garantir a segurança do time e de ser **pesado na sua importância**.
  + Um projeto forte e blindado por todos garante a robustez do sistema. Precisa ser objeto de **constante melhoria / revisão**.
  + Todos devem participar da etapa de peer review antes de iniciar o desenvolvimento, evitando erros crassos por falta de atenção e detalhando ao máximo detalhes que a história do usuário possa ter.
* **Pouca programação.** 
  + O uso de documentação primeiro e ajustes de código em segundo lugar pode ser um fator complexo em algumas equipes.
  + Ao mesmo tempo que um desenvolvedor júnior já começa a programar instantaneamente pode levar um desenvolvedor senior desinteressar-se pela falta de desafios.
  + A equipe precisa entender que a **segurança vem em primeiro lugar**, e que a “falta de interpretação” como arte de um programador resulta em um software com menos erros.
  + Foco total em testes de integração;
  + Mesmo assim, vai ter espaço para POCs onde a equipe possa exercer mais liberdade de expressão.
* **Cliente mais atento.** 
  + Se a escolha pela segurança de atendimento aos requisitos via um projeto de software acarreta um custo maior pelo tempo e um profissional mais dedicado à documentação – temos que o **alto investimento** resulta em um cliente mais propenso à judicialização e **cobrança**.
  + Provavelmente terá um time próprio para homologar o software e fornecer seus requisitos.
  + O foco do time precisa manter a qualidade em alta para justificar seus custos: se houver esta reciprocidade, **o projeto terá uma longa duração**.

## Ciclo de Vida (1)

Segue esquematização simplificada da primeira sprint do projeto. Ela é diferente das outras por que não temos nenhuma feature escrita para criarmos as tarefas.



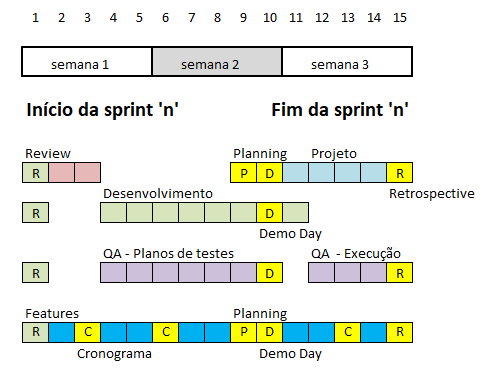
Conforme as **features** forem sendo escritas pelo PO e as partes interessadas (cliente) serem entrevistadas na primeira semana. A partir da segunda semana teremos material para os projetistas começarem o entendimento e a especificação via o evento de **review**.

A gerência de projetos deverá levar em conta sempre este feedback constante do PO e ajustar todo o registro de features preparando as priorizações (na ordem correta), e sua versão atual de **cronograma**.

Após a **review** (revisão geral, com projetista, líder técnico, PO e GP) de entendimento das features, o time vai traçar suas estratégias mais consolidadas para o projeto.

No evento final de **planning,** e já com o devido orçamento junto ao lider técnico, as tarefas deverão ser criadas e os projetos entregues pelos projetistas nos diretórios de **build**, conforme a decisão planejada no **cronograma** pelo gerente de projeto conhecendo as priorizações do cliente.

## Ciclo de Vida (2)



* **Review** – Todos participam e revisam os documentos da build em relação aos documentos das features. Nestes 3 dias, ajustes seram feitos na documentação e perguntas serão feitas ao PO para obter as regras de negócio do cliente.
* **Desenvolvimento** – A equipe de programação traduz os documentos da build em código, efetuando todas as mudanças no software.
* **Demo Day** – Ao restarem dois ou três dias antes do término do prazo de entrega, teremos este evento na equipe completa de demonstração do software antes do envio para QA. Nestes dias, os ultimos ajustes serão feitos e uma integração completa entre os desenvolvedores e lideres técnicos fecharão a entrega para testes.
* **QA** – Após a peer review, começa a criação dos planos de testes. Participa ativamente do demo day; após a entrega em homologação, inicia a execução de testes exploratórios e os planejados.
* **PO** – Durante toda a sprint atual, prepara as features dos próximos builds.
* **Planning** – Reservadamente, esta reunião é feita somente entre o líder técnico, PO e os projetistas. Com ela, a próxima sprint é montada, orçada e ajustada em prioridades e ordem de desenvolvimento (onde houver).
* **Projeto** – Criação dos documentos de build pela equipe de projeto.
* **Retrospective** – No fim da sprint, debatemos o que deu certo e errado e ações a serem feitas, além de revisar o que aconteceu desde a última retro.

# Construção de Software

## Padrão de Projeto

A criação das builds se dá via documentação em linguagem algoritmica de alto nível. Existirá uma tradução para código, feita pelo desenvolvedor – mas estas expressões deverão ser fáceis de entender e 100% localizáv eis nos artefatos de programação.

É preciso conscientização de que tais códigos deverão ser feitos por profissionais junior / pleno, para baixar o custo de toda a operação e maximizar a entrega de software com o máximo de segurança possível.

## Padrão de Código

Durante o tempo de teste / projeto, haverão momentos onde os desenvolvedores terão um tempo disponível para revisão de código / melhorias. Tais melhorias objeto de refatoração precisam estar seguras e dentro de testes unitários e de integração para não haver novos bugs.

Diferentemente de um processo ágil, num projeto de metodologia cascata os requisitos tendem a não mudar, pois chegam revisados e consolidados à equipe de desenvolvimento.

Desta maneira, **os testes de integração são mandatórios**, e 50% do trabalho do desenvolvedor será não quebrar os outros testes já criados enquanto seus próprios testes devem contemplar no mínimo 50% dos desvios do programa; dependendo da criticidade da área, o líder técnico deve pedir a totalidade de cobertura de código.

Cabe ao líder técnico revisar testes dos desenvolvedores e **aderência aos padrões**.

## Componentização (DRY)

Nem todo software pode ser descrito facilmente pelo projetista. Idealmente, abstrações e componentes padrão poderão ser escritos pelo lider técnico e depois referenciados na documentação.

A idéia é abstrair comportamentos complexos e re-utilizáveis dentro de classes de fácil uso pelo time de desenvolvimento.

Ao receberem os requisitos esperados na sprint, este planejamento de POCs (prova de conceito) precisa ser provocado pelos desenvolvedores (ou mesmo pelos projetistas) e executado pelo líder técnico, colocando o componente novo/re-escrito aderente 100% aos padrões de código.