# PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Ana Luiza Cerchiari de Andrade

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

# Métodos ágeis

## Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Descrever o conceito de método ágil.
- Identificar os princípios e valores ágeis.
- Enumerar os principais métodos ágeis existentes.

## Introdução

Atualmente, dois cenários impactam a rotina dos desenvolvedores: o cenário da *big data* e o cenário da Internet das Coisas. Profissionais que estudam *software* e atuam na área de desenvolvimento de *software*, sistemas e *sites* são forçados a implementar tecnologias novas a todo momento. A Internet das Coisas fez com que câmeras, *smartphones* e relógios fiquem conectados à internet a todo momento, e as empresas veem nisso uma oportunidade para estudar comportamentos através dos *logs* e registros das pessoas nos sistemas. Quem quer que trabalhe na área certamente será forçado a adaptar *software* rapidamente às tecnologias que surgem diariamente. Por isso, a rotina dos desenvolvedores é recheada de adrenalina.

Em pouco tempo, vimos os *sites* que possuíam linguagens HTML 4, CSS , PHP 5 e JavaScript se tornarem HTML 5 com suas animações, CSS 3, com suas decorações extravagantes, PHP 7, com sua agilidade e códigos novos, e JavaScript React, Angular e JQuery, trazendo novas formas de controlar e manipular os elementos frontais das páginas. Além disso, você deve ter percebido que os bancos de dados SQL ganharam a companhia dos bancos NoSQL, que chegaram com agilidade para *sites* que contêm muita interação com pessoas. Na criação de *software*, percebemos que o que antes era feito através de Ionic, um *framework* JavaScript que cria aplicativos com *plugins*, hoje é feito através de React Native, que traz experiências mais satisfatórias ao usuário e usa a linguagem nativa dos aparelhos *mobiles*.

Neste cenário, várias formas de planejamento de criação de *software* disputam as decisões das equipes que desenvolvem, e disso surgem dúvidas quanto ao critério de escolha do método de desenvolvimento de *software* mais dinâmico e condizente com o tempo em que vivemos. De acordo com Sommerville (2011), em 1980, com o surgimento das linguagens de quarta geração, e em 1990, com o surgimento de metodologias ágeis, como a Metodologia de Desenvolvimento de Sistemas Dinâmicos e a Metodologia Extreme Programming, os desenvolvedores se viram forçados a mudar a forma de trabalho. Vamos discutir um pouco sobre isso neste capítulo.

# 1 Conceito de método ágil

O desenvolvimento ágil envolve mudanças rápidas e prioriza-se, às vezes, agilidade frente à qualidade — alguns analistas dizem que este método é uma resposta ao cenário tecnológico. Após se fazer um planejamento de etapas de requisitos, é comum ter que se deparar com mudanças dos próprios requisitos, e isso é aceito neste método de desenvolvimento, pois não impossível antever todas as necessidades do mercado, visto que elas mudam com frequentemente; em alguns casos, os requisitos mudam depois da entrega do projeto (SOMMERVILLE, 2011).

De acordo com Amaral *et al*. (2012, p. 21) o conceito de ágil pode ser visto como:

> [...] é uma abordagem fundamentada em um conjunto de princípios, cujo objetivo é tornar o processo de gerenciamento de projetos mais simples, flexível e iterativo, de forma a obter melhores resultados em desempenho (tempo, custo e qualidade), menos esforço em gerenciamento e maiores níveis de inovação e agregação de valor para o cliente.

O desenvolvimento ágil é feito em equipes multidisciplinares, onde os processos são feitos em espiral e as equipes priorizam conversas em detrimento da documentação, ou seja, a rigidez é menor. Aqui, é mais importante a funcionalidade do *software* do que a documentação; é mais importante satisfazer as necessidades do cliente do que seguir um plano. No método ágil, os colaboradores têm perfil motivado, as entregas são semanais e não mensais, o que facilita o acompanhamento da satisfação do cliente. Logo, neste modelo, as adaptações falam mais alto do que a disciplina, e o processo de criação

de *software* é transparente, o cliente vê tudo, o que ajuda na customização, conforme Sommerville (2011). No método ágil, as equipes são pequenas, com alto nível técnico, *feedbacks* constantes, as reuniões são rápidas — duram geralmente 15 minutos — e muitas vezes são feitas em pé, e cada reunião gera um plano de ação, em outras palavras, um conjunto de atividades a serem realizadas em um intervalo de tempo determinado. A maioria dos métodos ágeis foi criada na década de 1990, e eles serão explicados mais adiante.

## Ferramentas eletrônicas de métodos ágeis

Que tal falarmos sobre ferramentas de gestão? O primeiro passo para criar um painel de metodologia ágil é escolher o seu modelo de gestão, estes modelos denominados Scrum, Kanban, entre outros, serão explicados posteriormente, mas precisamos mencionar que as empresas estão usando preferencialmente dois *sites*: **Trello** e **Jira**.

O Trello é uma ferramenta de gestão *on-line* para métodos ágeis que auxilia na organização de tarefas e possui três versões. O Quadro 1 compara as três versões desta ferramenta. A primeira é o plano básico, que é gratuito e, mesmo assim, permite vários usuários, porém não tem todas as ferramentas dos outros planos. A segunda versão do Trello, chamada Business Class, é paga e possui, por exemplo, relação com o Github. Hoje muitas empresas de tecnologia usam programas como o Github a fim de salvar versões de cada alteração do código automaticamente no servidor da plataforma. A terceira versão, o plano Enterprise, também é paga e possui muitas ferramentas de gerenciamento de acessos a pastas e a convites, entre outras funcionalidades administrativas.

**Quadro 1.** Comparação entre os três planos do Trello

| Plano básico | Plano Business Class | Plano Enterprise |
|---|---|---|
| ■ Gratuito<br>■ Permite o acesso de vários usuários<br>■ Sem limite de quadros | ■ Contempla os itens anteriores<br>■ Acesso sincronizado a Jira, Slack, Github<br>■ Suporte em até um dia útil<br>■ Classificação automática de quadros de atividade em coleções | ■ Todas as características anteriores<br>■ Botões, regras e comandos agendados ilimitados<br>■ Notificações por *e-mail* e solicitações externas na *web*<br>■ Controles de cada pessoa pela administração<br>■ Restrições de criação e exclusões de pastas<br>■ Restrições de anexo<br>■ Restrições ao convite do quadro de equipe<br>■ Gerenciamento de usuários em tempo real: desativar em massa; ativar em massa; filtrar por última atividade |

***Fonte:*** Adaptado de Trello (c2020).

O Jira também é uma ferramenta *on-line* de gestão ágil muito usada em métodos ágeis. Esta ferramenta é mais complexa que o Trello e possui maior processamento e velocidade, seu suporte é muito completo, atende 24 horas por dia, e foi desenvolvido para projetos de *software*.

O Quadro 2 mostra as utilidades do Jira e compara três planos: o básico, gratuito, e os demais, Standart e Premium, sendo este último o mais caro.

**Quadro 2.** Comparação entre os três planos do Jira

| Plano básico | Plano Standart | Plano Premium |
|---|---|---|
| ■ Visualização do trabalho da equipe em um painel customizado, seguindo modelos de agilidade como Scrum e Kanban<br>■ Até 10 usuários<br>■ 2 GB de armazenamento de arquivos<br>■ 100 execuções por mês<br>■ Suporte da comunidade<br>■ Lista de pendências<br>■ Relatórios de desempenho de equipes | ■ Todas as características anteriores<br>■ Até 5 mil usuários<br>■ 500 execuções por mês<br>■ Suporte em horário comercial<br>■ Permissões avançadas<br>■ 250 GB de armazenamento de arquivos<br>■ Acesso anônimo<br>■ *Log* de auditoria | ■ Todas as características anteriores<br>■ Até 5 mil usuários<br>■ 1.000 execuções por mês<br>■ Suporte em 24 horas<br>■ GB ilimitado de armazenamento<br>■ Roteiros avançados<br>■ Arquivamento de projetos<br>■ Lista de permissões por usuário |

***Fonte:*** Adaptado de Atlassian (c2020).

## 2 Valores e princípios ágeis

De acordo com o *site* do Manifesto Ágil (BECK *et al.*, c2001), no ano 2001, em Utah, nos EUA, 17 profissionais que praticavam metodologias ágeis se reuniram para praticar Snowbird e conversar sobre métodos de planejamento de *software*. Criaram, portanto, um documento, chamado de grito de guerra, o Manifesto Ágil, que possui quatro valores (contidos na coluna da esquerda na Figura 1). Estes valores priorizam os pontos a seguir.

1. Pessoas frente aos processos — Este tópico objetiva favorecer relacionamentos no ato da construção do *software*.
2. Funcionamento *versus* documentação — Este tópico objetiva favorecer a funcionalidade da criação de *software*, e não somente o *design*.

3. Colaboração do cliente, a funcionalidade da criação de *software* — Este tópico objetiva favorecer a interação contínua com o cliente, a fim de compreender seus desejos e anseios com mais precisão.
4. Consertar problemas e se adaptar a mudanças — Este tópico objetiva favorecer uma construção dinâmica e não engessada, onde consertar problemas é mais importante que manter a burocracia.


**Figura 1.** Valores do manifesto ágil.
*Fonte:* Adaptada de Isotani e Rocha ([20––]).

Perceba o quanto esses valores são aplicáveis. Não se trata de deixar de se organizar ou de documentar, obviamente, mas de tornar o trabalho mais funcional e orgânico. Considere que os profissionais da área de programação ficam sobrecarregados e que a área de tecnologia é estressante; há muita demanda para pouco profissional, e a tendência é aumentar a necessidade de profissionais.

Uma situação que pode acontecer durante o desenvolvimento de um *software*, por exemplo, é, apesar de haver um plano de ação, as pessoas perceberem, em meio ao processo, a necessidade de alterações ou de inclusão de novas funcionalidades. Outro exemplo seria, na criação de um *app*, constatar que os dados precisam ficar em uma tabela temporária, e então ter que parar tudo e criar um banco de dados temporário no painel do servidor.

Agora, imagine que você tem que ficar documentando os detalhes de tudo que fizer. Os programadores ficariam mais sobrecarregados ainda, pois além

de programar eles teriam que documentar. Neste ponto, ferramentas como o Github ajudam, pois salvam automaticamente as versões.

Imagine agora que você não permite que seus funcionários mudem um pouco o curso? Isso acabaria atrapalhando um atributo muito importante: funcionalidade! Por isso devemos seguir, sim, as boas práticas, sem, contudo, negligenciar o tempo de execução, pois cada minuto poderia ser um recurso a mais no seu *software*.

Falamos dos quatro valores do Manifesto Ágil. Vejamos agora seus 12 princípios desenvolvimento de *software*? Eles estão presentes do próprio *site* oficial do Manifesto Ágil (BECK *et al.*, c2001).

- Nossa prioridade é satisfazer o cliente através da entrega contínua e adiantada de *software* com valor agregado.
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
- Entregar frequentemente *software* funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo. Indivíduos e interação entre eles mais que processos e ferramentas.
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
- Construir projetos em torno de indivíduos motivados, dando a eles o ambiente e o suporte necessário e confiando neles para fazer o trabalho.
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é por meio de conversa face a face.
- *Software* funcionando é a medida primária de progresso.
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
- Contínua atenção à excelência técnica e bom *design* aumentam a agilidade.
- Simplicidade: arte de maximizar a quantidade de trabalho não realizado é essencial.
- As melhores arquiteturas, requisitos e *designs* emergem de times auto-organizáveis.
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz, e então refina e ajusta seu comportamento de acordo.

Como você pode perceber, é um método poderoso, para quem já atua com programação ele faz muito sentido. O sistema tradicional se mostra lento: nele, os clientes chamam a empresa de criação de *software*, e para fazer a elicitação são necessárias longas visitas técnicas, análises de documentos, entrevistas e análises de processos, para então dar uma data relativamente demorada com um valor relativamente alto para entregar a aplicação. Resultado? Alto custo e lentidão. No sistema ágil o processo é mais fatiado e incrementado. Falaremos agora sobre algumas metodologias famosas.

## 3 Principais métodos ágeis existentes

Primeiramente, listaremos os métodos ágeis famosos. O primeiro citado é o Scrum (será explicado adiante), no Trello. Na década de 1990 vários modelos influenciaram na criação de métodos ágeis. Podemos citar como exemplo os seguintes modelos: Desenvolvimento Incremental, DSDM (Metodologia de Desenvolvimento de Sistemas Dinâmicos), Crystal Clear, FDD (Desenvolvimento Direcionado a Funcionalidade), XP (Extrem Programming) e Scrum.

O Quadro 3 mostra a recomendação de métodos por fases do projeto de desenvolvimento de *software*.

**Quadro 3.** Áreas de gerenciamento e métodos com mais contribuição

| Área | Métodos com maior contribuição |
|---|---|
| Gerenciamento de requisitos | XP |
| Medição e análise | XP, Scrum |
| Planejamento de projeto | XP, Scrum |
| Monitoramento e controle | XP, Scrum |
| Gerenciamento do subcontratado | Não se aplica |
| Gerenciamento de qualidade de produto e processo | FDD |
| Gerenciamento de configuração | Nenhum |

***Fonte:*** Adaptado de Alegría *et al.* (2011).

O primeiro, **desenvolvimento incremental**, criado pela IBM em 1990, faz a construção do sistema de pedaço em pedaço, ou de incremento em incremento. Geralmente, um pedaço é feito, e após o cliente dá alguns *feedbacks*. Cada pedaço é uma parte inteira: por exemplo, uma página de *login link* com acesso a banco de dados é um incremento (SOMMERVILLE, 2011).

O segundo método cabível de explicação é a **Metodologia de Desenvolvimento de Sistemas Dinâmicos**, o qual é usado em situações onde o tempo e a verba são limitados. De acordo com Cruz (2018, p. 316) "[...] tem um conceito mais próximo a um *framework* do que um método propriamente dito, sua ênfase foca a criação de protótipos que posteriormente evoluem para o sistema, e para isso é utilizada muito fortemente a colaboração próxima com o cliente".

Este método buscar entregar 80% do projeto em 20% do tempo disponível. Ele é composto por três fases: a fase do pré-projeto (onde se elabora o orçamento), a fase do ciclo de vida (onde se analisa a viabilidade) e a fase do pós-projeto (onde ocorrem as alterações), conforme a Figura 2.

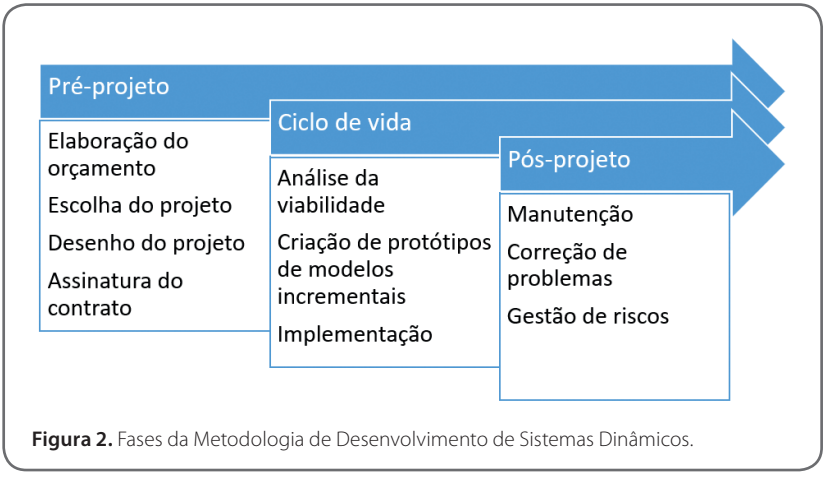


**Figura 2.** Fases da Metodologia de Desenvolvimento de Sistemas Dinâmicos.

A **Metodologia de Desenvolvimento de Sistemas Dinâmicos** é recomendada quando os projetos são urgentes, quando os projetos são formados por uma nova tecnologia e precisam ser acompanhados de testes do usuário, ou quando remetem a um lançamento de produto com data marcada. Qual é a diferença entre este método e outros? Em metodologias tradicionais, como por exemplo, o famoso *Project Management Institute* (PMI), o cronograma e o orçamento são abertos, e o escopo é fechado; já a **Metodologia de Desenvolvimento de Sistemas Dinâmicos** tem cronograma e orçamento fechados, mas o roteiro é aberto (SOMMERVILE, 2011).

O **Crystal Clear** (ISOTANI; ROCHA, [20––]) é outra metodologia ágil. Ela atende vários tipos de projetos e tem como valor a comunicação com o cliente, bem como o relacionamento do desenvolvedor com o cliente, a fim de que possa captar com facilidade suas expectativas. Este método evita a criação de ferramentas complexas que não serão utilizadas, objetivando reduzir tempo e custo na entrega. Este método busca diferenciar a metodologia específica conforme a natureza de cada projeto e permite que os desenvolvedores se manifestem quando algo os incomodar. O método Crystal é um cristal em figura elaborado pela gestão e mostrado com uma escala de cores, onde as cores variam de acordo com nível de criticidade e com o tamanho da equipe: quanto mais escuro o cristal, mais crítico de fazer é o *software* ou projeto; já as cores claras (branco e amarelo) atestam que os *software* ou projetos serão simples e poucas pessoas serão necessárias; projetos alaranjados ou até vermelhos demandam mais pessoas e são mais arriscados. No Crystal da Figura 3, desenhado pela gestão, C significa confortável e D significa baixo custo, E significa alto custo, e L significa risco de vida. O números indicam o número de funcionários.

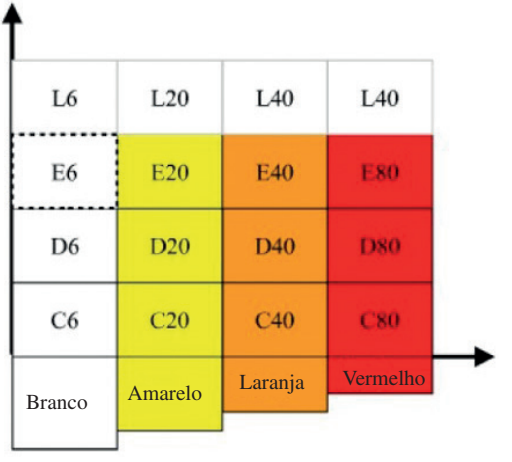


**Figura 3.** Método Crystal.
*Fonte:* Adaptada de Abrahamsson *et al.* (2002).

A metodologia **Feature-driven development** (FDD), ou "Desenvolvimento Dirigido por Funcionalidades", em português, fragmenta os projetos de gestão e de *software* em *features* (funcionalidades). Ela foi concebida na década de 1990, e tem como características os pontos, de acordo com Sommerville (2011):

- elaboração de lista de tarefas de funcionalidades, ou seja, cada passo tem foco em alguma funcionalidade do *software*;
- planejamento voltado ao método incremental por funcionalidade, ou seja, planeja-se etapas de acordo cada parte. Por exemplo: o primeiro item a ser feito é a área do cliente completa; o segundo item será o carrinho de compras completo; o terceiro item será o catálogo completo, e assim por diante;
- *design* voltado à funcionalidade, ou seja, criar um *design* que simplifica a navegação e promove a experiência do usuário;
- teste de *software* orientado à funcionalidade, ou seja, testar cada função em detrimento de testar uma interface, por exemplo.

Pode-se perceber que neste projeto a soma de cada etapa se faz maior do que o todo; assim, recomenda-se dividir em *features* curtas, a fim de agilizar a conclusão de cada função, aumentando a eficiência da construção do programa. Por exemplo: cria-se uma função de *upload* de imagens, cria-se uma função de adicionar produto, e assim sucessivamente.

De acordo com Sommervile (2011), o quarto método a ser citado é o XP, mais conhecido como **Extremming Programing**. Criado em 1996, possui como princípios básicos: trabalho de qualidade, mudanças incrementais, *feedback* rápido e abraçar mudanças. A metodologia possui algumas práticas:

- Jogos de planejamento: no início da semana os clientes e *developers* (desenvolvedores) se reúnem parar priorizar funcionalidades que serão entregues no final da semana. Cada versão deve ser pequena.
- Propriedade coletiva: qualquer pessoa do time pode usar o código do programa sem precisar de permissão para alterar, assim todos têm a oportunidade de conhecer o código inteiro e se familiarizar com ele. Isso é muito importante para agilizar manutenções.
- Teste de usuário: em equipes pequenas são realizados testes do *software* por clientes e desenvolvedores da empresa para avaliar sua qualidade.
- Ritmo sustentável: as equipes devem trabalhar 40 horas, divididas em 8 horas por dia, sem sobrecarga.
- Equipe inteira: as reuniões devem ser em pé e devem ser rápidas.
- Programação em par: um desenvolvedor mais experiente fica com um novato, o novato codifica e o mais experiente programa. O benefício deste método é que ele é revisado por duas pessoas.
- Padronizações de código: a equipe de *dev* (*developers*) determina regras de codificação de salvamento, bem como as boas práticas que devem

ser seguidas. Assim, parecerá que foi a mesma pessoa que editou o código, ele ficará com uma "cara" única.
- Desenvolvimento orientado a teste: elaborar códigos de forma que sejam capazes de ser testados por *software* de teste como Imeter (um *software* que mede desempenho), Selenium (um *software* que mede erros de *sites*), etc.
- Refatoração: é o processo de otimizar a estrutura do código sem alterar o seu comportamento externo para o usuário final.
- Integração contínua: integrar alterações de forma contínua, sem esperar uma semana, por exemplo. Permite saber a real situação do *software* da programação.
- Entregas curtas: entregar pequenos pedaços para o cliente corrigir e avaliar, aumentando a probabilidade de acertar o todo.
- Metáfora: entender a linguagem e as expressões do cliente.
- *Design* simples: o código deve ter exatamente o que o cliente pediu.

A quinta metodologia a ser citada é a **Scrum**. Ela também é citada por Sommervile (2011). Não citaremos todas que existem, obviamente. Vamos abordar alguns pontos sobre o Scrum.

- Três pessoas principais devem ser citadas:
  - o Product Owner, que define o que comporá o Product Backlog (lista de ações do Sprint) e prioriza isso nas Sprint Planning Meetings (reuniões de planejamento do Sprint);
  - o Scrum Master (geralmente um gerente ou líder técnico), que verifica se todos seguem as regras e também busca impedir trabalhos excessivos;
  - o Scrum Team é a equipe de desenvolvimento.
- No Scrum os projetos são divididos em etapas geralmente mensais. Pode-se dizer que são ciclos mensais. Essas etapas chamam-se **Sprints**. Cada Sprint é um Time Box, uma caixa no tempo com um conjunto de metas.
- No Scrum existem reuniões diárias, chamadas Daily Scrum. Elas são feitas no início do expediente e revisam os itens do dia anterior e determinam o que será feito no dia.
- As funcionalidades são agrupadas em uma lista. Product Backlog é esta lista; ela contém todas as funcionalidades necessárias para um produto, e é feita pelo Product Owner.

- O Sprint Planning Meeting é uma reunião feita no início de cada Sprint. Nesta reunião estarão presentes o Product Owner, o Scrum Master, o Scrum Team e interessados. Nesta reunião o Product Owner determina as prioridades, e todos juntos definirão um objetivo para aquele Sprint.
- Este objetivo Sprint será revisado na Sprint Review Meeting, uma reunião com o Product Owner, o Scrum Team, o Scrum Master, e algumas vezes com gerência e clientes, a fim de revisar o que foi atingido e o que não foi.
- O Release Burndown Chart é uma análise de metas atingidas no final de cada Sprint (iteração).

## Criar um sistema básico Scrum na prática

Vamos agora criar um projeto Scrum no Trello. Neste Scrum usaremos a Backlog (a lista priorizada com todas as atividades que se desenvolvem para ter o projeto completo), o Sprint e as reuniões diárias. Vamos à prática? Abra o Trello, conforme a Figura 4, e clique em "Quadros" e crie um quadro chamado "Gerência de Softwares".

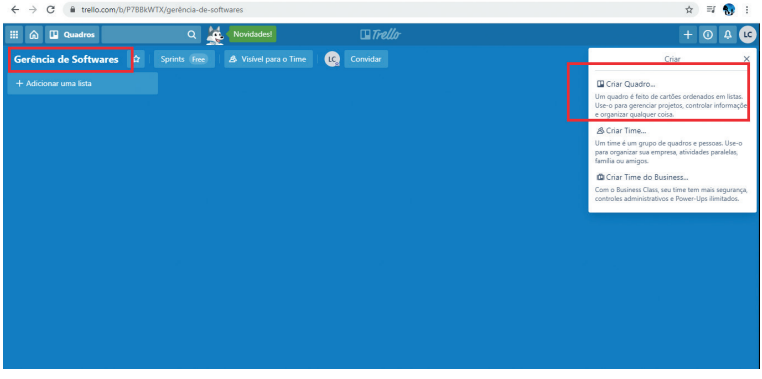


**Figura 4.** Etapa 1: criação de Scrum básico, criando quadros.

Agora crie três listas, conforme Martin (2011), conforme a metodologia Kanban (metodologia destinada a dividir o que foi feito, o que há a fazer e o que estão fazendo em um painel): Feito, A fazer e Concluído, e acrescente mais uma chamada Backlog, e lembre-se: esta Backlog pertence ao Sprint 1. Veja a Figura 5.

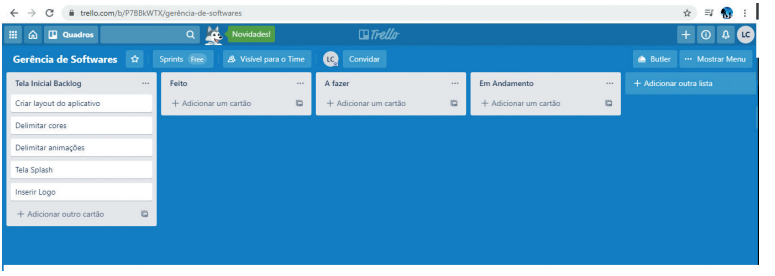


**Figura 5.** Etapa 2: criação de Scrum básico, criando Backlog.

A Figura 6 a seguir mostra os dados de cada tarefa como se fosse um local para colocar os detalhes, como etiquetas, membros que participarão, prazo, imagens etc. Agora abra cada tarefa, adicione os dados da Figura 6, que são: Membros, prazo, *checklist*.

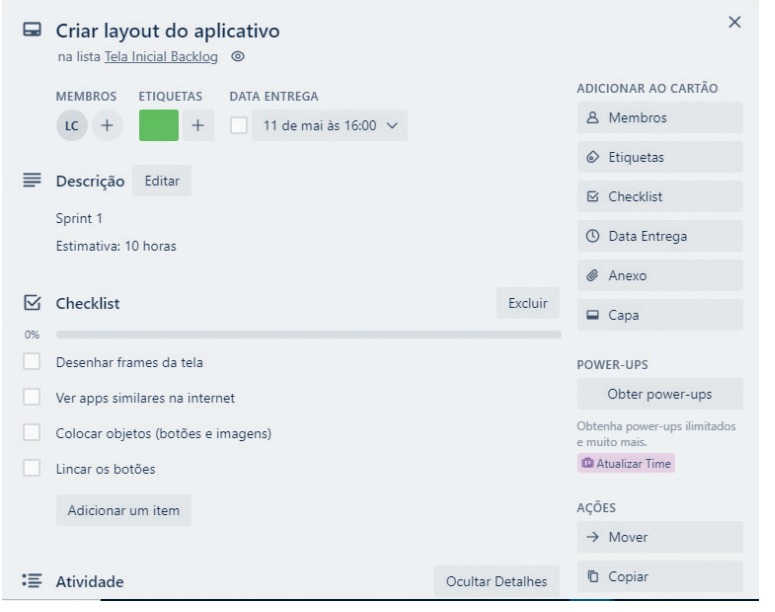


**Figura 6.** Configurando etapas Scrum.

No final da execução, vale a pena anotar, em comentários, qual foi a duração do trabalho, que pode ser 3, 4, 5 ou qualquer outra quantidade de horas — isso irá para o Realease Burndown, com a finalidade de analisar *perfomance*. Neste Backlog arrastamos os Sprints para suas devidas caixas e incluímos em *frames*. Veja a Figura 7.

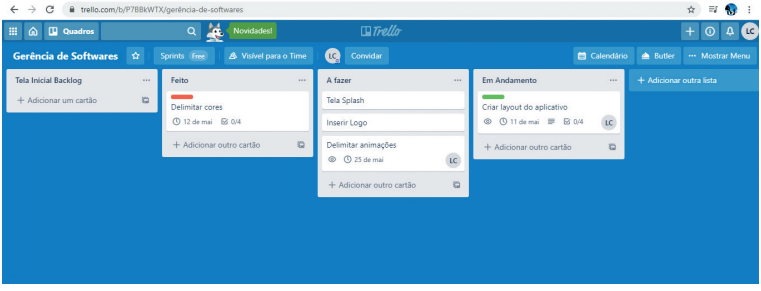


**Figura 7.** Etapa 3: criação de Scrum básico, criando Backlog.

Crie agora mais dois Sprints: um chamado "reunião", e convide os membros do time, e outro "Sprint", para o Burndown. Cabe citar que é ideal que o Burndown tenha gráficos de análise de desempenho. O gráfico a seguir já tem um gráfico demonstrativo, mas normalmente ele é feito no final do Sprint. Veja a Figura 8.

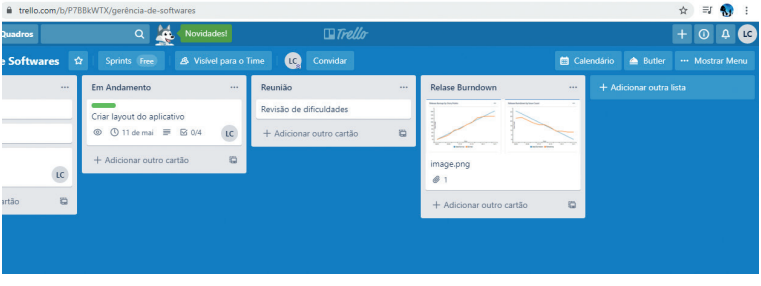


**Figura 8.** Inclusão de reunião e de gráfico Release Burndown.

## Referências

ABRAHAMSSON, P. *et al. Agile software development methods*: review and analysis. Espoo: VTT, 2002.

ALEGRÍA, J. H. A. *et al.* An MDE approach to software process tailoring. *In*: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEMS PROCESS, 2011, Honolulu. *Proceedings* [. . .]. New York: ACM, 2011. p. 43–52.

AMARAL, D. C. *et al. Gerenciamento ágil de projetos*: aplicação em produtos inovadores. São Paulo: Saraiva, 2012.

ATLASSIAN. *Jira sotware*: planos e preços. c2020. Disponível em: https://www.atlassian.com/br/software/jira/pricing. Acesso em: 15 jun. 2020.

BECK, K. *et al. Manifesto para desenvolvimento Ágil de software.* c2001. disponível em: https://agilemanifesto.org/iso/ptbr/manifesto.html. Acesso em: 15 jun. 2020.

CRUZ, F. *Scrum e Agile em projetos*: guia completo. 2. ed. Rio de Janeiro: Brasport, 2018.

ISOTANI, S.; ROCHA, R. V. *Desenvolvimento Ágil.* [20––]. Disponível em: https://edisciplinas.usp.br/pluginfile.php/3128670/mod_resource/content/1/Aula04_Desenvolvimento_agil_Rafaela.pdf. Acesso em: 15 jun. 2020.

MARTIN, R. C. *The clean coder*: a code of conduct for professional programmers. Upper Saddle River: Prentice Hall, 2011.

SOMMERVILLE, I. *Engenharia de software.* 9. ed. São Paulo: Pearson Prentice-Hall, 2011.

TRELLO. *O Trello do seu jeito.* c2020. Disponível em: https://trello.com/pricing. Acesso em: 15 jun. 2020.

## Leituras recomendadas

CAMPOS NETTO, C. *Autodesk Revit Architecture 2018*: conceitos e aplicações. São Paulo: Érica, 2018.

LACERDA, G. S.; BARBOSA, A. B.; RIBEIRO, V. G. Adoption of CMMI and agile methodologies in brazilian companies. *Revista Avances en Sistemas e Informática*, v. 8, n. 3, p. 33–42, 2011.

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

## Fique atento

Os *links* para *sites* da *web* fornecidos neste livro foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS