2
Processos Ágeis

Processos ágeis, também conhecidos como métodos ágeis, referem-se a um grupo de processos de desenvolvimento de software baseados em desenvolvimento iterativo, onde os requisitos e as soluções para os problemas evoluem através da colaboração das equipes.

Métodos ágeis, geralmente, promovem um processo gerencial que encoraja:

- Freqüente inspeção e adaptação do que foi implementado às necessidades observadas no momento;
- Trabalho colaborativo em equipes auto-gerenciáveis;
- Um conjunto de boas práticas de engenharia de software, permitindo entregas freqüentes com alto grau de qualidade;
- Um conjunto de ferramentas de apoio ao processo e ao desenvolvimento, que provêem uma produtividade mais elevada;
- Uma abordagem de negócio que alinha o desenvolvimento com as necessidades do cliente e com os interesses da organização;

Existem várias abordagens para processos ágeis. Esta seção apresenta duas das principais: Scrum e XP. Neste trabalho, as regras de negócio e práticas dos dois processos são base de inspiração para a criação das regras deste projeto (seção 6.3).

2.1.
Scrum

Scrum é um framework para o gerenciamento de projetos e sua função primária é ser utilizado para o gerenciamento de projetos de desenvolvimento de software. Ele tem sido usado com sucesso para isso, assim como Extreme Programming (apresentado na seção 2.2) e outros modelos de processo de desenvolvimento. Porém, teoricamente pode ser aplicado em qualquer contexto no qual um grupo de pessoas necessite trabalhar junto para atingir um objetivo

comum, tais como iniciar uma escola pequena, projetos de pesquisa científica, ou até mesmo o planejamento de um casamento.

As práticas do Scrum baseiam-se em um esqueleto incremental e iterativo (Ilustração 1). O esqueleto funciona da seguinte maneira: no inicio da iteração, a equipe revisa o que deve ser feito. Depois, seleciona o que acredita que pode compor um incremento no final da iteração. No fim da iteração o incremento é apresentado, permitindo sua validação junto aos *stakeholders* [1] do produto.

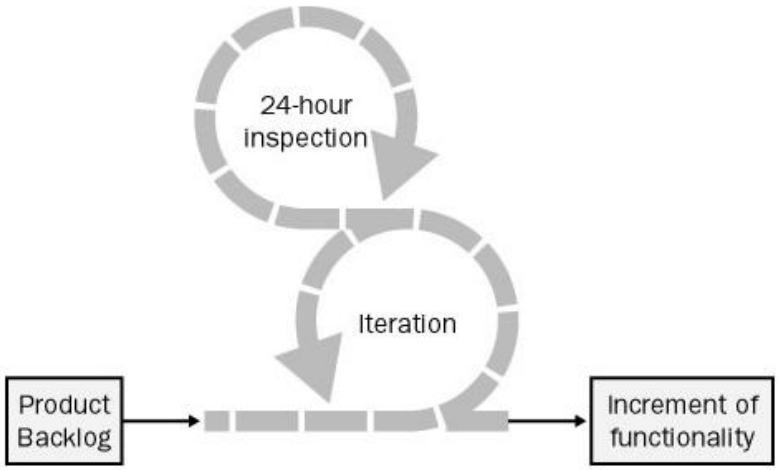

Ilustração 1 - Esqueleto do Scrum (Schwaber, 2004)

Na ilustração acima, o círculo inferior representa a iteração das atividades de desenvolvimento. A saída de cada operação é um incremento do produto. O círculo superior representa o trabalho realizado durante um dia e avaliado na reunião no início do dia seguinte, onde os membros da equipe se encontram para disseminar entre eles o que foi realizado no dia anterior e definir o que será feito no dia corrente.

1 Qualquer envolvido diretamente no uso ou na confecção do produto.

2.1.1.
Processo Scrum Detalhado

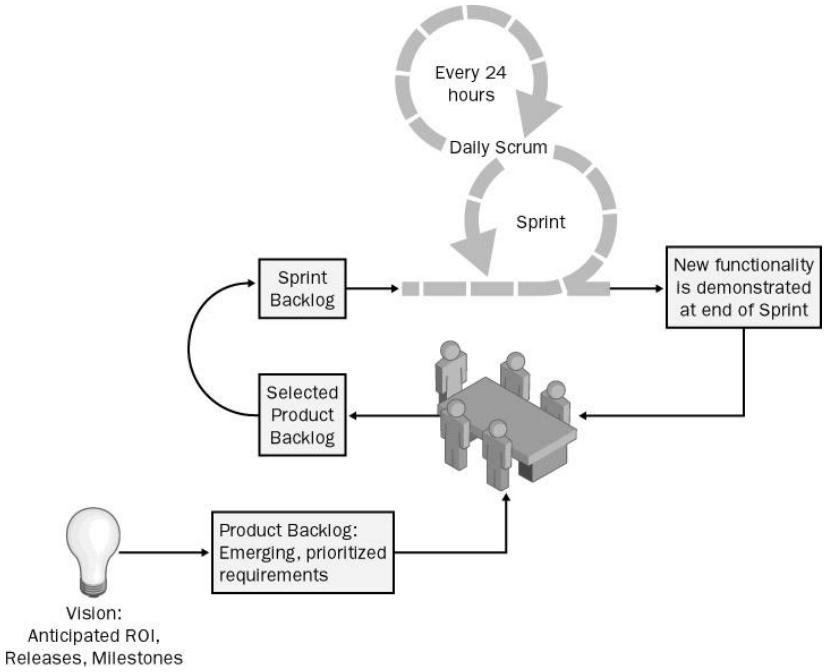

Ilustração 2 - Processo Scrum Detalhado (Schwaber, 2004)

Como apresentado na Ilustração 2, um projeto Scrum começa com a visão do sistema a desenvolver. O *Product Owner* é responsável por especificar os objetivos e requisitos do projeto e por prover a visão em uma maneira que maximize seu retorno de investimento (ROI - *Return of Investments*). O *Product Owner* formula um plano do projeto que inclua o *backlog* do produto. O *backlog* do produto é uma lista de requisitos funcionais e não funcionais (incluindo solicitações de alterações, correções, funcionalidades novas etc.) que, quando transformados em funcionalidades, irão prover a implementação inicial do sistema. O *backlog* do produto representa os requisitos do sistema ainda a serem desenvolvidos ou aprimorados e é priorizado de maneira que, os itens que irão gerar mais valor possuam maior prioridade e sejam divididos em entregas diferentes. A priorização dos itens é um passo inicial, mas sua atualização é realizada ao longo da execução do projeto. (Schwaber, 2004)

Todo o trabalho é feito em *sprints*. Cada *sprint* é uma iteração de 30 dias consecutivos, iniciado com uma reunião de planejamento, onde o *Product Owner* e a equipe decidem juntos o que será realizado na próxima iteração. Selecionando os itens mais prioritários do *Backlog*, o *Product Owner* passa para a equipe o que deseja que seja desenvolvido na iteração e a equipe por sua vez

diz para o *Product Owner* o quanto eles acreditam que poderá ser realizado na iteração. As reuniões de planejamento da iteração não devem durar mais do que 8 horas. (Schwaber, 2004)

A reunião de planejamento possui duas partes. Nas primeiras 4 horas, o *Product Owner* apresenta os itens de maior prioridade que deseja que sejam realizados pela equipe, enquanto esta tira o máximo de dúvidas sobre os itens apresentados. Quando a equipe já sabe o suficiente sobre os itens (ainda na primeira parte da reunião), ela seleciona os itens que acredita que podem fazer parte do *Sprint* dentro do limite de tempo estabelecido, e que comporão um incremento funcional do sistema. Durante a segunda parte da reunião, a equipe planeja o *Sprint*. Como a equipe é responsável por gerenciar seu próprio trabalho, ela precisa de um plano para começar o *Sprint*. As tarefas que irão compor este plano são organizadas em um *Backlog* do *Sprint* (o *Backlog* do *Sprint* não é o mesmo que o *Backlog* do projeto. Ambos são mais detalhados na seção 2.1.2). As tarefas evoluem conforme o *Sprint* evolui. (Schwaber, 2004)

Todo início de dia a equipe se reúne para uma reunião de 15 minutos. Nesta reunião, cada membro da equipe responde a três perguntas: O que você fez ontem? O que fará hoje? Existe algum impedimento para realizar seu trabalho hoje? O objetivo desta reunião é sincronizar o trabalho de todos os membros da equipe, marcar alguma reunião necessária para a evolução da iteração e identificar possíveis fontes de problemas que afetem a continuidade do trabalho. (Schwaber, 2004)

Ao término de cada *Sprint* são realizadas a revisão do *sprint* e uma reunião de retrospectiva.

A revisão do *sprint* deve possuir no máximo 4 horas. O objetivo da revisão é apresentação das funcionalidades terminadas ao *Product Owner* e aos *stakeholders*. A revisão do *sprint* começa com a equipe apresentando o objetivo do Sprint, Os itens do *backlog* planejados para o *sprint* e quais foram terminados. Diferentes membros da equipe podem debater sobre o que ocorreu bem ou mal no *sprint*.

A maior parte da revisão do *sprint* é gasta com os membros da equipe apresentando funcionalidades, respondendo perguntas dos *stakeholders* e verificando mudanças desejadas. No fim da apresentação, os *stakeholders* são questionados, um por um, sobre suas impressões, mudanças desejadas e suas respectivas prioridades. O *Product Owner* define com os *stakeholders* e a equipe mudanças potenciais no *Backlog* do produto baseadas no *feedback* da

apresentação. No final da revisão do *sprint*, o *Scrum Master* apresenta a data e local da revisão do próximo *sprint* para o *Product Owner* e os *stakeholders*.

A reunião de retrospectiva deve possuir no máximo 3 horas. Os únicos participantes são a equipe técnica, o *Scrum Master* e o *Product Owner* (este é opcional).

A reunião começa com os membros da equipe respondendo duas perguntas:

- O que foi bem no ultimo *sprint*?
- O que pode melhorar no próximo *sprint*?

O Scrum *Master* anota as respostas em um formulário e a equipe debate sobre as melhorias potenciais para o próximo *sprint*. O *Scrum Master* não está na reunião para prover respostas, mas para auxiliar a equipe a encontrar meios para melhorar o processo *Scrum*. O objetivo final da reunião é revisar o processo *Scrum* para torná-lo mais eficiente e agradável na próxima iteração.

Após a reunião de retrospectiva, uma nova reunião de planejamento é marcada, iniciando um novo *sprint.*

Juntas, a reunião de planejamento, a reunião diária, a revisão do *Sprint* e a retrospectiva do Sprint constituem as práticas empíricas de inspeção e adaptação do Scrum. (Schwaber, 2004)

2.1.2.
Os Artefatos do Scrum

Um artefato é o resultado de uma ou mais atividades dentro do contexto do desenvolvimento de um software ou sistema e possuem algum critério de aceitação definido no processo. Artefatos que são entregue ao cliente (ou usuário) são conhecidos como produtos. Abaixo são apresentados os artefatos gerados ao longo da execução de um projeto utilizando a metodologia *Scrum*.

2.1.2.1.
Backlog do Produto

O *backlog* do produto é a lista formada por requisitos a serem desenvolvidos, solicitações de alterações, correções, novas funcionalidades, etc. O *Product Owner* é responsável por prover seu conteúdo, e a priorização dos itens registrados. O *backlog* do produto nunca está completo, e é simplesmente

uma estimativa inicial (e contínua) dos requisitos. Conforme o produto e seu ambiente de desenvolvimento evoluem, o *backlog* do produto também evolui a partir de atualizações realizadas pelo *Product Owner*. (Schwaber, 2004)

2.1.2.2. Backlog do Sprint

O Backlog do Sprint define o trabalho (as atividades) que a equipe seleciona do *backlog* do produto e que corresponde a um incremento do sistema sendo desenvolvido ou mantido.

A equipe técnica e o *Scrum Master* definem uma versão inicial da lista dessas atividades na primeira parte da reunião de planejamento. É nela que é definido o que será desenvolvido e isso em concordância com as necessidades do *Product Owner*. Ou seja, a versão inicial do backlog do sprint é criada a partir do backlog do produto. No final da segunda parte da reunião, com o planejamento refinado, eles possuem uma compilação final desta lista. As atividades devem ser dividas de maneira que levem entre 4 a 16 horas para serem finalizadas. Atividades mais longas que 16 horas, devem ser revistas e subdivididas em atividades menores. Somente a equipe pode alterar o *backlog* do *sprint*. (Schwaber, 2004)

Task Description	Originator	Responsible	Status (Not Started/ In Progress/ Completed)	Hours of work remaining unt 1	2	3	4	5	6	7	8	9	10	11	12
Meet to discuss the goals and features for Sprint 3-6	Danielle	Danielle/ Sue	Completed	20	0	0	0	0	0	0	0	0	0	0	0
Move Calculations out of Crystal Reports	Jim	Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Get KEG Data		Tom	Completed	12	0	0	0	0	0	0	0	0	0	0	0
Analyse KEG Data - Title		George	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Analyse KEG Data - Parcel		Tim	Completed	12	12	12	12	12	4	4	4	0	0	0	0
Analyse KEG Data - Encumbrance		Josh	In Progress							12	10	10	10	10	10
Analyse KEG Data - Contact		Danielle	In Progress	24	24	24	24	12	10	8	6	6	6	6	6
Analyse KEG Data - Facilities		Allen	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Define & build Database		Barry/ Dave	In Progress	80	80	80	80	80	80	60	60	60	60	60	60
Validate the size of the KEG database		Tim	Not Started												
Look at KEG Data on the G:\		Dave	In Progress	3	3	3	3	3	3	3	3	3	3	3	3
Confirm agreement with KEG		Sue	Not Started												
Confirm KEG Staff Availablity		Tom	Not Started	1	1	1	1	1	1	1	1	1	1	1	1
Switch JDK to 1.3.1. Run all tests.		Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Store PDF files in a structure		Jacquie	Completed	8	0	0	0	0	0	0	0	0	0	0	0
TopLink. Cannot get rid of netscape parser		Richard	Comleted	4	0	0	0	0	0	0	0	0	0	0	0
Buld test data repository		Barry	In Progress	10	10	10	10	10	10	10	10	8	8	8	8
Move application and database to Qual (incl Crystal)		Richard	Completed	4	4	4	4	4	4	4	0	0	0	0	0
Set up Crystal environment		Josh	Completed	2	2	2	2	1	1	1	0	0	0	0	0
Test App in Qual		Sue	In Progress												20
Defining sprint goal required for solution in 2002		Lynne	In Progress	40	40	40	40	40	40	40	38	38	38	38	38
Reference tables for import process		Josh	In Progress												
Build standard import exception process		Josh	In Progress									12	12	12	10
Handle multiple file imports on same page		Jacquie	Disregarded												
Migrate CruiseControl Servlet to iWS 6.0 (landcc_7101) server		Allen	Not Started	4	4	4	4	4	4	4	4	4	4	4	4
Create web server for Qual on PF1D8		Allen	Completed	1	0	0	0	0	0	0	0	0	0	0	0
LTCS Disk		Danielle/ George	In Progress	12	12	12	12	8	8	8	8	8	8	8	8
Follow thru with questions about KEG data to Sue/Tom, re: Keg, LTO	Jacquie	Danielle	Completed	10	10	10	10	10	8	8	0	0	0	0	0
Map KEG data to Active Tables - see also #14	Jacquie	Jacquie/ Allen	In Progress	50	50	50	50	50	50	50	50	50	50	50	50
Preparer SQL to import from KEG tables to Active Tables	Jacquie	George	In Progress	25	25	25	25	25	25	25	25	25	25	25	25

Ilustração 3 - Backlog do Sprint (Schwaber, 2004)

2.1.2.3.
Incremento do Produto

O Scrum requer uma equipe para construir funcionalidades novas do produto a cada iteração. Este incremento deve ser potencialmente funcional, uma vez que o *Product Owner* pode optar por colocá-lo em produção imediatamente. O incremento precisa estar largamente testado, bem estruturado, e com o código bem escrito para que seja gerado um sistema executável, pronto para o uso e com a documentação adequada. Esta é a definição de "pronto" do incremento. (Schwaber, 2004)

2.2.
Extreme Programming

Extreme Programming[2], programação extrema, ou XP, é um processo de desenvolvimento de software voltado para:

- Projetos cujos requisitos são vagos e mudam com freqüência;
- Desenvolvimento de sistemas orientados a objetos;
- Equipes pequenas;
- Desenvolvimento incremental;

Sendo um processo ágil, o XP utiliza como premissa o aprendizado do cliente ao longo de seu desenvolvimento. Este aprendizado permite que a equipe de desenvolvimento direcione constantemente o desenvolvimento do software para produzir a cada momento o que agrega mais valor para o negócio

O XP busca assegurar que o cliente receba o máximo de *feedback* no final de cada dia de trabalho. Ele é organizado em torno de um conjunto de práticas. As práticas do processo são apresentadas na seção seguinte.

2.2.1.
Práticas

O XP se baseia em treze práticas, apresentadas nas seções seguintes.

2.2.1.1.
Cliente presente

O XP trabalha com a premissa de que o cliente deve conduzir o desenvolvimento a partir do *feedback* que recebe do sistema. É importante que o cliente participe ativamente do projeto, para obter ou prover o máximo de *feedback*, facilitando o andamento e a comunicação do projeto.

2 http://www.extremeprogramming.org/

2.2.1.2.
Jogo do Planejamento

O XP utiliza diversas formas de planejamento para garantir que a equipe esteja sempre trabalhando em funcionalidades que agregam alto valor ao projeto. Por isso o projeto é dividido em *releases* e iterações, permitindo que o cliente e a equipe tenham várias oportunidades de revisar o planejamento.

No início de cada *release* e de cada iteração, acontece o Jogo do Planejamento. Trata-se de uma reunião em que o cliente avalia as funcionalidades que devem ser implementadas e prioriza aquelas que farão parte do próximo *release* ou da próxima iteração (Telles, 2004).

No XP, as funcionalidades são descritas em pequenos cartões e são chamados de estórias. Durante o jogo do planejamento, as estórias são estimadas e priorizadas de acordo com a necessidade do cliente.

2.2.1.3.
Stand-up Meeting

Reunião diária realizada pela equipe para avaliar o trabalho realizado no dia anterior e planejar o trabalho do dia. A reunião é realizada com todos os membros da equipe em pé, para mantê-la rápida e objetiva.

Similarmente à reunião diária do *Scrum*, o objetivo do *stand-up meeting* é sincronizar o trabalho de todos os membros da equipe. Esta reunião é a oportunidade que os membros da equipe têm de apresentar os desafios encontrados e quais soluções foram aplicadas, disseminando o conhecimento na equipe e diminuindo o número de problemas repetidos no projeto.

2.2.1.4.
Programação em pares

No XP, os desenvolvedores programam as funcionalidades em pares, ou seja, dois desenvolvedores no mesmo computador. Esta prática permite que o código seja revisado constantemente. Enquanto um desenvolvedor programa, o outro instantaneamente já revisa o código criado, melhorando sua qualidade.

2.2.1.5.
Desenvolvimento guiado por testes

Todo o dinamismo do projeto e mudanças constantes no software exige um controle constante sobre a validade de suas funcionalidades. Daí a importância dos testes automatizados, que aumentam a cobertura de testes sobre o código e diminuem possibilidade de uma mudança no código "quebrar" uma funcionalidade despercebidamente.

Um mecanismo de validação usado pelo XP é o desenvolvimento guiado por testes. Os desenvolvedores escrevem testes para cada funcionalidade antes de codificá-las. Dessa maneira, eles aprofundam seu conhecimento sobre a funcionalidade, se preocupam com interfaces externas dos métodos e classes antes de codificá-los, e criam uma massa de testes, podendo ser usadas constantemente para validar o sistema.

Existem vários frameworks e ferramentas que auxiliam no desenvolvimento guiado por testes, utilizando diferentes tecnologias, como *JUnit*, *DbUnit*, *NUnit*, entre outros.

2.2.1.6.
Refactoring

Para que um sistema evolua de forma incremental, é importante revisar constantemente o código e melhorá-lo sempre que necessário.

Refactoring é o ato de alterar um trecho de código sem alterar sua funcionalidade. Ele é utilizado para manter o software mais simples, e se apóia na qualidade e cobertura dos testes para garantir sua validade.

Importante lembrar que o *Refactoring* deve ser realizado com o apoio de testes automatizados. Quando uma alteração é realizada em um código, existe a chance de sua funcionalidade inicial ser alterada acidentalmente. A forma de validar um *Refactoring* é garantindo que o comportamento do trecho do código alterado continue gerando informações previamente definidas. O XP prega que testes automatizados sejam criados, garantindo que o código continua produzindo as mesmas respostas mesmo depois de ter sido alterado. Os testes provêem uma enorme segurança ao processo de *Refactoring*.

2.2.1.7.
Código coletivo

No XP, os pares se revezam no desenvolvimento de módulos de um projeto, as pessoas se revezam na formação dos pares e todos têm acesso e autorização para editar qualquer parte do código da aplicação, a qualquer momento. Ou seja, a propriedade do código é coletiva e todos são igualmente responsáveis por todas as partes. Com isso, os desenvolvedores ganham tempo, pois não precisam esperar a autorização de um colega para editar uma área do código e há maior disseminação de conhecimento.

O risco de um par realizar uma alteração no sistema que venha a quebrar a funcionalidade desenvolvida por um par anterior é minimizada pela execução constante de testes e pela velocidade do *feedback*.

2.2.1.8.
Código padronizado

Para que todos os desenvolvedores possam trabalhar em qualquer parte do sistema, a equipe desenvolve padrões de codificação. Tornando o sistema mais homogêneo e facilitando sua manutenção.

2.2.1.9.
Design Simples

Para garantir que o cliente receba *feedback* o mais rápido possível, os desenvolvedores devem optar sempre pela simplicidade do projeto. Ao invés de criar generalizações no código, pensando nas possíveis mudanças necessárias no futuro, e equipe foca nas necessidades do momento atual, partindo do principio que: quaisquer mudanças que surjam no futuro, não geraram um retrabalho muito grande, já que o design está simples, o código está testado e padronizado.

2.2.1.10.
Metáfora

Para manter o design simples, a equipe utiliza metáforas. Permitindo que a equipe trate de idéias complexas com uma linguagem simples entre seus elementos e o cliente.

2.2.1.11.
Ritmo Sustentável

A qualidade do projeto é diretamente proporcional à qualidade dos desenvolvedores. Nenhum desenvolvedor consegue manter-se atento, criativo e disposto a solucionar problemas trabalhando horas extras constantemente. O XP recomenda que a jornada de trabalho seja no máximo de oito horas por dia.

2.2.1.12.
Integração Contínua

Quando uma nova funcionalidade é incorporada ao sistema, ela pode afetar outras que já estavam validadas pelo cliente. Para garantir que isto não ocorra, os componentes da equipe integram seus códigos constantemente ao longo do dia, garantindo o controle através dos testes. Existem ferramentas que realizam esta verificação automaticamente, como o Hudson [3].

2.2.1.13.
Releases Curtos

Para garantir o *feedback* constante do cliente, o XP utiliza *relases* curtos. Ou seja, a equipe produz um conjunto pequeno de funcionalidades e os coloca em produção rapidamente. Durante todo o projeto, a equipe coloca em produção o projeto várias vezes, incluindo sempre novas e novas funcionalidades.

---

3 https://hudson.dev.java.net/

2.2.2.
Equipe

Uma equipe XP possui um formato próprio. Abaixo são apresentados os papéis de uma equipe:

- Gerente de Projeto: Responsável pelas atividades administrativas do projeto.
- Coach: Responsável técnico do projeto. Orienta a equipe tecnicamente, garantindo que as às práticas de programação apresentadas pelo XP sejam seguidas.
- Analista de Testes: Auxilia o cliente a escrever os testes de aceitação do sistema.
- Redator Técnico: Ajuda a equipe a documentar o sistema. Sua presença permite que a equipe se concentre mais fortemente no código.
- Desenvolvedor: Pessoa que analisa, projeta e desenvolve o software.