4
Processos Ágeis

Este capítulo tem como objetivo estabelecer uma visão geral dos métodos ágeis, destacando principalmente o *Scrum* [11]. Uma vez que o objetivo principal deste estudo envolve a implantação de *Scrum* na organização estudada, é importante que seus conceitos sejam aqui citados, a fim de prover o embasamento teórico adequado.

4.1. História

Apesar de alguns autores defenderem que a origem dos métodos ágeis ocorrera há 75 anos [26], é mais comum a idéia de que os métodos ágeis nasceram a partir da segunda metade de 1990. Nesta época, havia um descontentamento em relação aos métodos dirigidos a planos, devido, principalmente, à extensa documentação gerada no desenvolvimento de software e a alguns problemas de atraso na entrega do software propriamente dito.

Em 2001, foi publicado o Manifesto ágil [9], um documento assinado por dezessete defensores dos métodos ágeis – como Kent Beck, Ken Schwaber, Dave Thomas, entre outros – que possui os princípios desta metodologia de desenvolvimento. Neste documento, os autores reúnem os itens que consideram mais importantes no desenvolvimento de software, que podem ser vistos a seguir.

"Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

– Indivíduos e interações entre eles mais do que processos e ferramentas;

– Software em funcionamento mais do que documentação abrangente;

– Colaboração com o cliente mais do que negociação de contratos;

– Resposta a mudanças mais do que seguimento de um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda [9]".

4.2. Aplicabilidade

A escolha da metodologia a ser utilizada durante o ciclo de vida de um projeto, seja ela ágil ou dirigida a planos, depende de algumas limitações do próprio projeto. A complexidade da atividade de desenvolver software e a grande variedade de métodos existentes para desenvolvê-lo torna a comparação entre métodos ágeis e dirigidos a planos um pouco imprecisa. Porém, há alguns fatores que podem ser levados em conta no momento de escolha de qual estratégia utilizar. São eles: características da aplicação, como tamanho, objetivo e plataforma; características de gestão, como relação com o cliente, planejamento e controle; características técnicas, como os métodos de levantamento de requisitos, desenvolvimento e teste; e características pessoais, como cultura da organização, perfis dos desenvolvedores e do cliente [15].

Em relação ao tamanho e à formação do time, para os métodos ágeis, existe um interesse por times pequenos de desenvolvedores, os quais devem conter uma parcela grande de desenvolvedores sênior. A comunicação entre eles e com os clientes deve ser intensa, rápida e eficiente. Eles defendem uma comunicação face-a-face para as tomadas de decisão e acompanhamento do projeto. Desta maneira, quanto maior for a experiência dos componentes de um time, mais propensa a organização poderá se tornar à adoção de um método ágil.

Neste contexto, é muito utilizado o conhecimento tácito, adquirido pela própria experiência dos desenvolvedores. Assim, em uma organização que haja um rodízio muito grande de colaboradores, a intensa confiança no conhecimento tácito pode ser prejudicial.

Outro fator importante que deve ser levado em consideração é a postura da organização em relação a mudanças. Ela deve ser susceptível a adaptações, deve também aceitar as tomadas de decisões dos desenvolvedores – uma vez que os métodos ágeis defendem que os times devem ser auto-gerenciáveis. Portanto, organizações que possuem uma cultura de poucas mudanças ou de micro-gerência podem oferecer restrição à adoção de um processo ágil.

Em seguida, há também o fator de mutabilidade do projeto: se os requisitos do software sofrem mudanças freqüentes e o projeto em andamento deve responder rapidamente às mesmas, pode ser mais interessante a utilização de métodos ágeis nestes casos. Os métodos dirigidos a planos tendem a estabelecer todo o

cronograma do projeto para ter um controle mais completo do mesmo. Porém, se houver mudanças de escopo, parte do trabalho previsto pode não ser mais realizado, ou novo trabalho pode ser adicionado, tornando necessário redefinir parte do cronograma e das atividades a serem realizadas.

Os métodos ágeis, por sua vez, só detalham um determinado requisito quando este for escolhido para ser desenvolvido em seguida. Ou seja, se uma funcionalidade precisar ser entregue na próxima iteração, é neste momento que os seus detalhes serão buscados. Em contrapartida, não se tem uma estimativa de custo e prazo para o projeto como um todo.

Assim, os métodos ágeis possuem uma grande capacidade de resposta a mudanças. No manifesto ágil, os defensores dos métodos ágeis defendem a rápida resposta em detrimento do seguimento de planos. Porém, ações reativas, apesar de serem vantajosas, algumas vezes podem ser perigosas. Portanto, se o projeto a ser desenvolvido for relativamente estável, pode ser interessante a utilização de métodos dirigidos a planos, que requerem um investimento inicial em processos e planos, mas podem oferecer estabilidade e segurança [15].

Outro fator que deve ser levado em consideração é a quantidade de pessoas trabalhando no mesmo projeto. Métodos ágeis tendem a ser melhor aproveitados em times pequenos e médios. Assim, se um projeto for composto por cem pessoas, por exemplo, pode ser mais interessante a utilização de métodos dirigidos a planos, apesar de haver alguns relatos de sucesso em times maiores [13]. Kent Beck explicita que o tamanho do time impacta diretamente no projeto desenvolvido. Ele diz que até mesmo em um time de vinte desenvolvedores poderia ser difícil a utilização de *Extreme Programming* com sucesso [5].

Por último, a complexidade e o risco do projeto devem ser analisados no momento de escolha da metodologia a ser utilizada. Alguns autores defendem que, na medida em que o risco do projeto aumenta, é importante que se considere a utilização de métodos mais formais – aqueles dirigidos a planos [15]. Contudo, Ken Schwaber, [18] defende que projetos de missão crítica e mais complexos já obtiveram sucesso com a utilização de métodos ágeis.

4.3.Benefícios

Há muitos relatos na literatura de casos de sucesso obtidos a partir da utilização de métodos ágeis [18] [16] [24]. Os processos ágeis se concentram, principalmente, na satisfação do cliente – que deve ser obtida por meio de software que funciona, entregue no prazo acordado. Desta maneira, os esforços de desenvolvimento são concentrados neste objetivo, que, uma vez alcançado, deixa o cliente satisfeito.

Os métodos ágeis defendem a formação de times reduzidos para o desenvolvimento do projeto, o que permite que pequenas empresas possam se tornar mais competitivas no mercado frente a grandes organizações. Grandes organizações podem desenvolver pequenos projetos. Dificilmente, no entanto, pequenas empresas podem desenvolver grandes projetos. A vantagem competitiva está em uma produtividade maior, quando se mede volume de funcionalidade correta entregue por unidade de custo (não tempo). A corretude está vinculada à satisfação do usuário. Daí também a necessidade de poder se amoldar a requisitos cambiantes.

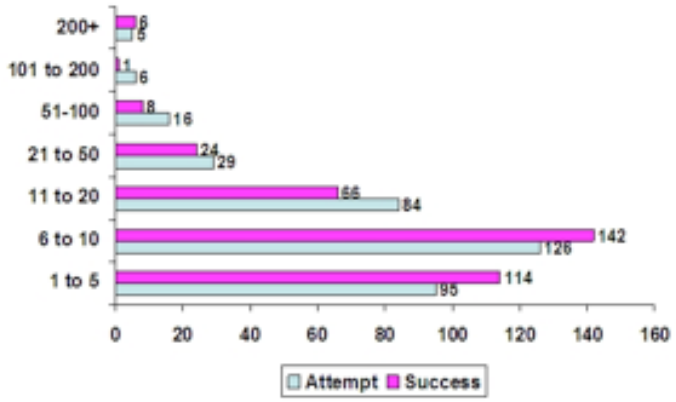

Figura 4 – Relação entre a adoção de *Scrum*, suas histórias de sucesso e o tamanho dos times (http://www.ambysoft.com).

Na Figura 4 é possível visualizar um gráfico que contém as relações entre tentativas de adoção de *Scrum* e histórias de sucesso em times de diferentes tamanhos. Quanto menor o time, maior a adoção de *Scrum*, assim como o número de histórias de sucesso. Com times muito grandes, pode-se constatar que a adoção do *Scrum* se reduz bastante e as histórias de sucesso se tornam cada vez mais raras. Porém, há relatos de tentativas em times de mais de duzentas pessoas, com

considerável número de histórias de sucesso, quando comparamos ao número de tentativas.

Em relação ao time, os processos ágeis destacam a importância de seus componentes serem auto-gerenciados e tomadores de decisão, uma vez que estes são os indivíduos que mais conhecem o projeto que está sendo desenvolvido. Este tipo de gerenciamento acaba por motivar os desenvolvedores, que passam a se sentir mais importantes para o sucesso do software.

Outro ponto positivo dos métodos ágeis é a intensa colaboração do cliente, que participa ativamente dos processos de desenvolvimento, acompanhando as várias iterações realizadas, responsáveis por produzir um incremento de software. Assim, o cliente pode visualizar a evolução do software com o passar do tempo, em vez de esperar um longo período de desenvolvimento para poder ter acesso ao software desenvolvido. Este tipo de desenvolvimento possui a vantagem de correção rápida (e de menor custo) caso alguma falha seja detectada, uma vez que esta é encontrada cedo e sua correção é rápida. Assim, caso haja algum defeito na especificação, por exemplo, este poderá ser descoberto e corrigido ao final da iteração, quando os incrementos de funcionalidades são mostrados ao cliente e devem ser validados pelo mesmo.

4.4.Críticas e Desvantagens

Assim como os métodos de desenvolvimento dirigidos a planos, os métodos ágeis também têm alguns pontos negativos.

No manifesto ágil, indivíduos e interações são considerados mais importantes do que processos e ferramentas. Assim, algumas empresas, utilizando uma interpretação errada desta proposta, acabam negligenciando a utilização de processos e ferramentas que poderiam auxiliar o desenvolvimento. Outro fator negativo dos métodos ágeis é a idéia de que eles devem ser utilizados por desenvolvedores experientes, aqueles considerados seniores. Porém, é complicado e caro encontrar no mercado pessoas com este perfil.

No manifesto ágil, também é dado mais valor a software que funciona do que documentação em excesso. Neste caso, a documentação mínima necessária pode também não estar presente e atrapalhar o desenvolvimento e a manutenção. Documentação técnica é fundamental em um projeto e ela deve ter como foco a

manutenibilidade. Ou seja, a documentação deve ser gerada se ela oferecer uma vantagem no decorrer da vida útil do sistema.

Em [19], constata-se que a utilização de documentação baseada em UML reduz o tempo necessário para se fazer mudanças no código de um programa. Porém, quando se leva em consideração o tempo gasto para alterar os diagramas UML a cada mudança, esta vantagem já não parece ser tão grande. Em compensação, os experimentos presentes neste artigo mostram que, para a realização de tarefas mais complexas, o uso de UML tem forte influência na corretude do trabalho realizado.

Em seguida, no manifesto ágil, a colaboração do cliente é considerada mais importante do que negociações contratuais, ou seja, o cliente deve se sentir parceiro do desenvolvedor e vice-versa. Entretanto, isto pode ocasionar certas dificuldades relacionadas a brechas contratuais.

Por último, algumas grandes empresas confiam mais em lugares que possuem certificações do tipo CMMI – por exemplo – e somente aceitam fazer negócios com fornecedores que a possuam. Ter um certificado CMMI é uma indicação de reputação produzida por um organismo neutro, porém, infelizmente não garante sempre bons serviços.

4.5.Comparações com Outros Modelos

Os métodos de desenvolvimento de software dirigidos a planos, como o CMMI [14], o MPS.BR [17], entre outros, preocupam-se em ter um planejamento detalhado do projeto, prevendo todas as etapas e tarefas a serem realizadas durante o desenvolvimento.

Desta maneira, se os requisitos não sofrerem grandes alterações com o passar do tempo, os processos dirigidos a planos têm mais capacidade de predição das fases do projeto do que os processos ágeis. Por outro lado, uma mudança nos requisitos que afete as próximas etapas do projeto, como arquitetura e desenvolvimento das funcionalidades, poderá invalidar algumas partes do trabalho feitas anteriormente, ocasionando um retrabalho indesejado. Ou seja, o processo de mudanças e as respostas do projeto a estas mudanças são de maneira geral um ponto negativo neste tipo de abordagem, uma vez que elas tendem a ser caras e demoradas.

Os métodos ágeis, por sua vez, possuem três características essenciais: visibilidade, inspeção e adaptação [18].

Neste contexto, visibilidade significa que todos os aspectos relacionados ao processo precisam ser visíveis, além de representarem a realidade. Por exemplo, se um desenvolvedor afirmar que uma funcionalidade está pronta, entende-se que o cliente já pode receber este incremento de software (ele já foi desenvolvido, testado e refatorado). Portanto, é necessário que haja uma comunicação clara entre as partes envolvidas, para minimizar qualquer mal-entendido.

A inspeção, por sua vez, é um pilar importante dos métodos ágeis, pois permite que o processo tenha um acompanhamento intenso para verificação de possíveis pontos de mudança e adaptação. Um bom exemplo de inspeção usada nos métodos ágeis é a revisão de código realizada ao empregar o desenvolvimento em pares. Normalmente, verifica-se o código desenvolvido atende aos padrões de codificação exigidos, se apresenta conformidade com a especificação e se a estrutura de arquivos criada está de acordo com o esperado.

Por último, a adaptação também é um conceito importante para os métodos ágeis, uma vez que estes devem ser capazes de realizar adaptações quando necessário. Estas adaptações no contexto dos métodos ágeis podem ser vistas como a rápida resposta a mudança de requisitos e a constante preocupação com melhorias no processo (que muitas vezes são levantadas em curtas reuniões para este fim).

Alguns representantes do desenvolvimento Ágil são: *Lean*, *Crystal*, *eX-Treme Programming* e *Scrum*. Este trabalho tem como objeto de estudo principal, o *Scrum*.

4.6.Scrum

O *Scrum* é um método ágil de desenvolvimento criado na década de 1990 por Jeff Sutherland [22]. Porém, Ken Schwaber [11] adicionou algumas características ao modelo, tornando-se sua referência. O *Scrum* defende o desenvolvimento incremental e iterativo de software. Iterativo porque quebra o projeto em pedaços a serem realizados um após o outro e incremental porque cada pedaço leva a um resultado útil, que acrescenta alguma característica de interesse

(acrescenta valor) adequada ao usuário, isto é, satisfaz necessidades e expectativas explícitas e implícitas.

4.6.1.Papéis

Há três figuras principais que compõem um time no *Scrum*: o *Product Owner* (PO), o *Scrum Master* (SM) e o próprio time de desenvolvedores, testadores e designer.

O *Product Owner* é o representante do cliente; é ele o responsável por listar as funcionalidades e características do software a ser desenvolvido, priorizando o que deve ser feito antes. É ele também quem estabelece os retornos de investimento (ROI) que deseja alcançar e o plano de lançamentos de versões.

O *Scrum Master* é o responsável por garantir que as práticas do *Scrum* estão sendo seguidas e também por auxiliar os demais integrantes do time, removendo impedimentos que possam surgir durante a execução das tarefas. Ou seja, ele possui o papel de garantir o correto funcionamento do *Scrum*.

Já o time é composto por todos os demais membros (desenvolvedores, testadores, designers, etc.) e deve ser auto-gerenciável, auto-organizável e multidisciplinar. O time recebe a lista de funcionalidades priorizadas pelo PO e escolhe aquelas que consegue entregar até o final da próxima iteração. Ele também é o responsável por transformar as funcionalidades em tarefas que permitam o desenvolvimento e a entrega das mesmas ao encerrar o ciclo.

4.6.2.Reuniões

Existem fundamentalmente quatro tipos de reuniões no *Scrum*: a reunião de planejamento, o *Daily Scrum*, a reunião de revisão de uma iteração e a reunião de retrospectiva de uma iteração.

A reunião de planejamento é realizada antes do início de cada *Sprint* (iteração). É nesta reunião que o *Product Owner* mostra ao time a lista priorizada de itens que devem ser desenvolvidos nas próximas iterações. Cada item é apresentado individualmente para que o time conheça alguns dos seus detalhes e possa estimar o esforço para completá-lo. Em seguida, com base no histórico de *Sprints* anteriores, o time escolhe a quantidade de itens que consegue realizar no próximo ciclo. Por último, os itens escolhidos são quebrados em tarefas menores e

o tempo para completar cada tarefa individualmente é estimado, para verificar se o escopo assumido cabe no total de horas disponíveis do time.

O *Daily Scrum* é a única reunião que ocorre durante um *Sprint*. Sua periodicidade é diária, ela ocorre sempre no mesmo horário e com os participantes em pé, em frente ao quadro onde ficam todos os itens a serem desenvolvidos no ciclo corrente. Nesta reunião, o *Scrum Master* pergunta a cada um dos integrantes do time o que ele fez desde a última reunião, o que ele vai fazer até a próxima e quais os impedimentos encontrados durante a realização da tarefa. Esta reunião atende ao princípio da visibilidade e transparência, uma vez que os integrantes do time estarão sempre atualizados em relação ao trabalho de cada indivíduo e, conseqüentemente, ao andamento do projeto. Esta reunião também é importante, uma vez que é um meio de comunicação rápido e eficiente entre todos os integrantes do time. O PO pode participar desta reunião para poder acompanhar o projeto mais de perto. Já ao *Scrum Master*, é atribuída a responsabilidade de remover os impedimentos citados pelo time nesta reunião.

As reuniões de revisão e de retrospectiva ocorrem imediatamente após o término do *Sprint*. Na primeira, o time apresenta ao *Product Owner* todo o trabalho feito durante o ciclo que acabou de terminar. O PO, por sua vez, verifica se os itens foram feitos da maneira como esperado, ou seja, é neste momento que há a aceitação ou não dos itens. Caso todos os itens sejam aprovados, o *Sprint* termina com sucesso. Caso contrário, algum trabalho ainda terá de ser feito para o PO considerar os itens prontos.

Na reunião de retrospectiva, na qual participa preferencialmente apenas o time e o SM, são discutidos assuntos relativos ao processo em si, como, por exemplo: quais os eventos que impactaram o *Sprint*, o que funcionou no último ciclo e o que pode ser melhorado. Este último tema é separado entre o que é responsabilidade da organização e o que é responsabilidade do time. Ao final, estes itens são priorizados para que medidas de melhoria sejam empregadas conforme a necessidade.

É muito importante entender o conceito de *time-box* das reuniões do *Scrum*. Este conceito está relacionado à idéia de simplicidade, defendida por este método ágil. Ele consiste na importância de as reuniões terem horário para iniciar e horário para terminar, para que elas não se durem mais do que o necessário. O *Daily Scrum*, por exemplo, deve durar no máximo 15 minutos, uma vez que os

times do *Scrum* são reduzidos e o trabalho de um dia não é muito extenso. Isto é, os desenvolvedores precisam ter senso de pragmatismo, citando somente aquilo que realmente importa.

As reuniões de planejamento e de revisão dependem, principalmente, do tamanho do *Sprint*, uma vez que nesta são discutidos assuntos relacionados ao trabalho desenvolvido no último ciclo e naquela, o trabalho a ser desenvolvido no próximo. Já a reunião de retrospectiva deve também ter o seu tempo controlado para que os integrantes do time discutam apenas o mais prioritário (normalmente, o mais importante é a seção em que se discute o que pode ser melhorado). Uma vez que em apenas um ciclo é complicado resolver todos os problemas do desenvolvimento, é importante que haja um foco nos itens que realmente impactam o *Sprint* e que podem ser melhorados na próxima iteração.

O *Sprint*, por sua vez, também representa o conceito de *time-box*, uma vez que os desenvolvedores estão escolhendo trabalho suficiente para ser desenvolvido em um determinado espaço de tempo. O tamanho do *Sprint* também está relacionado à capacidade da empresa prever seu trabalho e suas prioridades. Isto é, se uma empresa não consegue ter certeza de que suas prioridades não mudarão em um intervalo de um mês, talvez seja melhor que ela tenha *Sprints* mais curtos para não correr o risco de mudar o trabalho planejado após o início da iteração.

4.6.3.Fluxos

O desenvolvimento no *Scrum* é iterativo e incremental, isto é, em um ciclo de desenvolvimento, novas funcionalidades são desenvolvidas e entregues ao final do ciclo (idealmente aquelas que agreguem maior valor para o cliente).

Este ciclo de desenvolvimento é denominado de *Sprint* e ele pode durar – preferencialmente – de duas a quatro semanas.

Durante o *Sprint*, representado pela iteração maior na Figura 1, ocorrem as reuniões diárias (*Daily Scrum Meeting*), onde o time se compromete a terminar uma determinada quantidade de trabalho até a próxima reunião, no dia seguinte. O *Daily Scrum* está representado na mesma figura, como a iteração menor, dentro do *Sprint*.

É na reunião de planejamento que o time se compromete a terminar uma determinada quantidade de trabalho, realizada durante o *Sprint*. Esta lista de itens

a serem realizados representa o *Sprint Backlog*, que será detalhado na seção a seguir.

4.6.4.Artefatos

Há alguns artefatos que devem ser produzidos durante a adoção do *Scrum*: o *Product Backlog*, o *Sprint Backlog*, e o *Burndown Chart*. O primeiro é uma lista priorizada de itens a serem desenvolvidos, com seus esforços estimados pelo time, cuja manutenção é de responsabilidade do *Product Owner*. O segundo artefato representa a lista de itens assumidos pelo time para serem entregues ao final de um *Sprint*. Normalmente, possui os itens a serem feitos e todas as tarefas necessárias para a conclusão de cada item. Por último, o *Burndown Chart* é o gráfico de acompanhamento diário das tarefas, preenchido após cada *Daily Scrum*, para facilitar o acompanhamento do ciclo atual.

Na Figura 5, há uma representação do quadro utilizado por um time que adota o *Scrum*. Nele, é possível visualizar à esquerda os itens do *Sprint Backlog* (pedaços brancos de papel) e as tarefas que devem ser realizadas para finalizar o item (os pedaços menores de papel, em amarelo). Todos estes itens estão concentrados na coluna de itens a serem feitos. As próximas colunas representam o trabalho em andamento e o trabalho terminado.

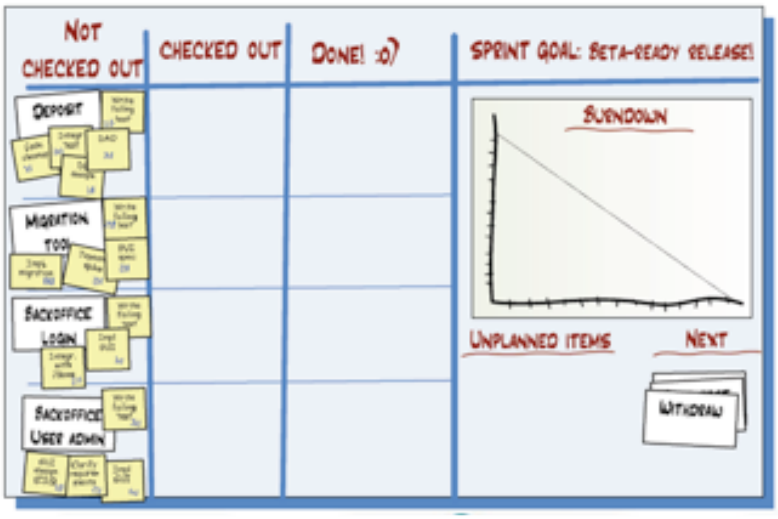

Figura 5 – Exemplo de um *Sprint Backlog* e de um *Burndown Chart* (http://justaddwater.dk/wp-content/uploads/2008/03/burndown.png).

À direita da figura está o *Burndown Chart*, o gráfico de acompanhamento diário das tarefas. É desejável que a linha que representa o trabalho real esteja sempre abaixo da linha que aparece na figura. Na mesma figura, também é possível visualizar a seção de itens não planejados, que impactam o andamento normal do *Sprint* – à direita.

Por último, há um documento que, apesar de não ser obrigatório no *Scrum*, é altamentente defendido por muitos autores como imprescindível na adoção deste modelo [30], [27] e [23]: o *Definition of Done* (DoD).

4.6.5.*Definition of Done* (DoD)

Apesar de o *Scrum* ser uma técnica de gerência de projetos e de não estabelecer a forma e a tecnologia a serem utilizadas pelo time, ele defende que processos técnicos e infraestrutura adequada são importantes para o sucesso de um projeto.

Em [18], Schwaber menciona que ajudava os times a entender que a existência de uma infraestrutura adequada e a utilização de processos da Engenharia de Software são necessários para o *Scrum* funcionar. Em seu livro, ele afirma que as práticas de inspeção e adaptação do *Scrum* somente funcionam se estiverem apoiadas em engenharia de excelência. Isto é, uma organização não é capaz de verificar os benefícios do *Scrum* se não melhorar seus processos de engenharia.

Desta maneira, uma vez que o *Scrum* não trata diretamente de assuntos relacionados à tecnologia, infraestrutura e processos utilizados, é interessante a utilização de outras metodologias para cobrir esta lacuna, como por exemplo, utilizar o CMMI para garantir que os processos de engenharia estão bem estabelecidos e que há uma infraestrutura adequada. O CMMI defende aspectos interessantes para o *Scrum*, como: gerência de configuração, treinamento dos integrantes da equipe, medições quantitativas, gerência de qualidade, preocupação com melhoria constante, verificação, validação e outros.

É nexte contexto que surge a ideia do *Definition of Done*, isto é, que características uma funcionalidade deve possuir para ser considerada entregue.

O DoD pode então conter alguns itens relacionados ao CMMI, uma vez que representa uma *checklist* de atividades requeridas para aceitar o incremento da funcionalidade. Isto é, cada item do *Sprint Backlog*, para ser considerado

entregue, deve atender a esta *checklist*. As atividades que fazem parte do DoD podem ser (por exemplo): fazer esboço, atualizar o manual, fazer teste de unidade, ter cobertura de código acima de 75%, fazer teste de aceitação, entre outras. Normalmente, esta lista é criada pelo time e validada pelo *Product Owner*. Seu escopo não é imutável, porém, é desejável que ela não sofra muitas alterações. Na Figura 6, tem-se um exemplo de um DoD.

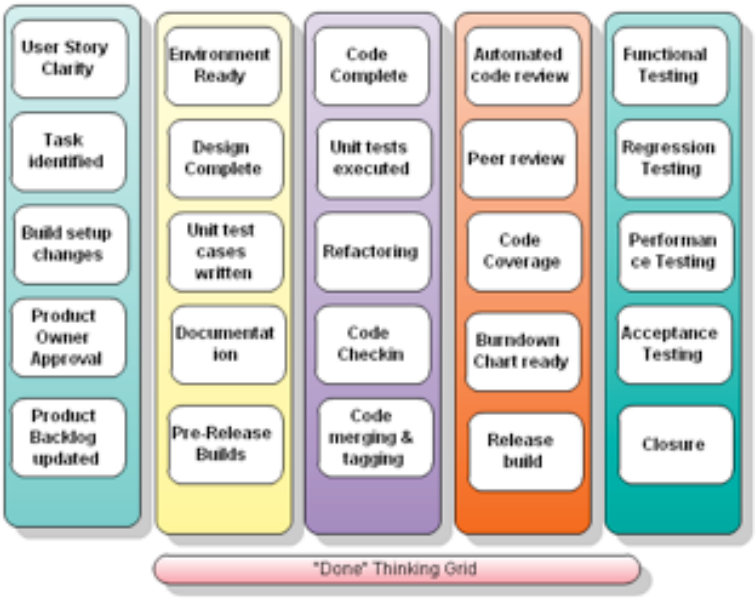

Figura 6 – Exemplo de um documento "*Definition of Done*" (http://blog.vsharing.com/agiledo/A933111.html).

4.7.Resumo

Os processos ágeis de desenvolvimento de software possuem algumas características em comum; seu ponto chave é a satisfação do cliente, obtida por meio da entrega incremental de software que produz resultados satisfatórios logo nas primeiras etapas do projeto.

Os métodos ágeis recomendam a adoção de times pequenos, constituídos por pessoas experientes e altamente motivadas, que utilizam métodos de desenvolvimento focados na simplicidade do projeto e implementação [22]. Porém, é necessário avaliar alguns fatores para verificar se um método ágil pode ser utilizado em um determinado projeto, como: perfil da organização, dos desenvolvedores e do cliente, criticidade e tamanho do projeto, mutabilidade dos requisitos, entre outros.

A utilização de métodos ágeis é algo que está crescendo entre empresas do ramo de tecnologia e pode gerar grandes benefícios a seus adeptos. Principalmente, quando se trata de pequenas e médias empresas, que podem se tornar mais competitivas, uma vez que a produtividade é maior, quando se mede o volume de funcionalidade correta entregue por unidade de custo.