UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

TÁLISSON OLIVEIRA DA COSTA

GERENCIAMENTO DE DEPENDÊNCIAS EM EQUIPES ÁGEIS QUE UTILIZAM SCRUM EM ESCALA

CHAPECÓ
2019

TÁLISSON OLIVEIRA DA COSTA

GERENCIAMENTO DE DEPENDÊNCIAS EM EQUIPES ÁGEIS QUE UTILIZAM SCRUM EM ESCALA

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Raquel Aparecida Pegoraro, Drª

CHAPECÓ
2019

Costa, Tálisson Oliveira da

Gerenciamento de dependências em equipes ágeis que utilizam Scrum em escala / Tálisson Oliveira da Costa. – 2019.
61 f.: il.

Orientador: Raquel Aparecida Pegoraro, Drª .
Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2019.

1. Métodos ágeis. 2. Scrum. 3. Dependências. 4. Múltiplas equipes. 5. Scrum em escala. I. Drª , Raquel Aparecida Pegoraro„ orientador. II. Universidade Federal da Fronteira Sul. III. Título.

E-mail: talisson.odcosta@gmail.com

TÁLISSON OLIVEIRA DA COSTA

GERENCIAMENTO DE DEPENDÊNCIAS EM EQUIPES ÁGEIS QUE UTILIZAM SCRUM EM ESCALA

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Raquel Aparecida Pegoraro, Drª

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 29/11/2019.

BANCA AVALIADORA

Raquel Aparecida Pegoraro, Drª – UFFS

Graziela Simone Tonin, Drª. – UFFS

Doglas André Finco. Esp. – UFFS

RESUMO

Os métodos ágeis vem ganhando cada vez mais popularidade, uso e aceitação na comunidade de desenvolvimento de software. Da mesma forma, *frameworks* como o Scrum, que originalmente foi recomendado para uso por equipes pequenas, vem ganhando cada vez mais adoção em grandes projetos de software. Com a necessidade de escalar, múltiplas equipes trabalham juntas no mesmo projeto, compartilhando recursos e código e em decorrência a isso o surgimento das dependências é inevitável. Para que seja possível identificá-las e monitorá-las, a coordenação e a comunicação se tornam vitais para viabilizar o crescimento dos projetos. Este trabalho teve como objetivo apresentar uma proposta de processo e artefatos para gerenciar dependências em equipes que utilizam Scrum em escala. Para alcançar o objetivo foi adotada uma metodologia de 2 etapas. Na etapa 1 foi realizada a revisão de literatura, que buscou identificar os problemas de dependência vivenciados pelas empresas que utilizam o método Scrum em múltiplas equipes e quais estratégias utilizam para resolver ou amenizar os problemas de dependência; e as práticas utilizadas para realizar a reunião de planejamento da sprint que possibilitam identificar e monitorar as dependências. Na etapa 2 foi proposto um processo de planejamento de dependência em múltiplas equipes Scrum que com o intuito de identificar e monitorar dependências, também foi proposto um quadro de tarefas e artefatos para monitorar as dependências entre as equipes de forma visual.

Como resultado, através da compreensão dos problemas, da identificação das práticas mais utilizadas e da visão das lacunas da gestão de dependências em múltiplas equipes ágeis, foi possível apresentar uma proposta de processo, de quadro de monitoramento e de artefatos para documentar e controlar as dependências, assim esperando contribuir para amenizar os problemas neste contexto.

Palavras-chave: Métodos ágeis. Scrum. Dependências. Múltiplas equipes. Scrum em escala.

ABSTRACT

Agile methods have been gaining increasing popularity, use and acceptance in the software development community and frameworks like Scrum, which was originally recommended for use by small teams, has been gaining increasing adoption in large software projects. In addiction, with the need to scale, multiple teams are involved in the same project, sharing resources and code, as a result dependencies are inevitable. In order to identify and monitor them, coordination and communication become vital to project growth. This paper aimed to present a proposal of process and artifacts to manage dependencies in teams that use Scrum at scale. To achieve the objective, a 3-step methodology was adopted. To achieve the objective, a 3-step methodology was adopted. In step 1, a literature review was performed, which sought to identify the dependency problems experienced by companies that use the Scrum method in multiple teams system and which strategies they use to solve or mitigate the dependency problems; and the practices used to run a sprint planning in order to promote the identification and monitoring of dependencies. In step 2 a Scrum multi-team dependency planning process was proposed, in order to identify and monitor dependencies, also proposed a board and artifacts to monitor dependencies between teams visually. As a result, by understanding the problems, identifying the most commonly used practices, and identifying the gaps in dependency management across multiple agile teams, it was possible to present a process, monitoring framework, and artifact proposal to document and control dependencies, thus hoping to contribute to alleviate the problems in this context.

Keywords: Agile methods. Scrum. Dependencies. Multi-team. Scaling Scrum. Coordination.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

NIT Nexus Integration Team

PD Pesquisa e Desenvolvimento

PO Product Owner

TI Tecnologia da Informação

XP eXtreme Programming

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Desde a sua definição no manifesto de 2001 (3), o desenvolvimento ágil de software vem ganhando cada vez mais popularidade, uso e aceitação na comunidade de desenvolvimento de software (17). As conclusões dos resultados da pesquisa de 2019 sobre as taxas de sucesso de projetos de TI evidenciam a superioridade dos métodos ágeis no contexto de eficácia e sucesso em relação aos métodos tradicionais (19).

Alguns métodos ágeis, como Scrum e eXtreme Programming (XP), foram originalmente recomendados para serem utilizados por equipes pequenas, cujos membros são organizados com o intuito de trabalharem cara a cara, preferencialmente no mesmo ambiente de trabalho (21).

A pesquisa realizada por Wheelan (35) descobriu que pequenos grupos de 3-6 membros são mais produtivos do que grupos maiores e alcançam alta produtividade mais rapidamente. Segundo Rodríguez et al. (26) equipes de desenvolvimento de software de 9 ou mais membros são menos produtivas que equipes menores.

Os métodos ágeis foram originalmente projetados para pequenas equipes (22), mas devido a sua popularidade, benefícios demonstrados e potenciais os tornaram atraentes também fora desse contexto, particularmente para projetos maiores (11, 22). Eles foram desenvolvidos inicialmente para equipes únicas de cinco a nove desenvolvedores e adaptados para uso em projetos com dezenas de equipes e centenas de desenvolvedores, que envolvem a integração com centenas de sistemas existentes e afetam centenas de milhares de usuários (12).

A adoção do Scrum em grande escala acaba desencadeando a necessidade de organizar os integrantes em múltiplas equipes trabalhando juntas para viabilizar o desenvolvimento do produto (7, 16). Neste contexto, cada equipe ágil não pode trabalhar isoladamente e, portanto, a coordenação entre as equipes é uma necessidade (16). Porém, as bibliografias disponíveis sobre o Scrum, entre eles o Scrum Guide (28) apresentam pouca orientação em relação à coordenação entre equipes, dificultando a sua adoção a medida que o número de integrantes aumenta dentro das empresas de software (2).

Com muitas equipes trabalhando juntas na mesma versão, em recursos e código compartilhados, é impossível antecipar todos os problemas, impedimentos, mudanças, falhas e sucessos que as equipes encontrarão durante o desenvolvimento do software (2). Então, a comunicação entre equipes se torna vital para viabilizar esse crescimento sendo altamente relacionada às práticas ágeis usadas. Segundo Rahy e Bass (25) as práticas ágeis mais utilizada são as reuniões diárias em pé, com 90% de uso, seguidas pelo planejamento da sprint (*Sprint Planning*), com 88% de uso, e a terceira são as reuniões de retrospectivas, com 85%, sendo que todas essas práticas exigem habilidades de comunicação eficazes (25).

Com o intuito de possibilitar a adoção do Scrum em escala, a Scrum.org propôs o Nexus. O Nexus é um *framework* desenvolvido para auxiliar múltiplas equipes Scrum a trabalharem juntas desenvolvendo o mesmo produto (29). O Nexus é Scrum, com algums pequenos incrementos que possibilitam a colaboração e transparência entre as equipes. A principal diferença

é que mais atenção é dada para as dependências e comunicação entre as equipes Scrum (7).

O Nexus pode ser pensado como uma espécie de "exoesqueleto" que protege e fortalece as equipes Scrum, simplificando e auxiliando no gerenciando das dependências (7).Detendo-se ao fato de que o Nexus preza pela simplicidade e usufrui de todos os benefícios do Scrum, a sua adoção é extremamente fácil para as equipes Scrum, reduzindo consideravelmente o impacto da implantação de um novo *framework*, pois como no Scrum, o foco está altamente relacionado ao valor do produto (13).

Um dos principais problemas do Scrum em escala é quando os requisitos possuem dependências entre eles, as quais restrinjam a ordem de implementação. Essas dependências podem ser difíceis e complexas de identificar (16). Essa imprevisibilidade é um dos motivos pelos quais o gerenciamento de dependências é difícil (2) e a falta de comunicação entre as equipes pode causar o atraso da sprint, e assim afetando o planejamento, que poderia ter sido evitado se a comunicação tivesse ocorrido de forma eficiente (25). Outro ponto importante é que as dependências podem levar a conflitos entre equipes, como por exemplo, se uma equipe precisa de outra para realizar uma tarefa, deverá haver uma negociação em torno dessa tarefa. Se eles negociam de maneira eficaz e chegam a um acordo de quando a tarefa pode ser realizada, então, a administração dessa dependência se torna mais fácil. No entanto, se eles não chegarem a um acordo claro, ou se o trabalho ainda for uma prioridade menor para a outra equipe, haverá maior risco de incerteza em torno dessa dependência (2).

Devido ao dinamismo que o Scrum fornece, possibilitando alterar as prioridades das tarefas em cada sprint, pode dificultar ainda mais o gerenciamento das dependências, tornando assim a identificação das dependências uma atividade continua e não ocorrendo somente no inicio da *release* (2).

Sendo a identificação e gerenciamento de dependências um dos principais problemas em empresas que trabalham com múltiplas equipes Scrum. É de grande valor a criação de um processo de trabalho que permita estabelecer periodicidade e práticas de comunicação, e artefatos que possibilitem a visualização das dependências durante as reuniões de planejamento e também que permita seu monitoramento ao longo da execução da sprint. Baseando-se nesta premissa e na escassez de recomendações da literatura de como gerenciar essas dependências, foram definidos os objetivos desta pesquisa.

1.1 OBJETIVOS DO TRABALHO

O objetivo geral deste trabalho é:

- Apresentar uma proposta de processo e artefatos para gerenciar dependências em equipes que utilizam Scrum em escala.

Baseado no objetivo geral foram definidos os objetivos específicos:

- Identificar os problemas de dependência vivenciados pelas empresas que utilizam o método Scrum em múltiplas equipes e quais estratégias utilizam para resolver ou amenizar os problemas de dependência.
- Apresentar as práticas utilizadas para realizar a reunião de planejamento da sprint que possibilite identificar e monitorar as dependências.
- Propor um processo de planejamento de dependência em múltiplas equipes Scrum que possibilite identificar e monitorar dependências.
- Propor um quadro de tarefas para monitorar as dependências entre as equipes de forma visual.
- Recomendar artefatos a serem criados para gerenciar as dependências.

1.2 DELIMITAÇÃO DA PESQUISA

Esta pesquisa se delimita em analisar o processo de identificação e gerenciamento de tarefas durante a fase de planejamento da sprint (*sprint planning*) em empresas de pequeno e médio porte que utilizam o método Scrum em múltiplas equipes.

1.3 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte forma, no capítulo 1 é apresentado o tema, problema e os objetivos desta pesquisa. No capítulo 2 é apresentado o referencial teórico e os trabalhos correlatos. No capítulo 3 a metodologia da pesquisa e finalmente no capítulo 4 é apresentado os resultados alcançados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 DESENVOLVIMENTO DE SOFTWARE UTILIZANDO MÉTODOS ÁGEIS

Com o atual cenário, em que as empresas operam em um ambiente global que muda rapidamente e precisam ter respostas rápidas para as novas oportunidades, novos mercados e o surgimento de produtos e serviços concorrentes. O software está presente em quase todos os tipos de negócios, então o desenvolvimento de software deve ocorrer com muita rapidez para que as oportunidades sejam aproveitadas, e responder a pressão gerada pela concorrência (31). Em muitas situações, não é possível definir totalmente os requisitos antes do projeto iniciar (24).

A medida que os requisitos mudam ou quando os problemas de requisitos são descobertos, o *design* ou a implementação do sistema deve ser retrabalhado e testado novamente. Como consequência, um processo convencional baseado em cascata ou especificação é geralmente prolongado e o software final é entregue muito depois de ter sido originalmente especificado (31).

Devido a essas necessidades, em 2001, 17 renomados desenvolvedores de software, escritores e consultores assinaram o "Manifesto para o desenvolvimento ágil de software" (24). O manifesto ágil é um conjunto de quatro valores e doze princípios, que compõe a filosofia ágil, que visam melhores formas de desenvolver software (3).

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

Mesmo havendo valor nos itens à direita, valorizamos mais os itens da esquerda (31). Com isso, fica claro a mudança de visão em relação a valores, uma vez que as metodologias tradicionais de desenvolvimento valorizam muito mais os itens à direita.

Além dos quatro valores, o manifesto ágil resultou em 12 princípios:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.

5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.

6. O método mais eficiente e eficaz de transmitir informações para, e por dentro de uma equipe de desenvolvimento, é através de uma conversa cara a cara.

7. Software funcional é a medida primária de progresso.

8. Processos ágeis promovem um ambiente sustentável.Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.

9. Contínua atenção à excelência técnica e bom *design*, aumenta a agilidade.

10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.

11. As melhores arquiteturas, requisitos e *designs* emergem de equipes auto-organizáveis.

12. Em intervalos regulares, a equipe reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

Os métodos ágeis transformaram a maneira como o software é desenvolvido, enfatizando o envolvimento ativo do usuário final, a tolerância a mudanças e a entrega evolutiva de produtos (12). A eficiência dos métodos ágeis fica evidente nas conclusões dos resultados da pesquisa sobre as taxas de sucesso dos projetos de TI de 2013, que atestaram a superioridade dos métodos ágeis no contexto de eficácia e sucesso em relação às suas contrapartes tradicionais (1).

Os métodos ágeis de desenvolvimento têm atraído interesse principalmente em engenharia de software, mas também em várias outras disciplinas, incluindo sistemas de informação e gerenciamento de projetos (12). Dentre os métodos ágeis, alguns *frameworks* se destacam, como por exemplo o Scrum, que segundo Dingsøyr, Falessi e Power (12) atualmente é *framework* mais comum para o desenvolvimento de software (12). Além do mais, os princípios do Scrum são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades de estrutura: requisitos, análise, projeto, evolução e entrega (24).

2.2 ESCALONAMENTO DOS MÉTODOS ÁGEIS

Os métodos ágeis foram originalmente projetados para pequenas equipes (22), mas devido a sua popularidade, seus benefícios demonstrados e potenciais os tornaram atraentes também fora desse contexto, particularmente para projetos maiores (11, 22). Eles foram desenvolvidos inicialmente para equipes únicas de cinco a nove desenvolvedores e adaptado para uso em projetos com dezenas de equipes e centenas de desenvolvedores, que envolvem a integração com centenas de sistemas existentes e afetam centenas de milhares de usuários (12).

Em comparação com pequenos projetos, que são ideais para o desenvolvimento ágil, os projetos maiores são caracterizados pela necessidade de coordenação adicional. Um problema específico na aplicação ágil para esses projetos, é como coordenar equipes. A agilidade em grande escala envolve preocupações adicionais na interface com outras unidades organizacionais, como recursos humanos, marketing, vendas e gerenciamento de produtos (11).

A utilização dos métodos ágeis de grande escala é cada vez mais predominante em organizações contemporâneas de desenvolvimento de software. Embora haja muitos benefícios em potencial, as transformações em larga escala são repletas de desafios, como problemas de comunicação, falta de flexibilidade e desafios de coordenação (9).

Uma diferença significativa entre as adoções de pequena e grande escala é que as organizações maiores têm mais dependências entre projetos e equipes (11). A falta de comunicação entre as equipes pode causar o atraso dos sprints e assim, afetando o planejamento, que poderia ter sido mitigado se a comunicação tivesse ocorrido de forma eficaz (25).No entanto, a identificação e análise de dependência não podem acontecer apenas uma vez no início do lançamento, mas devem ser um processo contínuo (2).

Uma dependência é criada quando o progresso de uma ação depende da saída de uma ação anterior ou da presença de alguma elemento específico. Dependências levam a restrições potenciais ou reais em projetos. As restrições potenciais são aquelas que são atualmente organizadas ou bem gerenciadas, sem causar problemas na progressão de um projeto.Restrições reais são gargalos ou pontos em um projeto que as partes interessadas estão cientes, mas não têm meios imediatos para contornar (32).

Existem diferentes tipos de dependências que emergem durante um projeto com múltiplas equipes. Essas dependências podem ser separadas em três categorias:

1. **Dependência de conhecimento**: Uma dependência de conhecimento ocorre quando uma forma de informação é necessária para que um projeto progrida. Existem quatro formas de dependência do conhecimento (32):

    - **Requisito** - uma situação em que o conhecimento do domínio ou um requisito não é conhecido e deve ser localizado ou identificado e isso afeta o andamento do projeto.
    - **Especialização** - uma situação em que as informações técnicas ou de tarefas são conhecidas apenas por uma pessoa ou grupo específico e isso afeta o andamento do projeto.
    - **Alocação de tarefas** - uma situação em que quem está fazendo o quê e quando, não é conhecido e afeta o andamento do projeto.
    - **Histórico** - uma situação em que o conhecimento sobre decisões passadas é necessário e isso afeta o projeto.

2. **Dependência de tarefas**: Uma dependência de tarefa ocorre quando uma tarefa deve ser concluída antes que outra tarefa possa continuar e isso afeta o andamento do projeto. Existem duas formas de dependência de tarefas (32):

    - **Atividade** - uma situação em que uma atividade não pode prosseguir até que outra atividade seja concluída e isso afeta o andamento do projeto.
    - **Processo de negócios** - uma situação em que um processo de negócios existente faz com que as atividades sejam executadas em uma determinada ordem e isso afeta o andamento do projeto.

3. **Dependência de recursos**: Uma dependência de recurso ocorre quando um objeto é necessário para um projeto progredir. Existem duas formas de dependência de recursos (32):

    - **Entidade** - uma situação em que um recurso (pessoa, lugar ou coisa) não está disponível e isso afeta o andamento do projeto.
    - **Técnico** - uma situação em que um aspecto técnico do desenvolvimento afeta o progresso, como quando um componente de software deve interagir com outro e sua presença ou ausência afeta o andamento do projeto.

Segundo Babinet e Ramanathan (2), algumas estratégias comuns para gerenciar e resolver o problema das dependências incluem a criação de visibilidade, para que todos os envolvidos possam ter um rápido *feedback* em relação ao planejamento da *release*, dependências, *design*, *build*; e a promoção de fóruns para estimular a colaboração e compartilhamento de conhecimento entre as equipes.

Para resolver esses problemas, muitos recorreram a estruturas de desenvolvimento ágil em grande escala, dentre eles o Nexus. No entanto, evidências empíricas sobre a adoção, uso, eficácia e desafios ainda são muito incipientes (9).

Este trabalho tem como foco o estudo em equipes que utilizam o método ágil Scrum em escala, por este motivo escolhemos o *framework* Nexus para adotação. Na seção 2.3 é apresentada a fundamentação teórica sobre o Scrum em grandes projetos e detalhamento do *framework* Nexus.

2.3 SCRUM EM GRANDES PROJETOS

Scrum é um *framework* estrutural que está sendo usado para gerenciar o trabalho em produtos complexos desde o início de 1990. Scrum não é um processo, técnica ou um método definitivo. Em vez disso, é um *framework* dentro do qual pode ser empregados vários processos e técnicas. O Scrum deixa claro a eficácia relativa de suas práticas de gerenciamento de produto e técnicas de trabalho, de modo que você possa continuamente melhorar o produto, a equipe e o ambiente de trabalho. O *framework* Scrum consiste em equipes associadas a papéis, eventos,

artefatos e regras. Cada componente dentro do *framework* serve a um propósito específico e é essencial para o uso e sucesso do Scrum (28).

A abordagem do Scrum para o planejamento da *release* é muito diferente da abordagem usada pelos modelos tradicionais. Em vez de empregar planos de projeto baseados em um conjunto de fatores e restrições predefinidos, o Scrum conta com um julgamento humano colaborativo e negociações informais (16).

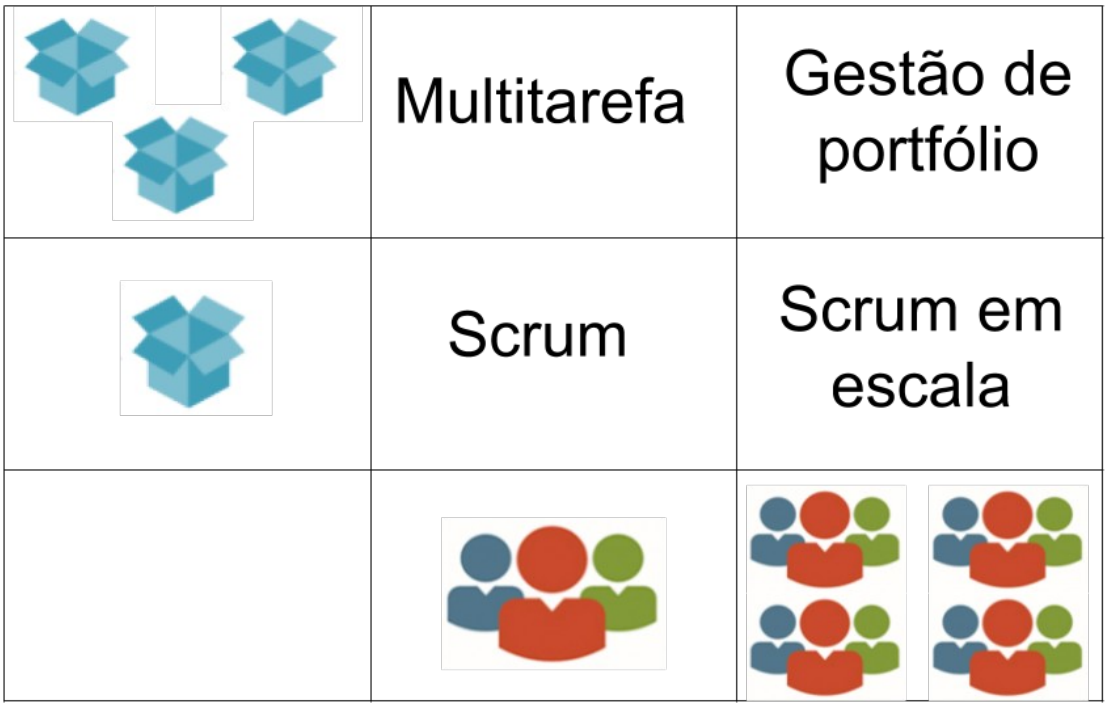
De acordo com a pesquisa de 2013 do State of Agile, o método de desenvolvimento de software Scrum alcançou um *status* estabelecido na comunidade de desenvolvimento de software, e muitas organizações de desenvolvimento de software adotaram ou aspiram adotar métodos ágeis (20).

Segundo Rahy e Bass (25) as práticas ágeis mais utilizadas são as reuniões diárias em pé, com 90% de uso, seguidas pelo Sprint Planning, com 88% de uso, e a terceira posição são as retrospectivas, com 85%, sendo que todas essas práticas exigem habilidades de comunicação eficazes (25).

Quando uma equipe trabalha em um único produto, eles simplesmente podem utilizar o Scrum. Nos casos em que uma equipe trabalha em vários produtos, eles ainda podem se beneficiar do uso do Scrum, se trabalharem no desenvolvimento de cada produto em sprints diferentes. Isso é simplesmente um tipo de multitarefa para gestão de portfólio (6).

O Scrum é direcionado a um único produto sendo produzido por uma única equipe do Scrum. Duas equipes, podem até usar o Scrum para desenvolver um único produto, mas, à medida que mais equipes são adicionadas, as equipes precisam incluir mais processos ao Scrum para gerenciar suas dependências entre equipes e garantir que possam fornecer entregas de qualidade (6). Para possibilitar esse crescimento, a adoção do Scrum em escala vem sendo uma alternativa, pois fornece uma estrutura eficiente para possibilitar a colaboração entre as equipes. Na figura 1 podemos visualizar a relação entre a quantidade de produtos sendo desenvolvidos com a quantidade de equipes presentes no desenvolvimento.

Figura 1 – Scrum em escala



Fonte: Adaptado de Bittner (6)

Muitas equipes trabalhando em muitos produtos independentes ou semi-independentes são realmente um tipo diferente de problema. Neste caso, geralmente é necessário realizar a gestão de portfólio de produtos para coordenar os trabalhos. Podendo utilizar o Scrum em cada projeto, sem nenhum tipo de modificações (6).

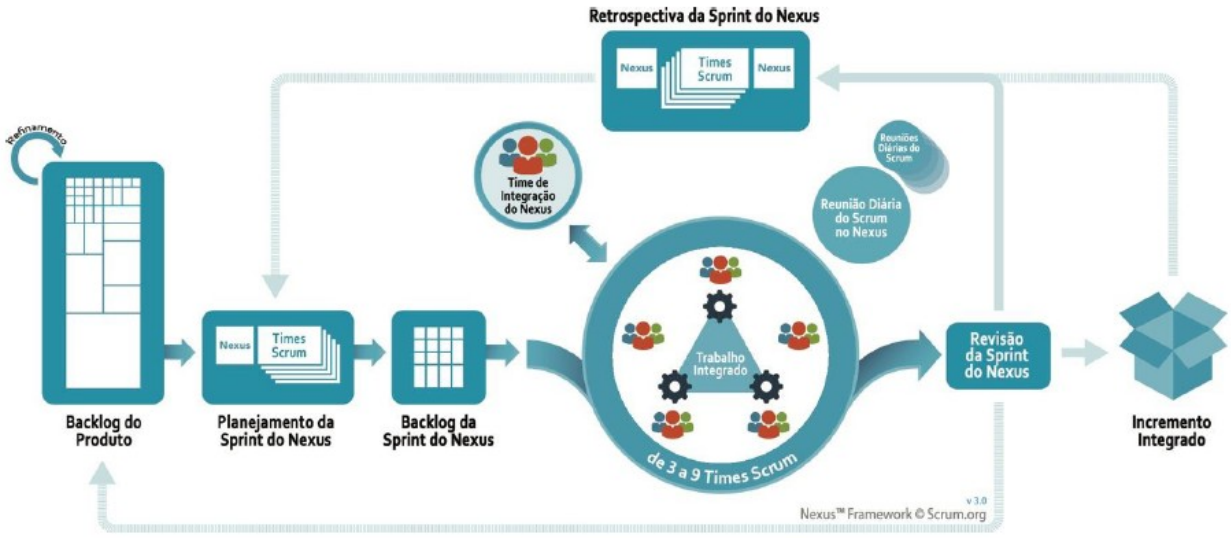
Após esta introdução sobre o *framework* Scrum, será apresentado a seguir o *framework* Nexus, proposto pela Scrum.org, que é responsável por prover mais recursos ao Scrum, proporcionando meios para que as equipes possam trabalhar eficientemente em escala sem abrir mão dos benefícios do Scrum.

2.3.1 Nexus

Nexus é um *framework* para desenvolvimento e sustentação de iniciativas de entrega de produtos e de softwares em escala. Ele usa o Scrum como alicerce para sua construção e é constituído de papéis, eventos, artefatos e regras que os unem e entrelaçam junto o trabalho de várias equipes Scrum em um único *product backlog* para construir um incremento integrado que alcance uma meta (29).

O Nexus é consistente com o Scrum e seus conceitos serão familiares para quem já utilizou o Scrum. A diferença principal é que mais atenção é dada para as dependências e interoperação entre as equipes, entregando pelo menos um incremento integrado e "pronto" a cada sprint (29). A figura 2 apresenta a visão geral do *framework* Nexus (29).

Figura 2 – Visão geral do *framework* Nexus



Fonte: Scrum.org (29)

2.3.1.1 Os eventos do Nexus

O fluxo de processo do Nexus é composto por eventos, que como no Scrum, criam uma regularidade e minimizam a necessidades de reuniões não definidas. A duração dos eventos do Nexus é guiada pelos tamanhos dos seus eventos correspondentes no guia do Scrum (28). O fluxo do Nexus é composto por:

- Refinamento do *product backlog*: O *product backlog* precisa ser decomposto para que as dependências sejam identificadas, removidas ou minimizadas (29).
- Planejamento de sprint do Nexus: Representantes apropriados de cada equipe Scrum se reúnem para discutir e revisar o refinamento do product backlog.Eles selecionam os itens do *product backlog* para cada equipe (29).
- Meta da sprint: A meta da sprint do Nexus é o objetivo definido para a sprint. Ela é a soma de todo o trabalho das sprints das equipes scrum (29).
- Reunião diária do Nexus: Representantes apropriados de cada equipe de desenvolvimento se encontram diariamente para identificar se existe alguma questão de integração. Se identificado, essa informação é transferida de volta para cada reunião diária das equipes Scrum (29).
- Revisão da sprint do Nexus: A Revisão de sprint do Nexus é feita no final da sprint para promover comentários e opiniões sobre o incremento integrado que um Nexus construiu através da sprint (29).

- Restrospectiva da sprint do Nexus: Representantes apropriados de cada equipe Scrum se encontram para identificar os desafios compartilhados. Então, cada equipe Scrum realiza sua reunião de retrospectiva do Scrum individualmente. Representantes apropriados de cada equipe se encontram novamente para discutir quaisquer ações necessárias baseadas nos desafios compartilhados a fim de fornecer inteligência de baixo para cima (29).

2.3.1.2 Os papéis do Nexus

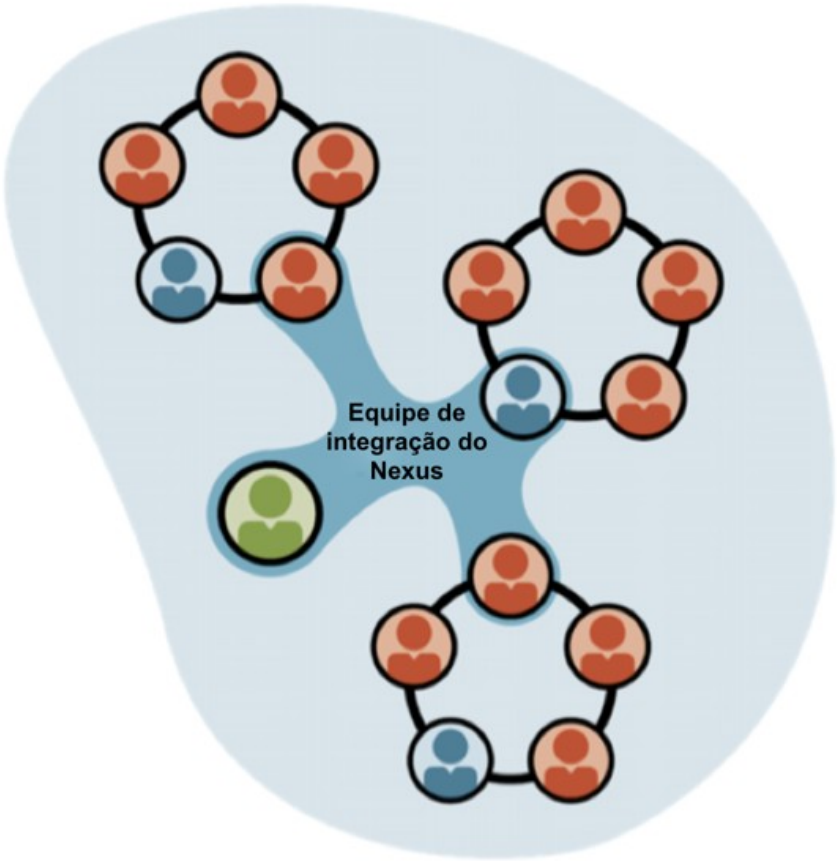
O Nexus apresenta uma equipe chamada de equipe de integração do Nexus - Nexus Integration Team (NIT) para ajudar as demais equipes a lidar com os desafios. Apesar do nome, o NIT geralmente não é uma equipe permanente, com membros da equipe em período integral, nem é realmente encarregado de realizar a integração do trabalho realizado, embora seja responsável por garantir que as equipes tenham condições de realizar a integração com sucesso (29).

Em casos de emergências, o NIT pode atuar como uma equipe Scrum, auxiliando no desenvolvimento. No entanto, geralmente é uma equipe virtual composta por membros das próprias equipes Scrum, além do Product Owner (PO) como mostrado na figura 4. Na figura podemos ver o exemplo de 3 equipes Scrum, compostos pelo PO, pela equipe de desenvolvimento e Scrum Master, identificados na figura 3.No entanto, a composição da equipe pode ser alterado no decorrer do projeto, e a escolha dos integrantes depende dos objetivos e desafios da sprint atual, podendo até mesmo ser composto por membros externos (29).

Figura 3 – Membros da equipe de integração do Nexus

Fonte: Bittner et al. (7)

Figura 4 – Estrutura da equipe de integração do Nexus



Fonte: Bittner et al. (7)

O papel do NIT é muito parecido com o de um Scrum Master em uma equipe Scrum, pois fornece um mecanismo para identificar problemas e facilitar soluções, mas não assume o papel de resolver problemas para as equipes Scrum. O NIT tem responsabilidades muito específicas e é encarregado de aplicar uma série de práticas, com eficácia comprovada, que auxiliam as equipes a resolver os desafios enfrentados (29).

Ao contrário do que o nome pode sugerir, o NIT não integra o trabalho de todas as equipes Scrum. Em vez disso, é responsável por garantir que as equipes sejam capazes de alcançar a integração por conta própria (7).

Os membros treinam as equipes Scrum e ajudam a remover dependências. Se algo está impedindo que as equipes Scrum no Nexus desenvolvimento do produto, o NIT é responsável por garantir que esses problemas sejam resolvidos (7).

2.3.1.3 Os artefatos do Nexus

Os artefatos representam o trabalho ou valor que fornece transparência e oportunidades para inspeção e adaptação.

- Product backlog: Há um único *product backlog* para todo o Nexus e todos as equipes Scrum. O PO é o responsável pelo *product backlog*, incluindo seu conteúdo, disponibilidade e ordenação. O *product backlog* deve ser entendido em um nível onde as dependências possam ser detectadas e minimizadas (29).

- Product backlog da sprint do Nexus: O *product backlog* da sprint do Nexus é composto pelos itens das sprints das equipes Scrum individuais. Ele é usado para destacar as dependências e o fluxo de trabalho durante a sprint e é atualizado pelo menos uma vez ao dia, como parte da reunião diária do Nexus (29).

- Incremento integrado: Um incremento integrado representa a soma atual de todos os trabalhos realizados pelas equipes. Ele deve atender a definição de "pronto" estabelecida para o projeto e é inspecionado na revisão da sprint do Nexus (29).

2.3.1.4 Definição de pronto

A equipe de integração do Nexus é responsável pela definição de "pronto" que pode ser aplicado para o incremento integrado desenvolvido a cada sprint. O incremento é "pronto" somente quando é integrado, utilizável e possível de ser entregue pelo PO (29).

2.4 TRABALHOS CORRELATOS

Babinet e Ramanathan (2) relataram a experiência através de um estudo de caso realizado no setor de Pesquisa e Desenvolvimento (PD) da Salesforce.com que tem mais de 30 equipes Scrum trabalhando simultaneamente em um único *branch*.Babinet e Ramanathan (2) destacam os desafios que foram encontrados em relação ao gerenciamento de dependências escalonando múltiplas equipes Scrum e como superaram os desafios. Babinet e Ramanathan (2) destacam a imprevisibilidade como um dos maiores motivos pelos quais o gerenciamento de dependências é difícil e que o primeiro passo para gerenciar as dependências é identificá-las (2).

Babinet e Ramanathan (2) também relatam a ocorrência de conflitos entre equipes e como isso pode impactar negativamente se não cheguem rapidamente a um consenso sobre a importância das tarefas e priorizar elas de acordo com a importância para minimizar os impactos das dependências (2).

O dinamismo que o Scrum fornece, possibilitando alterar as prioridades das tarefas em cada sprint, pode dificultar ainda mais o gerenciamento das dependências, tornando assim a identificação das dependências uma atividade continua e não ocorrendo somente no inicio da *release*. Dentre as estratégias para gerenciar as dependências, destacam-se a criação de visibilidade, para que todos os envolvidos possam ter um rápido *feedback* em relação ao planejamento da *release*, dependências, *design* e *build*. E a promoção de fóruns para estimular a colaboração e compartilhamento de conhecimento entre as equipes (2).

Para identificar e fornecer visibilidade para todos os interessados no projeto, é realizado um "Exercício para identificação de dependências", que ocorre após as equipes realizarem os seus planejamentos da *release*. Devido a muita discussão informal sobre as dependências, foi compreendido que seria necessário formalizar essas discussões, para englobar todas as equipes e facilitar a comunicação entre eles (2).

O exercício para identificação de dependências consiste em reunir representantes de cada equipe Scrum em uma sala com um grande quadro branco. Na primeira parte, todos os representantes das equipes vão até o quadro e criam dois diagramas para representar as dependências entre as equipes. No primeiro, o intuito é identificar quais equipes precisam que outra equipe realize algo por eles. E no segundo, o intuito é identificar as equipes que estão realizando algo que irá impactar em outras equipes.Para esses diagramas são utilizadas notações como:

- Um círculo para representar a equipe.
- Uma flecha entre as equipes representa a dependência. No primeiro diagrama a flecha aponta da equipe que está realizando a tarefa para a equipe que precisa que a tarefa seja concluída. No segundo diagrama, a flecha aponta para a equipe impactada pela tarefa.
- Um rótulo na flecha descreve a dependência.
- Se a dependência já está resolvida (ambas as equipes entraram em acordo com a prioridade da tarefa), é colocado uma caixa entorno do rótulo da dependência.

Com esse exercício é obtido em cerca de 20 minutos uma visualização clara das dependências, assim é possível identificar as equipes que serão mais afetados pelas dependências, possibilitando tratar as dependências com maior antecedência (2).

A salesforce.com demonstrou que é possível escalar o Scrum à medida que a organização cresce. O gerenciamento de dependências e a coordenação entre equipes são um desafio significativo à medida que o número de equipes aumenta, mas o aumento da visibilidade por meio de práticas é muito útil, pois isso facilita a coordenação a comunicação e o compartilhamento de conhecimento entre as equipes (2).

Martakis e Daneva (17) , descrevem em sua pesquisa que desenvolveram através de um grupo focal, como as dependências são tratadas na indústria de software. O autor deixa claro a falta de pesquisa sobre o assunto e como ainda é um tópico inexplorado na literatura. Para realizar a pesquisa, foi utilizado um fórum onde os participantes poderiam responder as perguntas estabelecidas e também interagir entre eles (17).

Os participantes concordaram que a presença e a importância do gerenciamento das dependências em projetos é algo crucial para o sucesso do projeto, tanto para projetos ágeis quanto em projetos tradicionais. A dependência entre requisitos era considerada parte do gerenciamento de risco, e demandava responsabilidade apenas da equipe, não tendo uma colaboração e uma coordenação para gerenciar as dependências, e em muitos casos afetando o planejamento

do projeto. A comunicação e a colaboração, que são essenciais de projetos ágeis, são vitais para amenizar os impactos das dependências nos projetos (17).

Martakis e Daneva (17) ressaltam a compreensão das dependências como sendo de suma importância para grandes projetos ágeis, pois impactam diretamente na priorização dos requisitos. Se o valor comercial for tratado como critério mais importante, pode se tornar um grande risco, por isso as dependências devem ser levadas em consideração na hora de priorizar.

Heikkiä et al. (16) descrevem a adoção de um método de planejamento de *release*, definido como "método de planejamento da iteração da *release*", que foi empregado em dois projetos na F-Secure, uma grande empresa de software finlandesa. A F-secure estava em uma fase de transição do modelo tradicional para o ágil.O planejamento da *release* consistia em uma série de eventos, realizados no decorrer de 3 dias, que consistia em introdução e visão sobre o projeto, reuniões de planejamento, revisões e retrospectivas.

O projeto de desenvolvimento foi dividido em iterações *release* e cada iteração foi dividida em sprints de desenvolvimento. As equipes de desenvolvimento, juntamente com seus POs, eram responsáveis por criar histórias de usuários com base nos recursos e planejar o conteúdo das sprints de desenvolvimento. A ordem de implementação das histórias de usuário era normalmente baseada nas dependências entre as histórias de usuário (16).

Após o término das apresentações, as equipes de desenvolvimento começaram a planejar a primeira iteração de *release*. Primeiro, as equipes se reuniram juntamente com o PO, que juntamente as equipes discutiam como os recursos devem ser atribuídos às equipes.Depois que cada equipe recebia pelo menos um recurso, as equipes poderiam se encaminhar para os seus respectivos quadros de planejamento para continuar o planejamento individual de cada equipe (16).

Durante o planejamento foram realizadas revisões ao final do primeiro e do segundo dia, chamadas de revisões intermediarias. Todos os participantes do evento se reuniram na área de apresentação e um representante de cada equipe de desenvolvimento, que geralmente era o Scrum Master, relatava em quatro minutos como o planejamento havia progredido e quanto tempo ainda era necessário, quais foram os problemas não resolvidos e quais dependências, se houver, haviam descoberto. Os outros participantes eram incentivados a fazer perguntas e comentar as apresentações (16).

A revisão final do planejamento foi realizada no final do terceiro dia de maneira semelhante às revisões intermediárias do planejamento.Cada equipe teve seis minutos para apresentar o planejamento e os objetivos que eles tinham para a primeira iteração da *release*. As equipes escreveram os risco em notas adesivas e cada risco em potencial foi discutido brevemente e atribuído a uma pessoa ou equipe que estaria lidando com o risco (16).

Após a primeira iteração, foram realizadas mudanças para as próximas iterações, as principais mudanças foram: introdução de uma matriz de planejamento, para fornecer uma visão geral do cronograma de desenvolvimento do progresso do planejamento, além de auxiliar na identificação de conflitos; reorganização das equipes de desenvolvimento, criando equipes

de funcionalidades com o intuito de mitigar o impacto das dependências; introdução de Scrum Masters em tempo integral que contribuiu quebrando tarefas técnicas muito complexas em histórias de usuários (16).

Foi possível identificar que muitos problemas foram enfrentados inicialmente, principalmente pela falta de preparo e experiência dos integrantes das equipes. Porém, apesar das dificuldades encontradas, o método trouxe muitos benefícios como: melhora na comunicação entre as equipes; comunicação cara a cara nos eventos de planejamento, possibilitando a rápida identificação e gerenciamento das dependências; revisões constantes, permitindo o constante refinamento dos requisitos (16).

O método trouxe benefícios, como a transparência obtida através do planejamento da *release*. Os eventos promovidos durante o planejamento resultaram em melhora na comunicação e transparência, promovendo a identificação precoce das dependências. Possibilitando mitigar os impactos das dependências. Assim reduzindo o impacto que as dependências tem ao priorizar o *product backlog* (16).

Heikkiä et al. (16) não deixam claro como as dependências eram identificadas e não mencionam artefatos que identifiquem as dependências, mas deixam claro, a partir dos relatos dos entrevistados a importância da identificação das dependências o mais breve possível e como os eventos que proporcionaram encontros entre os interessados ajudaram a resolver os problemas relacionados as dependências.

Tanto as referências da literatura quanto os trabalhos correlatos, possuem como limitação a falta da definição de um processo que possa ser replicado na prática pelas empresas que possuem dependência entre equipes de desenvolvimento. Também não apresentam a proposta de artefatos (documentos) a serem utilizados para mapear e monitorar as dependências. Esta pesquisa pretende contribuir neste contexto apresentando uma solução para esses problemas.

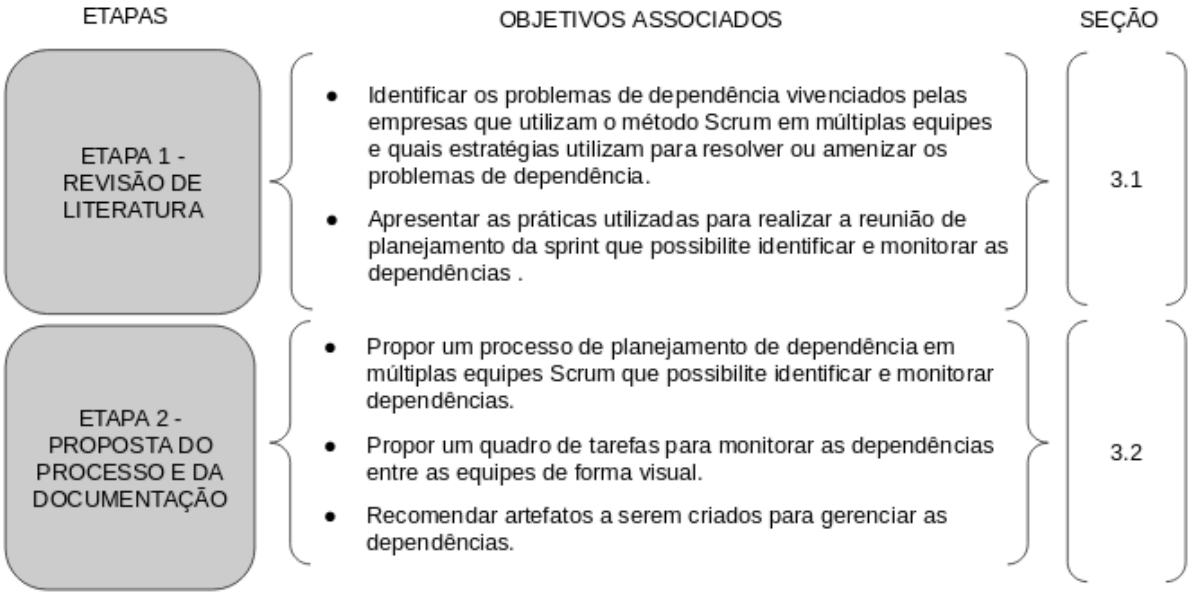
3 METODOLOGIA DE PESQUISA

Essa seção descreve a estrutura da metodologia utilizada para alcançar os objetivos deste trabalho. O objetivo principal:

- Apresentar uma proposta de processo e artefatos para gerenciar dependências em equipes que utilizam Scrum em escala.

O objetivo principal foi desmembrado em 5 objetivos específicos. Para atender esses objetivos está sendo proposta uma metodologia subdividida em duas etapas, sendo cada etapa associada a um objetivo específico, conforme apresentado na figura 5. A figura apresenta as etapas de pesquisa, o objetivo associado aquela etapa e a seção do trabalho em que é descrita a metodologia a ser utilizada.

Figura 5 – Etapas da metodologia de pesquisa



Fonte: Autor

3.1 ETAPA 1 - REVISÃO DE LITERATURA

A etapa inicial buscou atender aos seguintes objetivos específicos:

- Identificar os problemas de dependência vivenciados pelas empresas que utilizam o método Scrum em múltiplas equipes e quais estratégias utilizam para resolver ou amenizar os problemas de dependência.
- Apresentar as práticas utilizadas para realizar a reunião de planejamento da sprint que possibilite identificar e monitorar as dependências.

Para esta fase da pesquisa foi realizada uma revisão de literatura. Segundo Gil (14) a revisão da literatura consiste na identificação, localização e análise de publicações que contêm informações relacionadas ao tema da investigação e busca identificar contribuições teóricas aplicáveis ao estudo (14).

Na revisão de literatura foi realizada uma busca na base da Engineering Village, que concentra artigos das principais bases cientificas como IEEE e SpringerLink, com a string de busca **((scrum and dependenc* and scal*) OR (agile and dependenc* and scal*))**. Os critérios de inclusão para seleção dos artigos foram: estudos que tratem especificamente sobre a dependência entre múltiplas equipes ágeis. Foram encontrados 109 artigos, dos quais 98 foram descartados por não tratar sobre dependências entre equipes de desenvolvimento de software ou por não tratar sobre dependências entre múltiplas equipes.

Tabela 1 – Estudos selecionados pela string de busca

Titulo do artigo	**Referência**
Operational release planning in large-scala Scrum with multiple stakeholders - A longitudinal case study at F-secure Corporation	Heikkiä et al. (16)
Technical dependency challenges in large-scale agile software development	Sekitoleko et al. (30)
The effects of team backlog dependencies on agile multiteam systems: A graph theoretical approach	Scheerer et al. (27)
Inter-team coordination in large agile software development settings: five ways of practicing agile at scale	Bick, Scheerer e Spohrer (4)
Handling requirements dependencies in agile projects - A focus group with agile software development practitioners	Martakis e Daneva (17)
Dependency management in a large agile environment	Babinet e Ramanathan (2)
Identifying and structuring challenges in large-scale agile development based on a structured literature review	Uludag et al. (33)
Challenges and success factors for large-scale agile transformations: A systematic literature review	Dikert, Paasivaara e Lassenius (11)
Networking in a Large-Scale Distributed Agile Project	Moe et al. (18)
Agile requirements prioritization in large-scale outsourced systemprojects: An empirical study	Daneva et al. (10)
Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings	Bick et al. (5)

Fonte: Autor

3.2 ETAPA 2 - PROPOSTA DO PROCESSO E DA DOCUMENTAÇÃO

Após compreender o atual estado da pesquisa e da prática, então foi realizada a segunda etapa da pesquisa que buscou atender aos seguintes objetivos específicos:

- Propor um processo de planejamento de dependência em múltiplas equipes Scrum que possibilite identificar e monitorar dependências.
- Propor um quadro de tarefas para monitorar as dependências entre as equipes de forma visual.
- Recomendar artefatos a serem criados para gerenciar as dependências.

O processo proposto é compatível com o *framework* Scrum. O objetivo é que as dependências possam ser monitoradas de forma visual, por este motivo foi proposto um quadro de tarefas de acesso das várias equipes. Os artefatos estão associados as histórias e ao *product backlog*, desta forma podem ser monitorados através de *post-it* que estão neste quadro de tarefas.

Conforme apresentado na seção 2.2, as dependências podem ser caracterizadas em três categorias: de conhecimento, de tarefas e de recursos (32). Como delimitação deste trabalho será tratado a dependência de tarefas.

Conforme Strode e Huff (32), uma dependência de tarefa ocorre quando uma tarefa deve ser concluída antes que outra tarefa possa continuar e isso afeta o andamento do projeto. Existem duas formas de dependência de tarefas:

- **Atividade** - uma situação em que uma atividade não pode prosseguir até que outra atividade seja concluída e isso afeta o andamento do projeto.
- **Processo de negócios** - uma situação em que um processo de negócios existente faz com que as atividades sejam executadas em uma determinada ordem e isso afeta o andamento do projeto.

4 RESULTADOS ALCANÇADOS

Neste capítulo são apresentados os resultados finais da pesquisa. Ele está subdividido em 2 seções, sendo a seção 4.1 que apresenta os resultados obtidos na etapa 1, que trata da revisão da literatura, a seção 4.2 apresenta a proposta do processo e dos artefatos.

4.1 PROBLEMAS E PRÁTICAS

Nesta seção é descrito os resultados alcançados pela revisão de literatura, onde buscou-se atender os seguintes objetivos específicos.

- Identificar os problemas de dependência vivenciados pelas empresas que utilizam o método Scrum em múltiplas equipes e quais estratégias utilizam para resolver ou amenizar os problemas de dependência.
- Apresentar as práticas utilizadas para realizar a reunião de planejamento da sprint que possibilite identificar e monitorar as dependências.

4.1.1 Problemas em decorrência das dependências vivenciados pelas empresas

Na literatura foram identificados vários problemas decorrentes da não identificação das dependências na fase de planejamento ou implicações das dependências no decorrer do desenvolvimento do projeto. Os principais problemas encontrados são:

- Redução do grau de liberdade para priorização do *product backlog*. As dependências devem ser levadas em consideração na hora de priorizar o *product backlog*, e quanto mais dependências existirem, menor será a liberdade na priorização dos itens (27, 16).
- Dificuldade em integrar o trabalho realizado. Com as equipes trabalhando em tarefas dependentes separadamente, muitas vezes na mesma funcionalidade, ao integrar o trabalho realizado, conflitos podem emergir (30).
- Conflitos entre equipes ao negociar as prioridades das tarefas com dependência. Como por exemplo, quando uma equipe depende de um componente com menor prioridade no *product backlog* de outra equipe (2, 5).
- Aumento da complexidade no gerenciamento do *product backlog* durante o processo de desenvolvimento. Devido a necessidade de adaptação às mudanças a cada sprint, as prioridades dos itens do podem mudar e isso pode afetar as demais equipes (2, 5).
- Estresse, queda na produtividade e aumento do número de *bugs*. Se as dependências não forem minimizadas, as equipes começarão a depender uma das outras para desenvolver

suas tarefas. Assim, desencadeando uma sequência de problemas, como estresse devido as dificuldades para lidar constantemente com as dependências, o aumento do número de *bugs* e consequentemente a queda da produtividade (36).

- Imprevisibilidade, pois com muitas equipes trabalhando juntos, compartilhando recursos e código é impossível antecipar todos problemas, surpresas, falhas e sucessos que serão encontrados durante o desenvolvimento de um recurso (2).

Para minimizar os impactos das dependências, as empresas buscam adotar algumas práticas que serão abordadas na seção 4.1.2.

4.1.2 Práticas utilizadas

Com o intuito de minimizar os impactos das dependências durante o desenvolvimento do produto, foram encontradas na literatura as práticas utilizadas pelas empresas. As práticas encontradas são:

- Dividir os requisitos em componentes menores para que possam ser executadas da forma mais independente possível (30, 27, 16, 17, 11, 5).
- Organizar equipes por funcionalidades (*Feature Teams*) também encontrado na literatura como equipes multifuncionais, que consiste em uma equipe que possui todas as competências necessárias para todo o desenvolvimento e lançamento de uma funcionalidade. A equipe possui a responsabilidade de desenvolver a funcionalidade de ponta a ponta, reduzindo assim a probabilidade do surgimento de dependências entre as demais equipes (16, 34, 18, 30).
- Revisões constantes dos impedimentos através de reuniões diárias, promovendo conversas cara a cara entre as equipes ou representantes de cada equipe que possuem tarefas dependentes (23, 11, 5, 2).
- Reuniões de espaço aberto (*Open Space Meeting*) para estimular colaboração e troca de conhecimento entre equipes, com o intuito de reduzir a necessidade de documentação, possibilitando que todos tenham conhecimento a respeito do *status* do projeto e dos objetivos comuns (2, 23).
- Exercícios para identificar dependências. Promoção de dinâmicas entre as equipes, para realizar a identificação das dependências durante a fase de planejamento. Através de abordagens que buscam gerar uma visão geral das conexões entre requisitos tornando as dependências transparentes desde o início (27, 23, 2).
- Promoção de visibilidade dos artefatos. Uma vez que os requisitos foram definidos, promover a visibilidade dos artefatos, como o *product backlog* ou um quadro de tarefas

contribuem muito para melhorar ainda mais a transparência e coordenar o trabalho entre várias equipes (27, 34, 23).

- Organização de uma equipe responsável por auxiliar na priorização, identificação e gerenciamento das dependências (5).

4.1.3 Considerações sobre a revisão da literatura

Além dos problemas e práticas, foram identificados outros aspectos importantes na revisão da literatura, que são descritos nesta seção.

4.1.3.1 Estrutura da equipe

A estruturação da equipe tem um grande impacto na minimização das dependências. Segundo Vlietland, Van Solingen e Vliet (34), idealmente as equipes Scrum devem cobrir a entrega de ponta a ponta, trabalhando de forma independente. Para atingir a independência cada equipe precisa ser organizado como uma equipe de funcionalidades(*Feature teams*) assim minimizando o impacto negativo das dependências (34).

Bick et al. (5) descreve a equipe central como sendo responsável pelo planejamento de alto nível do produto e pela facilitação da coordenação de atividades entre as várias equipes de desenvolvimento. A equipe central se reúne diariamente por uma hora para discutir tópicos atuais, bem como os planos para os próximos sprints. Uma vez por sprint, a equipe central organizava uma reunião de status e entrega com cada um dos PO das equipes de desenvolvimento separadamente.

A estratégia principal da equipe central para garantir uma boa coordenação entre as equipes está relacionada a impedir o surgimento de dependências. Para as dependências que não puderam ser eliminadas, mas foram identificadas, a equipe central busca mitigar o impacto durante a fase de planejamento.A equipe central repassa os casos não resolvidos para as equipes com tarefas dependentes entre si, promovendo comunicação e facilitando a colaboração entre elas para que possam chegar a um consenso para resolver as dependências (5).

Cabe destacar que todos os artigos identificados são estudos de caso de grandes empresas, observando a necessidade de pesquisas direcionadas para pequenas e médias empresas de desenvolvimento de software.

4.1.3.2 Documentação e artefatos

Nenhum dos artigos da revisão da literatura especificaram artefatos que registram as dependências entre requisitos do *product backlog* ou entre tarefas durante a iteração que poderiam facilitar o trabalho da equipe central e auxiliar na comunicação. Essa pesquisa procura resolver este problema e está descrito na seção 4.2.1.

4.1.3.3 Priorização do backlog

Em um sistema de múltiplas equipes ou de um única equipe, o *product backlog* é uma lista ordenada de tudo que é necessário para o desenvolvimento do produto. O grau de liberdade é a quantidade relativa de escolha em relação a um *product backlog* sem dependências. Para priorizar o *product backlog*, além de levar em consideração o valor para o cliente, as dependências também devem ser levadas em conta, para minimizar os impactos no decorrer do desenvolvimento do produto (27).

A priorização de tarefas se torna um desafio quando existe uma falta de planejamento efetivo. Quando surgem dependências não planejadas, as equipes precisam atualizar o *product backlog* para levar em conta as mudanças nos requisitos e planejamento. Essas alterações surgem, por exemplo, de novas solicitações de componentes de outras equipes que não foram planejadas antes e geralmente levam a conflitos no *product backlog* (30).

As dependências desempenham um papel crítico na priorização dos requisitos, pois deve haver um equilíbrio entre a geração de valor para o cliente e as dependências de requisitos. Atribuir foco para as dependências o mais cedo possível no projeto é de grande valia, devido ao seu potencial de minimizar o retrabalho. Caso sejam descobertas tardiamente, as dependências podem prejudicar a produtividade, pois uma pequena mudança em uma funcionalidade de uma equipe pode levar a uma cascata de retrabalho em tarefas de várias outras equipes. Então, a compreensão e identificação das dependências de requisitos o mais rápido possível é de suma importância para o sucesso de projetos ágeis (10).

4.1.3.4 Comunicação

Pikkarainen et al. (23) evidencia em sua pesquisa que os principais desafios no desenvolvimento de software se concentraram nos aspectos de comunicação.

Rahy e Bass (25) cita a comunicação entre equipes com dependências como um fator muito importante para o sucesso do projeto, pois a ausência de comunicação entre essas equipes podem trazer vários problemas, como atraso da sprint.

Segundo Uludag et al. (33) a expansão do desenvolvimento ágil envolve novos desafios de comunicação e coordenação. O uso do desenvolvimento ágil em grande escala contribuiu para o surgimento de muitos desafios de comunicação e coordenação. Surgindo a necessidade de criar novos papeis para possibilitar o gerenciamento das equipes.

Sekitoleko et al. (30) enfatiza a importância da comunicação para o sucesso de uma organização. Porem, apesar de ser fundamental, ela apresenta grandes desafios, não se limitando apenas na criação de meios para se transmitir a informação, mas garantir que essa mensagem seja recebida pelos membros das equipes.

4.1.4 Considerações da revisão da literatura

Através da análise dos artigos foi possível chegar as seguintes considerações que justificam e embasam a elaboração da proposta deste trabalho:

- Os estudos não citavam um método específico para tratar das dependências, chegando a conclusão que as abordagens utilizadas são bastante experimentais.
- Existem semelhanças entre os estudos, sendo os aspectos mais relevantes: a preocupação com a estrutura da equipe; a comunicação, seja ela visual ou cara a cara; a importância na priorização das tarefas; e identificação das dependências previamente.
- Limitações identificadas: falta de um processo formal; falta de artefatos de registro de dependências; e falta de artefatos que permitam o monitoramento das dependências no decorrer do processo.
- Não foram identificados estudos que apresentem a prática de implementação do método Nexus, *framework* desenvolvido pela Scrum.org para tentar resolver os problemas de dependência em múltiplas equipes Scrum.Acredita-se que seja pelo fato de ser relativamente novo, sendo o primeiro guia criado no ano 2015 e a sua última atualização em 2018.
- Não foram identificados estudos que tratam de pequenas e médias empresas que vivenciam problemas de dependência entre múltiplas equipes.

4.2 PROPOSTA DO PROCESSO

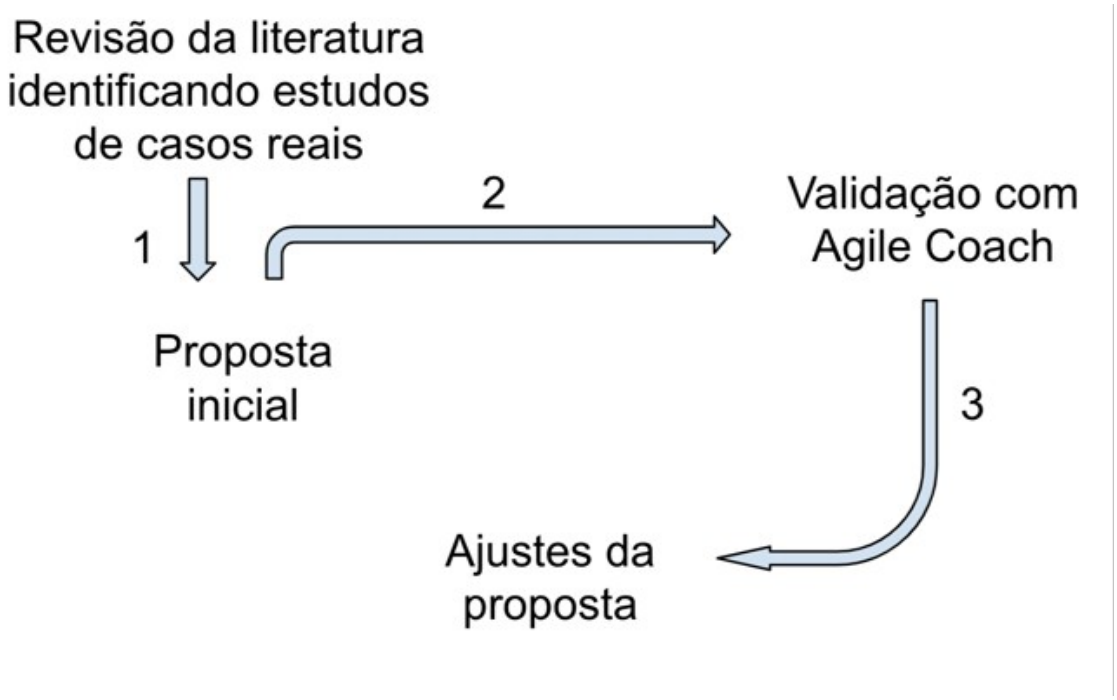
Nesta seção é apresentada a proposta desenvolvida para atender os seguintes objetivos específicos da pesquisa:

- Propor um processo de planejamento de dependência em múltiplas equipes Scrum que possibilite identificar e monitorar dependências.
- Propor um quadro de tarefas para monitorar as dependências entre as equipes de forma visual.
- Recomendar artefatos a serem criados para gerenciar as dependências.

A elaboração da proposta passou por várias fases de execução, conforme apresentado na figura 6. Primeiramente foi realizada uma revisão de literatura, onde foram identificados estudos de casos reais que abordavam o gerenciamento de dependências. A partir dessas informações foi desenvolvida uma proposta inicial onde as práticas encontradas na literatura foram adaptadas para atender a delimitação desta pesquisa. Após a formulação da proposta inicial, foi realizada uma entrevista com um Agile Coach que possui mais de 10 anos de experiência na indústria de

software, com vasta experiência na implantações de metodologias ágeis e reestruturações em empresas com diversos perfis, de startups a grandes empresas e órgãos públicos. Nesta entrevista buscou-se validar a proposta e identificar possíveis melhorias ou inconsistências. O principal ajuste realizado foi referente ao período de mapeamento das dependências, antes proposto a cada *release* e alterado para ser realizado a cada iteração. Os artefatos inicialmente tinham muitas informações que precisam ser preenchidas, e foram simplificados.

Figura 6 – Fases de desenvolvimento do processo proposto



Fonte: Autor

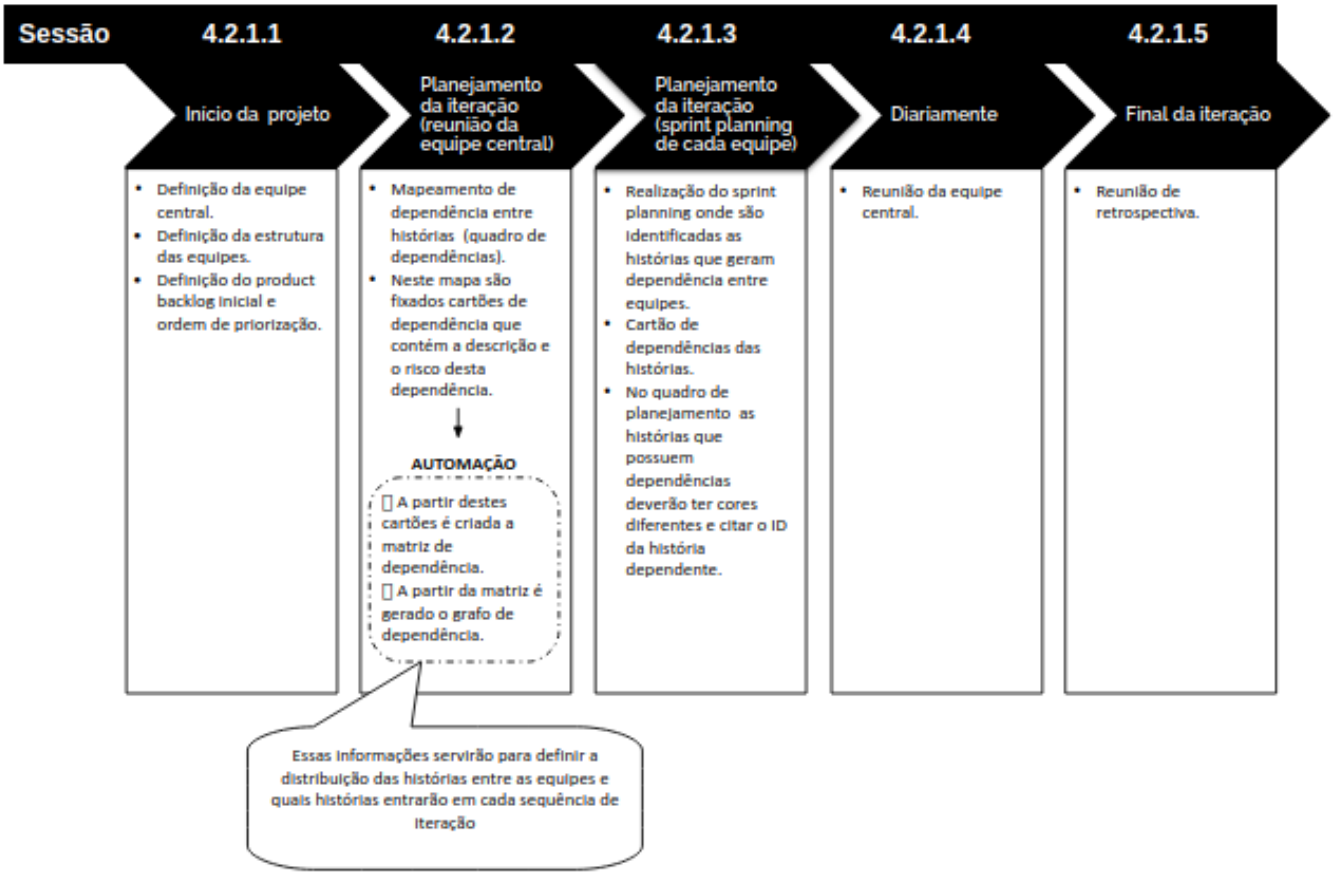
A figura 7 apresenta a proposta final desenvolvida.

4.2.1 Etapas do processo

Em empresas que possuem várias equipes ágeis trabalhando no desenvolvimento de um mesmo produto, com requisitos que se relacionam gerando dependência de projeto e nos seus artefatos, que trabalham na mesma versão com recursos e códigos compartilhados, é praticamente impossível prever todos os problemas, surpresas, mudanças, falhas e sucessos que as equipes encontrarão durante o desenvolvimento de software. Essa imprevisibilidade é uma das razões pelas quais o gerenciamento de dependências em múltiplas equipes é difícil (2).

Levando em consideração essas dificuldades, a proposta deste trabalho está subdividida em várias etapas de planejamento, identificação e gerenciamento das dependências. A figura 7 representa a visão geral da proposta desenvolvida, apresentando a sequência de fases, a visão geral das atividades que deverão ser realizadas em cada fase, e qual a seção deste trabalho que está descrita.

Figura 7 – Etapas da proposta do processo



Fonte: Autor

4.2.1.1 Início do projeto

No início de um projeto recomenda-se realizar as seguintes atividades: definição da estrutura das equipes; definição da equipe central; definição do *product backlog* inicial e definir ordem de priorização.

4.2.1.1.1 Definição da estrutura das equipes

O foco deste trabalho é pequenas e médias empresas que utilizam Scrum e estão em um processo de crescimento. Baseado nestas características de empresas, se propõe que os papéis e a estrutura das equipes sejam herdadas do Scrum para que ofereçam o mínimo de impacto possível durante o processo (34, 7).

Como ressaltado na seção 4.1.3, um dos pontos mais importantes para minimizar os impactos das dependências é a organização das equipes, podendo ser organizadas por diferentes critérios dependendo da realidade de cada produto ou recursos que a empresa possui. As equipes podem ser organizadas em grupos com base no conjunto de tecnologias de desenvolvimento, personas, recursos ou componentes (7).

A melhor estratégia é aquela que permite que cada equipe trabalhe de forma relativamente independente, em paralelo, enquanto integra continuamente seu trabalho. O ideal seria compôr equipes totalmente independentes e multifuncionais, não dependendo de nenhum outro agente para desenvolver as tarefas, onde os membros da equipe possam realizar qualquer tipo de trabalho, pois quanto mais específicas as habilidades dos integrantes da equipe, mais difícil é organizar o fluxo de trabalho para que ele seja mais fluido (7).

No entanto, esta é uma situação complexa para as empresas de grande porte e ainda muito mais complexo para empresas de pequeno e médio porte, devido a escassez de recursos humanos. Nos casos em que certas habilidades técnicas específicas são escassas, o pareamento de desenvolvedores fornece uma maneira de compartilhar conhecimento e ajuda a ampliar a base de conhecimento técnico e até comercial da equipe (7).

4.2.1.1.2 Definição da equipe central

Após a definição da estrutura das equipes, outra prática recomendada é a criação de uma equipe central, que consiste em uma equipe composta por profissionais capacitados que possam auxiliar no gerenciamento das dependências e disseminação das práticas no decorrer do processo (7, 4).

A equipe central é responsável por garantir que as equipes sejam capazes de realizar as integrações por conta própria, através de práticas ou servindo como *coaches* e consultores para auxiliar na coordenação entre as equipes (7).

A composição da equipe central normalmente consiste no PO e especialistas, membros das equipes de desenvolvimento, podendo ser o Scrum Master ou desenvolvedor. Os integrantes podem ser alterados no decorrer das sprints, cada integrante pode ser substituído por outra que seja mais capaz de representar a equipe e possa contribuir mais para resolver os problemas a medida que vão surgindo no decorrer do processo (7).

4.2.1.1.3 Definição do product backlog inicial e ordem de priorização

Em empresas de pequeno e médio porte a criação e priorização do *product backlog* inicial é realizada pelo PO no início do projeto, utilizando como critério principal a ordenação pela ordem de valor ao negócio (28).

Este *product backlog* é apenas uma visão de alto nível das funcionalidades a serem desenovolvidas, e ainda precisa passar por um refinamento adequado, mas é fundamental para nortear as etapas seguintes apresentadas na seção 4.2.1.2.

4.2.1.2 Planejamento da iteração - reunião da equipe central

Antes do início da sprint é realizado um encontro entre os membros da equipe central, onde são realizadas algumas atividades e também são criados artefatos para auxiliar na do-

cumentação e monitoramento das dependências no decorrer do processo. Neste momento do processo o *product backlog* já deve estar definido e priorizado. A priorização das funcionalidades é uma informação importante para as próximas etapas do processo que será descrita na seção 4.2.1.2.1.

4.2.1.2.1 Mapeamento das dependências entre histórias

O mapeamento das dependências entre histórias deve ocorrer após ter o *product backlog* já priorizado. Na reunião de planejamento da sprint, a equipe de integração se reúne e refina o *product backlog*, identificando as histórias que possuem dependências entre si (7).

Neste momento é proposto dois novos artefatos: quadro de histórias e cartão de dependência.

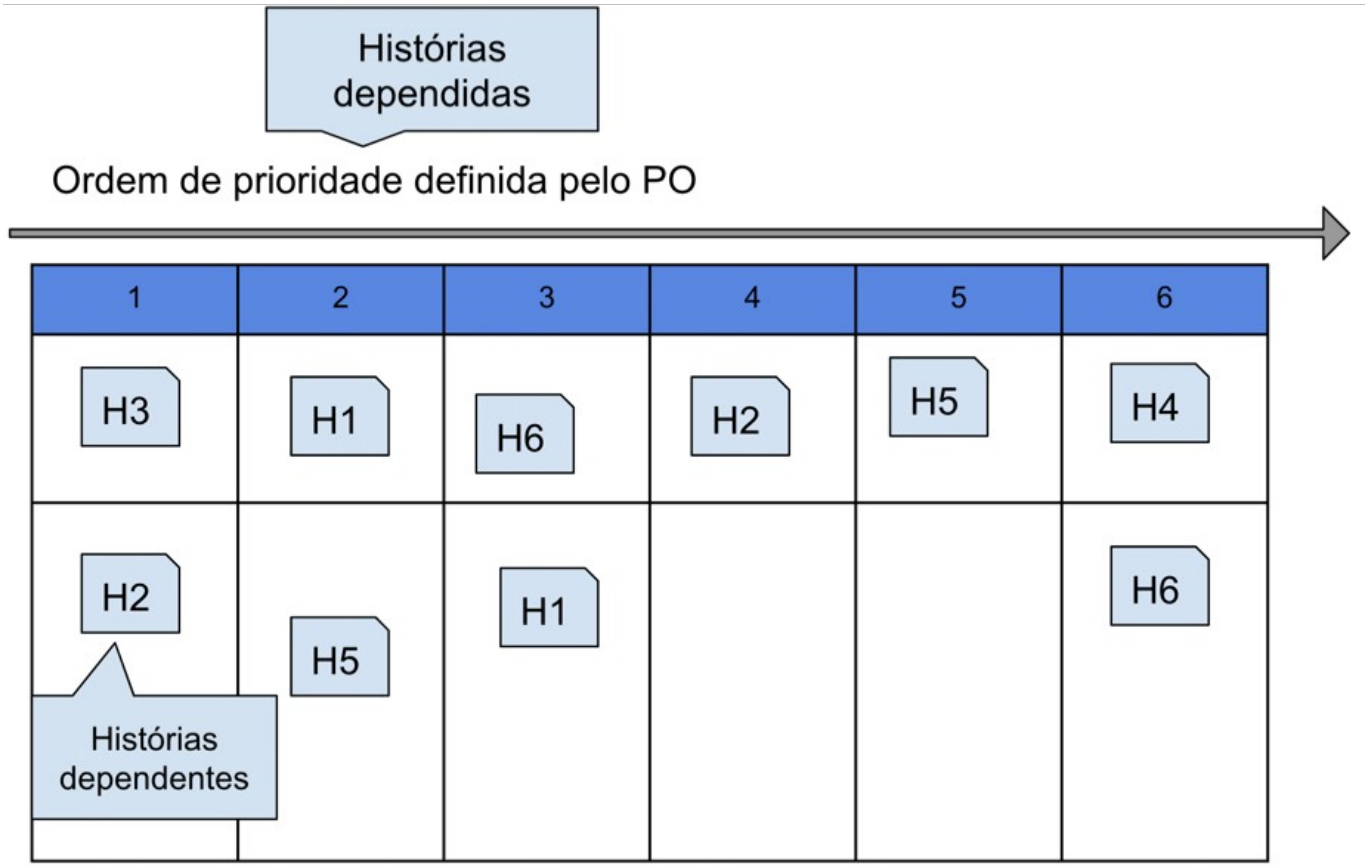
O quadro de histórias possui várias colunas. As colunas representam as histórias que compõe o *product backlog*. Na primeira coluna deve ser colado um *post-it* com a descrição da história com maior prioridade, na segunda coluna a próxima na ordem de prioridade, e assim sucessivamente. Havendo muitas histórias, recomenda-se colar *post-its* apenas das histórias da sprint atual e da sprint subsequente. Conforme descrito por Cohn (8), o planejamento das iterações deve ser em um período onde é possível visualizar, planejar e controlar todo desenvolvimento (8).

Após a distribuição das histórias nas respectivas colunas, deve-se discutir entre os membros quais histórias possuem dependências. No primeiro momento é necessário identificar quais histórias são dependentes entre si.

A construção do quadro de mapeamento de dependências é uma tarefa colaborativa, pois instiga a conversa cara a cara, e além do mapeamento e identificação das dependências, é uma excelente oportunidade para aproximar as equipes, desenvolvendo aspectos como a comunicação e a colaboração entre as equipes.

No segundo momento, para cada dependência é recomendado criar um cartão de dependência. O qual é descrito na seção 4.2.1.2.2. A figura 8 apresenta a proposta do quadro de mapeamento das dependências entre histórias.

Figura 8 – Quadro de mapeamento das dependências entre histórias



Fonte: Autor

O próximo passo é compreender as consequências que essas dependências podem gerar, descrito na seção 4.2.1.2.2.

4.2.1.2.2 Compreender o tipo e o grau de cada dependência

Para identificar e documentar as dependências entre as histórias é proposto a utilização de um cartão de dependência.

O cartão de dependência, apresentado na figura 9 deverá conter a identificação da história dependente e da história dependida. Também deverá conter o tipo da dependência, seja ela de sequência de execução ou de interdependência de componentes e o grau de risco relacionada a dependência. O tipo de dependência servirá também para a equipe analisar como será o impacto disso no decorrer da sprint. Essa análise servirá para chegarem a conclusão do grau de risco.

O cartão deve ser a memória da equipe sobre a reunião de análise das dependências.

Figura 9 – Cartão de dependência

História dependida 1	**História dependente 2**
Como... Preciso de... Para...	Como... Preciso de... Para...
Tipo de dependência: ( ) Dependência de sequência de execução. ( ) Componentes interdependentes.	
Grau de risco: ( ) Grau 1 - Baixo - possuem dependência em uma tarefa isolada ( ) Grau 2 - Alto - se a dependência não for resolvida a história não poderá ser concluída	
Descrição:	

Fonte: Autor

Cada dependência entre histórias deverá ser discutida individualmente pela equipe central. Neste trabalho propomos que as dependências sejam classificadas por tipo e grau de risco. O tipo de dependência se refere a forma de dependência que existe entre as histórias. Os tipos de dependência propostos são:

- Dependência de sequência de execução (S): uma deve ocorrer antes/depois da outra (ordem de execução) ou que produz um componente que irá ser obrigatoriamente desenvolvido antes, pois outra equipe irá precisar daquele componente pronto para desenvolver suas tarefas.

- Componentes Interdependentes (I): possuem interdependência de componentes específicos (arquitetura, bibliotecas, funções, entre outras funções) que serão reaproveitadas por ambas.

A segunda informação a ser discutida é o grau de risco de cada dependência. O tipo de dependência é muito importante para auxiliar na análise e definição do grau de risco da dependência.

A equipe deverá discutir e chegar a um consenso sobre o grau de risco que aquela dependência irá gerar caso não seja concluída. Os tipos de grau de risco propostos são:

- Grau 1 – Baixa dependência: possuem dependência em uma tarefa isolada.
- Grau 2 – Alta dependência: se a dependência não for resolvida a história não poderá ser concluída na sua totalidade.

As informações do cartão de dependência são muito importantes, pois além de documentar as decisões tomadas pela equipe durante a reunião de análise das dependências são fundamentais para o próximo passo do processo, que é a criação da matriz de dependências para gerar o grafo de dependências entre as histórias. Descrito na próxima seção.

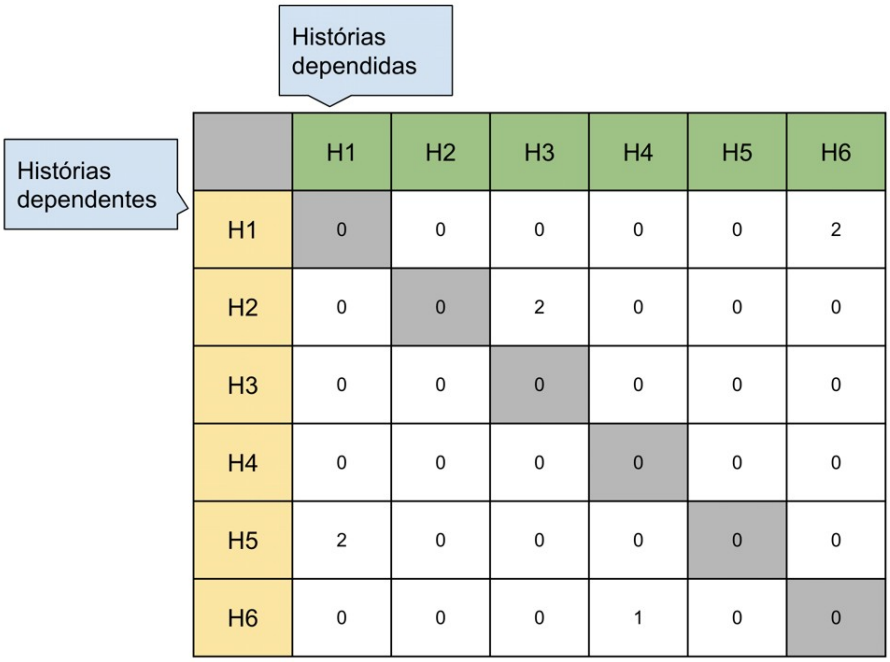
4.2.1.2.3 Criar matriz e grafo de dependências

A partir das dependências já identificadas e classificadas, pode ser criada a matriz de dependências, que possibilitará a geração do grafo. Este grafo irá auxiliar na identificação de possíveis ciclos de dependências entre histórias e casos de dependências críticos, onde existe uma grande concentração de dependências.

Tanto a criação da matriz de rastreabilidade quanto o grafo de dependência apresentado nesta seção, são processos que deve ser automatizados, ou seja, deve-se criar uma ferramenta de software para automatizar estas tarefas. A fonte base para isso são os cartões de dependência, apresentado na figura 9. Devido a limitação de tempo do TCC 2 está sendo proposto o desenvolvimento da ferramenta como trabalho futuro e apresentado nas considerações finais do trabalho.

Na figura 10 podemos visualizar a matriz gerada. Para cada relação entre história dependente e história dependida, é atribuído o número 0, caso não exista dependência. Caso possua dependência então é atribuído o grau de risco dessa dependência, como informado no cartão de dependência, sendo ele 1 para baixo risco ou 2 para alto risco.

Figura 10 – Matriz de dependências



	H1	H2	H3	H4	H5	H6
H1	0	0	0	0	0	2
H2	0	0	2	0	0	0
H3	0	0	0	0	0	0
H4	0	0	0	0	0	0
H5	2	0	0	0	0	0
H6	0	0	0	1	0	0

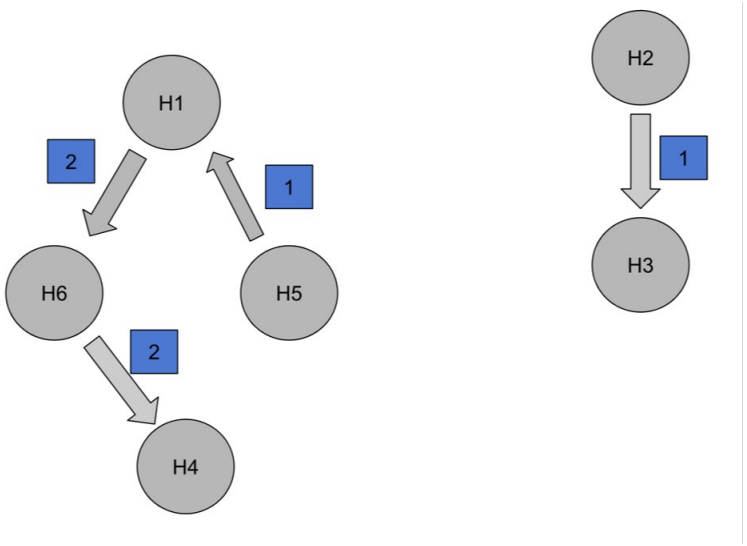
Fonte: Autor

O grau de risco de cada dependência apresentado na figura 11 é explícito no peso da aresta do grafo.

Um grafo é uma representação visual de um determinado conjunto de dados e da ligação existente entre alguns dos elementos desse conjunto. Elementos que atendem à relação imaginada são simbolicamente unidos através de um traço denominado aresta. O modelo possui uma interpretação gráfica muito confortável, possibilitando a identificação visual dos pontos com grande concentração de ligações (15).

Através do grafo é possível identificar as histórias com mais dependências e possíveis ciclos, que vão trazer problemas durante o planejamento da sprint. No grafo a flecha aponta da história dependente, para a história dependida. Ou seja, aponta para a história que precisar ser concluída antes que ela possa ser desenvolvida.

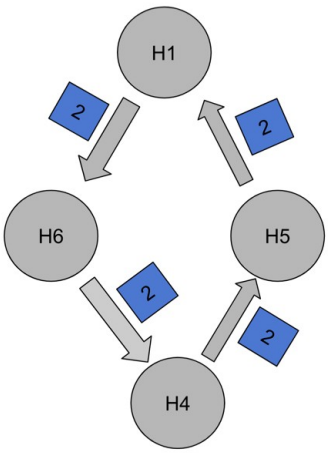
Figura 11 – Grafo de dependências



Fonte: Autor

A transparência obtida através do grafo de dependências possibilita a identificação de sequencias de tarefas dependentes, que podem trazer mais complexidade na priorização das tarefas ou a identificação de ciclos de dependências, como o exemplo da figura 12, que devem ser identificados e eliminados, através de uma revisão das histórias escritas pelo PO, tornando-as mais independentes.

Figura 12 – Ciclo entre histórias dependentes



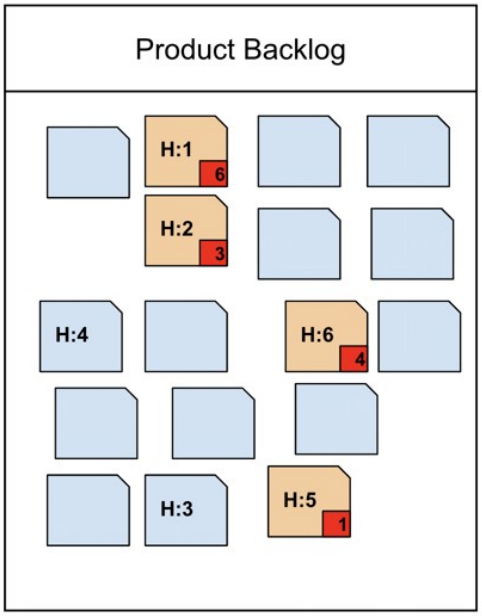
Fonte: Autor

A visualização que o grafo de dependências proporciona, possibilita um refinamento eficiente, resultando em um *product backlog* devidamente priorizado e com as dependências identificadas e destacadas como mostrado na figura 13.

As histórias que possuem dependências, são destacadas com uma cor diferente e uma

etiqueta no seu canto direito inferior, que indicam qual é a história dependida, ou seja a história que precisa ser finalizada para que ela possa estar apta a ser desenvolvida.

Figura 13 – Product backlog

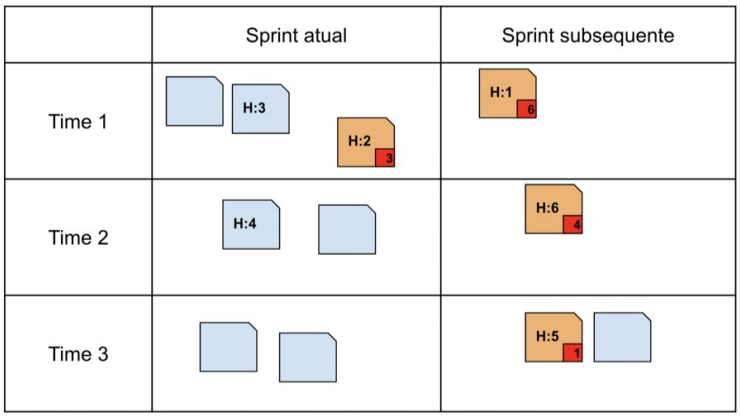


Fonte: Adaptado de Bittner et al. (7)

Com o *product backlog* definido, cada equipe seleciona seu trabalho para a sprint subsequente, buscando minimizar os riscos das dependências atribuindo um grupo de histórias dependentes a uma mesma equipe, se possível.

Após realizada a seleção das histórias, deve ser criado um quadro visual seguindo o modelo na figura 14, onde é exibido o planejamento para a sprint atual e para a sprint subsequente. Esta etapa é crucial para a minimização e o gerenciamento das dependências, pois as dependências podem ser tratadas a níveis de sprint e de equipe, organizando as histórias dependentes em uma equipe ou atribuindo as histórias para serem desenvolvidas em sprints diferentes, minimizando assim o impacto das dependências na sprint atual.

Figura 14 – Quadro de planejamento da sprint



Fonte: Adaptado de Bittner et al. (7)

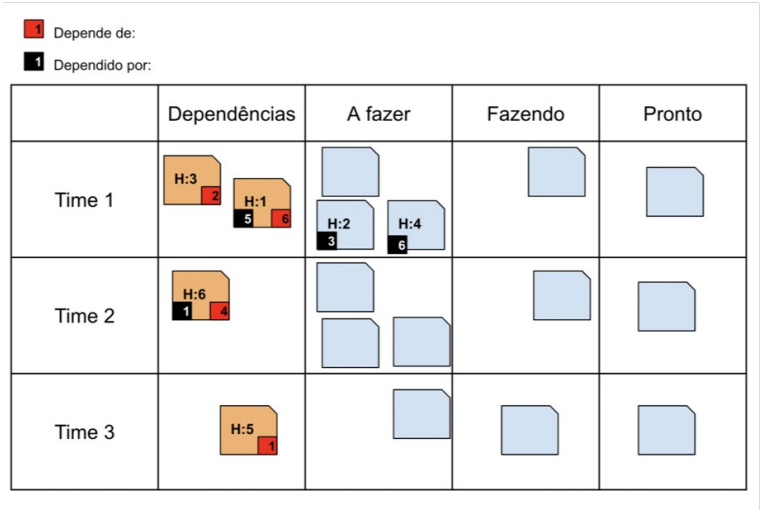
Com as dependências identificadas e as histórias devidamente selecionadas, cada equipe parte para o seu planejamento individual, descrito na próxima seção 4.2.1.3. A equipe central é responsável por promover a comunicação entre as equipes que possuem dependências entre elas, nas demais histórias a equipe possui independência para planejar e desenvolver as suas respectivas histórias.

4.2.1.3 Planejamento da iteração - sprint planning de cada equipe

A partir do planejamento e das histórias definidas para cada equipe, é criado um quadro compartilhado, onde todas as histórias de todas as equipes estão dispostas e com as dependências identificadas.

O quadro de tarefas possibilita uma boa gestão visual por parte de todos os interessados. As dependências são destacadas possibilitando que as equipes antecipem a comunicação entre eles com o intuito de resolver as tarefas dependentes o mais rápido possível, assim minimizando os impactos causados pelas dependências durante a sprint. O modelo do quadro é apresentado na figura 15.

Figura 15 – Quadro de tarefas



Fonte: Adaptado de Bittner et al. (7)

Além da etiqueta vermelha informando qual é a história dependente, também é adicionado uma etiqueta preta, no canto esquerdo inferior, caso a história possua uma tarefa que dependa dela para ser executada.

As histórias com dependências são adicionadas a coluna "Dependências" e somente são movidas para a coluna "A fazer" quando a sua dependência for resolvida, assim minimizando os impactos que essa dependência pode causar durante a sprint.

O objetivo é tornar o processo de identificação simples, com uma gestão visual eficiente e com foco na visualização das dependências.

4.2.1.4 Gerenciamento diário

Como mencionado na seção 4.2.1.1.2 os membros da equipe central podem variar de acordo com os problemas encontrados. Na reunião diária, é muito importante a presença das pessoas que realmente vão realizar o trabalho, sejam elas Scrum Master ou membros da equipe de desenvolvimento, pois são as pessoas que realmente vão contribuir para que o problema seja resolvido (7).

A reunião diária da equipe central tem o objetivo de tornar as dependências transparentes entre as equipes, para que assim possam tomar as decisões corretas de como lidar com os desafios encontrados. O objetivo não é que a equipe central resolva os problemas de dependências, mas que as próprias equipes possam se organizar e resolver entre si (7).

Outra recomendação é que a reunião diária da equipe central seja realizada antes das reuniões diárias das equipes Scrum. Assim as equipes primeiramente enxergam os problemas de integração e posteriormente vão para a reunião diária da equipe priorizando a resolução dos problemas. A reunião diária da equipe central ajuda as equipes a se concentrarem primeiro nos problemas de integração, incluindo dependências (7).

A reunião diária da equipe central ajuda na inspeção, na adaptação do processo e no gerenciamento de dependências, fornecendo transparência entre equipes (7).

4.2.1.5 Final da iteração

Após a retrospectiva de cada equipe, a equipe central se reúne para discutir quais práticas foram bem-sucedidas e encontrar maneiras de melhorar o processo.

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo propor um processo e artefatos para gerenciar dependências em equipes que utilizam Scrum em escala. Para que os objetivos deste trabalho fossem alcançados foi estruturado a metodologia de pesquisa apresentada em 2 etapas: (i) Revisão da literatura; (ii) Proposta do processo e da documentação;

Na etapa 1 procurou-se atingir os seguintes objetivos específicos: (a) Identificar os problemas de dependência vivenciados pelas empresas que utilizam o método Scrum em múltiplas equipes e quais estratégias utilizam para resolver ou amenizar os problemas de dependência; (b) Apresentar as práticas utilizadas para realizar a reunião de planejamento da sprint que possibilite identificar e monitorar as dependências.

Os estudos encontrados durante a revisão bibliográfica proporcionaram uma visão dos problemas vivenciados por empresas que adotaram o ágil em escala, e quais as práticas utilizadas para realizar o planejamento da sprint que buscam identificar e monitorar as dependências. Os principais problemas vivenciados entre as empresas são: redução do grau de liberdade para priorização dos itens do *product backlog*; aumento da complexidade no gerenciamento do *product backlog*; conflitos entre equipes.Para resolver os problemas, as práticas mais utilizadas são: a organização das equipes por funcionalidades; revisões constantes dos impedimentos através de reuniões diárias; promoção de visibilidade dos artefatos. Através da revisão foi possível validar o problema, além de observar as seguintes lacunas de pesquisa: falta de um processo formal; falta de artefatos de registro de dependências; não foram identificados estudos que apresentem a prática do método Nexus;

Na etapa 2 foi proposto o processo e a documentação para atender aos objetivos específicos: (a) Propor um processo de planejamento de dependência em múltiplas equipes Scrum que possibilite identificar e monitorar dependências; (b) Propor um quadro de tarefas para monitorar as dependências entre as equipes de forma visual; (c) Recomendar artefatos a serem criados para gerenciar as dependências;

A partir da revisão bibliográfica foi possível organizar as práticas encontradas, a fim de agrupar essas práticas em uma proposta de processo de planejamento de sprint. No processo proposto foram propostos documentos e artefatos, com o intuito de além de documentar, prover transparência para o processo.

Com o processo proposto, as empresas poderão ter as seguintes vantagens: melhora na produtividade e redução do número de *bugs*; redução da complexidade no gerenciamento do *product backlog*; minimização dos riscos relacionados aos atrasos em sprints.

Os artefatos criados possibilitam promover transparência e visibilidade, possibilitando a antecipação e mitigação dos problemas gerados pelas dependências e também prover documentação, a fim de auxiliar no monitoramento das dependências durante o processo.

A automação do processo é fundamental, por isso recomenda-se, a partir do processo e dos artefatos criados, desenvolver uma ferramenta para automação do processo.

5.1 TRABALHOS FUTUROS

Com o objetivo de dar continuidade neste trabalho, são apresentados os seguintes trabalhos futuros:

- Desenvolver ferramenta que possibilite automatizar o gerenciamento de dependências através da criação automatizada da matriz de dependências, da geração do grafo de dependências e do monitoramento no decorrer da iteração.
- Realizar pesquisa-ação em ambiente real para melhorar e validar o processo.

REFERÊNCIAS

1 AMBLER, Scott W. **The Non-Existent Software Crisis: Debunking the Chaos Report**. 2014. Disponível em: <https://www.drdobbs.com/architecture- and- design/the-non-existent-software-crisis-debunki/240165910 >. Acesso em: 21 nov. 2019.

2 BABINET, Eric; RAMANATHAN, Rajani. Dependency management in a large agile environment. In: IEEE. AGILE 2008 Conference. [S.l.: s.n.], 2008. p. 401–406.

3 BECK, Kent et al. Manifesto for agile software development, 2001.

4 BICK, Saskia; SCHEERER, Alexander; SPOHRER, Kai. Inter-team coordination in large agile software development settings: Five ways of practicing agile at scale. In: ACM. PROCEEDINGS of the Scientific Workshop Proceedings of XP2016. [S.l.: s.n.], 2016. p. 4.

5 BICK, Saskia et al. Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings. **IEEE Transactions on Software Engineering**, IEEE, v. 44, n. 10, p. 932–950, 2017.

6 BITTNER, Kurt. **Scaling Scrum with Nexus and Scrum Studio**. 2018. Disponível em: <https : / / www . scrum . org / resources / blog / scaling - scrum - nexus - and - scrum-studio>. Acesso em: 11 nov. 2019.

7 BITTNER, Kurt et al. **The Nexus Framework for Scaling Scrum: Continuously Delivering an Integrated Product with Multiple Scrum Teams**. [S.l.]: Addison-Wesley Professional, 2017.

8 COHN, Mike. **Agile estimating and planning**. [S.l.]: Pearson Education, 2005.

9 CONBOY, Kieran; CARROLL, Noel. Implementing Large-Scale Agile Frameworks: Challenges and Recommendations. **IEEE Software**, IEEE, v. 36, n. 2, p. 44–50, 2019.

10 DANEVA, Maya et al. Agile requirements prioritization in large-scale outsourced system projects: An empirical study. **Journal of systems and software**, Elsevier, v. 86, n. 5, p. 1333–1353, 2013.

11 DIKERT, Kim; PAASIVAARA, Maria; LASSENIUS, Casper. Challenges and success factors for large-scale agile transformations: A systematic literature review. **Journal of Systems and Software**, Elsevier, v. 119, p. 87–108, 2016.

12 DINGSØYR, Torgeir; FALESSI, Davide; POWER, Ken. Agile Development at Scale: The Next Frontier. **CoRR**, abs/1901.00324, 2019. arXiv: 1901.00324. Disponível em: <http://arxiv.org/abs/1901.00324 >.

13 FOWLER, Frederik M. The Scrum Team. In: NAVIGATING Hybrid Scrum Environments. [S.l.]: Springer, 2019. p. 25–30.

14 GIL, Antonio Carlos. **Estudo de caso**. [S.l.]: Atlas, 2009.

15 GOLDBARG, Marco Cesar; LUNA, Henrique Pacca L. Otimização combinatória e programação linear. **Editora Campus**, v. 2, 2000.

16 HEIKKIÄ, Ville T et al. Operational release planning in large-scale Scrum with multiple stakeholders–A longitudinal case study at F-Secure Corporation. **Information and Software Technology**, Elsevier, v. 57, p. 116–140, 2015.

17 MARTAKIS, Aias; DANEVA, Maya. Handling requirements dependencies in agile projects: A focus group with agile software development practitioners. In: IEEE. IEEE 7th International Conference on Research Challenges in Information Science (RCIS). [S.l.: s.n.], 2013. p. 1–11.

18 MOE, Nils Brede et al. Networking in a large-scale distributed agile project. In: ACM. PROCEEDINGS of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. [S.l.: s.n.], 2014. p. 12.

19 ONE, Version. **13th Annual State of Agile Report**. 2019. Disponível em: <https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508>. Acesso em: 10 dez. 2019.

20 _____. **7th Annual State of Agile Report**. 2013. Disponível em: <https://www.stateofagile.com/#ufh-i-338592786-7th-annual-state-of-agile-report/473508>. Acesso em: 21 nov. 2019.

21 PAASIVAARA, Maria; LASSENIUS, Casper. Scaling scrum in a large distributed project. In: IEEE. 2011 International Symposium on Empirical Software Engineering and Measurement. [S.l.: s.n.], 2011. p. 363–367.

22 PAASIVAARA, Maria; LASSENIUS, Casper; HEIKKILÄ, Ville T. Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work? In: ACM. PROCEEDINGS of the ACM-IEEE international symposium on Empirical software engineering and measurement. [S.l.: s.n.], 2012. p. 235–238.

23 PIKKARAINEN, Minna et al. The impact of agile practices on communication in software development. **Empirical Software Engineering**, Springer, v. 13, n. 3, p. 303–337, 2008.

24 PRESSMAN, Roger S. **Software Engineering Eight Edition**. [S.l.]: New York: McGraw-Hill, 2015.

25 RAHY, Scarlet; BASS, Julian. Information flows at inter-team boundaries in agile information systems development. In: SPRINGER. EUROPEAN, Mediterranean, and Middle Eastern Conference on Information Systems. [S.l.: s.n.], 2018. p. 489–502.

26 RODRÍGUEZ, Daniel et al. Empirical findings on team size and productivity in software development. **Journal of Systems and Software**, Elsevier, v. 85, n. 3, p. 562–570, 2012.

27 SCHEERER, Alexander et al. The effects of team backlog dependencies on agile multiteam systems: A graph theoretical approach. In: IEEE. 2015 48th Hawaii International Conference on System Sciences. [S.l.: s.n.], 2015. p. 5124–5132.

28 SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum**. Tradução: Fábio Cruz e Eduardo Rodrigues Sucena. [S.l.: s.n.], 2017. 19 p. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf>. Acesso em: 15 jun. 2019.

29 SCRUM.ORG, Ken Schwaber e. **Guia do Nexus**. Tradução: Fábio Cruz e Eduardo Rodrigues Sucena. [S.l.: s.n.], 2018. 14 p. Disponível em: <https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-01/2018-Nexus-Guide-Portuguese-Brazillian.pdf>. Acesso em: 15 jun. 2019.

30 SEKITOLEKO, Nelson et al. Technical dependency challenges in large-scale agile software development. In: SPRINGER. INTERNATIONAL Conference on Agile Software Development. [S.l.: s.n.], 2014. p. 46–61.

31 SOMMERVILLE, Ian. Software engineering 9th Edition. **ISBN-10**, v. 137035152, 2011.

32 STRODE, Diane E; HUFF, Sid L. A taxonomy of dependencies in agile software development. In: ACIS. ACIS 2012: Location, location, location: Proceedings of the 23rd Australasian Conference on Information Systems 2012. [S.l.: s.n.], 2012. p. 1–10.

33 ULUDAG, Ömer et al. Identifying and structuring challenges in large-scale agile development based on a structured literature review. In: IEEE. 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC). [S.l.: s.n.], 2018. p. 191–197.

34 VLIETLAND, Jan; VAN SOLINGEN, Rini; VLIET, Hans van. Aligning codependent Scrum teams to enable fast business value delivery: A governance framework and set of intervention actions. **Journal of Systems and Software**, Elsevier, v. 113, p. 418–429, 2016.

35 WHEELAN, Susan A. Group size, group development, and group productivity. **Small Group Research**, Sage Publications Sage CA: Los Angeles, CA, v. 40, n. 2, p. 247–262, 2009.

36 YACOUB, Maha Khaled; MOSTAFA, Mostafa Abdel Athim; FARID, Ahmed Bahaa. A New Approach for Distributed Software Engineering Teams Based on Kanban Method for Reducing Dependency. **JSW**, v. 11, n. 12, p. 1231–1241, 2016.