

Desafios na Implementação do Scrum: um Estudo de Caso Sobre a Utilização da Metodologia Ágil em uma Empresa Desenvolvedora de Software

Daniel Fazza Dielle, Marco Antônio Pereira Araújo

Bacharelado em Sistemas de Informação
Centro de Ensino Superior de Juiz de Fora (CES/JF) – Juiz de Fora – MG – Brasil
fazzadielle@gmail.com, marco.araujo@pucminas.cesjf.br

***Abstract.** This paper has the purpose on contributing to surpass the challenges observed when applying Scrum and, for this, it accomplishes a case study from historical data provided by a partner software development company, who adopted this practice. It details the particularities of this adoption and shows the management results obtained at various moments of its trajectory, pointing out which probable events and factors they are related to. Thus, possible causal relationships to the difficulties encountered are presented and, as conclusion, it suggests techniques to avoid them, and methods for the evaluation of the Scrum in course.*

***Resumo.** Este artigo tem como objetivo contribuir para a superação de desafios observados na aplicação do Scrum e, para tal, realiza um estudo de caso a partir de dados históricos cedidos por uma empresa parceira, desenvolvedora de software, que adotou essa prática. Detalha as particularidades dessa adoção e mostra os resultados gerenciais obtidos em momentos diferentes de sua trajetória, apontando os prováveis eventos e fatores aos quais estão relacionados. Assim, são apresentadas possíveis relações de causalidade para as dificuldades encontradas e, como conclusão, são sugeridas técnicas para evitá-las, e métodos para avaliação do Scrum em curso.*

1. Introdução

Muitas empresas desenvolvedoras de *software* estão habituadas a lidar diariamente com atrasos no cronograma e custos inesperados. A constante mudança durante o ciclo de vida do *software* é um evento comum que tem mostrado a necessidade de um controle sobre os processos de desenvolvimento que a leve em consideração. Nesse contexto, este trabalho expõe a aplicabilidade do *Scrum* como uma metodologia ágil capaz de atender a esse requisito e que pretende trazer às empresas o melhor resultado no menor tempo possível. Contudo, há desafios na transição para o *Scrum*, que estão relacionados à necessidade de mudanças na cultura organizacional (COHN 2011) e ao esforço de mensurar corretamente o andamento do mesmo (SCRUM ALLIANCE 2015).

No que diz respeito às mudanças na cultura organizacional, o desafio consiste em conseguir fazer as pessoas implementarem o *Scrum* juntas, independentemente de seu nível hierárquico. O *Scrum* depende de colaboração mútua e, por isso, a rotina de

trabalho não pode ser alterada de forma imperativa pelas pessoas de maior *status* para as de menor *status*, e nem mesmo o contrário (COHN 2011), ou seja, não é interessante que algumas pessoas decidam por conta própria que certas ações são necessárias e, então, passem a pressionar seus superiores para que as coisas aconteçam. Uma empresa que está estruturada dessa forma precisa se reorganizar culturalmente, caso contrário, enfrentará resistência interna e o sucesso do *Scrum* estará comprometido.

O desafio técnico de medir corretamente o *Scrum*, por sua vez, está diretamente relacionado ao desafio de mudar a cultura de uma empresa. Uma pesquisa recente (SCRUM ALLIANCE 2015) mostra que empresas que implementaram ou estão implementando essa metodologia possuem uma taxa média de sucesso nos seus projetos de apenas 62%. Apesar disso, na opinião de 87% dessas mesmas empresas, a qualidade do trabalho realizado está melhorando com o *Scrum*. A baixa taxa média de sucesso dos projetos parece estar relacionada com o fato de que 71% das empresas pesquisadas acreditarem que o *Scrum* causa uma tensão com outras partes da organização que não se utilizam dele, sendo que o desafio mais comum, relatado em 52% dos casos, é exatamente esse: a identificação e mensuração do sucesso do *Scrum*. É dedutível, portanto, que as empresas declaram melhora com a aplicação do *Scrum*, mas essa declaração é feita com pouco embasamento técnico e não convence. Isso torna difícil para os setores onde a mudança organizacional será mais profunda reconhecer as vantagens do *Scrum*, o que prejudica as taxas de sucesso dos projetos.

As dificuldades citadas anteriormente e comumente encontradas na aplicação do *Scrum* (COHN 2011) fazem com que muitos não cheguem ao menos a tentar. Diante desse cenário, a motivação desse artigo é contribuir na redução do esforço necessário para implementar a metodologia com sucesso. Por isso, a análise atenta do cotidiano do *Scrum* é tão importante: com ela, é possível provar para os envolvidos que a mudança vale a pena e garantir que a equipe não está se desviando das práticas ágeis. Neste trabalho, os números gerados pelos *sprints* e a observação das circunstâncias em que eles ocorreram compõem o método aplicado para avaliação do *Scrum*. Os resultados que foram gerados permitirão à empresa superar o desafio de mensurá-lo e, consequentemente, favorecerão a consolidação de uma cultura organizacional que o apoie.

A partir de dados reais (cuja obtenção e uso para fins de estudo acadêmico foram autorizados pela empresa que os possuem), este trabalho relata os detalhes da trajetória parcial do *Scrum* em uma empresa desenvolvedora de *software* (referenciada a partir de agora apenas pelo termo “empresa”) desde sua implementação e até o término do *sprint* de número 75. *Sprint* é o nome dado à abordagem de desenvolvimento iterativa e incremental dessa metodologia (COHN 2011). É importante salientar que a trajetória mostrada é entendida como parcial porque o último *sprint* analisado neste trabalho não marca o término da adoção do *Scrum* pela empresa. A data do término do *sprint* 75 marca o fim dos dados históricos, se ordenados por data. O estudo realizado reflete dados históricos apenas, mas foram obtidas informações que favorecem a visualização do que ocorreu até então e possibilitam uma compreensão melhor do que pode ser aprimorado na execução dos próximos *sprints* e do *Scrum*, como um todo.

Este trabalho é dividido em mais 3 seções, além dessa introdução. Na seção 2, o termo *Scrum* é definido quanto às suas características e finalidade prática, o seu

entendimento como uma metodologia ágil é explicado, juntamente com o significado dessa expressão. A seção 3 apresenta o momento em que a transição foi proposta, os motivos que levaram à essa proposição e as particularidades de sua implementação. Em seguida, são apresentadas informações gerenciais ao longo do tempo, geradas a partir dos dados extraídos da empresa em questão e relevantes ao desenvolvimento do *software* acompanhado pelo estudo. A seção 4 trata das considerações finais quanto aos resultados obtidos e atribui dedutivamente causas para os problemas encontrados no estudo de caso. Também informa o que a empresa ainda planeja executar no tempo seguinte à publicação deste trabalho.

2. Referencial Teórico: *Scrum*

O *Scrum* é entendido como uma maneira ágil de gerenciar um projeto, onde o desenvolvimento de *software* ainda é a finalidade mais comum (SCRUM ALLIANCE 2015). Nesse contexto, o *Scrum* pode ser compreendido como um quadro para gestão dos processos envolvidos na atividade de criar e modificar o *software* (MOUNTAIN GOAT SOFTWARE 2015).

O *Scrum* não supõe uma descrição detalhada de como as coisas devem ser feitas. Ao contrário, tudo deve ser decidido dinamicamente com a participação de toda a equipe. Para alcançar esse feito, novos papéis devem ser exercidos. Um deles é o *ScrumMaster*. A função do *ScrumMaster* é remover os impedimentos ao progresso da equipe (COHN 2011). Ele tem autoridade sobre os processos, mas não sobre pessoas, ou seja, o papel de *ScrumMaster* atribui à pessoa que o exerce a tarefa de se responsabilizar sobre o que será feito no que diz respeito ao *Scrum*, mas não permite demitir ou admitir pessoas, por exemplo. Esse papel não precisa ser necessariamente preenchido por uma pessoa externa à empresa, embora, em alguns casos, seja interessante que ocorra dessa forma. Outro papel que o *Scrum* cria é o do dono do produto (*Product Owner*), que é a pessoa que garante que a equipe está se dedicando ao objetivo correto. Dessa forma, os dois papéis se apoiam: o dono do produto direcionando o alvo e o *ScrumMaster* garantindo que esse alvo seja atingido com eficiência.

Alguns valores do *Scrum* são inerentes à sua filosofia de metodologia ágil, tais como: interagir indivíduos entre si, mais do que processos e ferramentas; manter o *software* em funcionamento, mais do que documentação abrangente; colaborar com o cliente, mais do que negociar contratos; e responder a mudanças, mais do que seguir um plano (MANIFESTO ÁGIL 2015). Ou seja, há mais valor nos itens mencionados primeiramente do que nos que foram mencionados em seguida. A metodologia ágil também tem como princípios manter a satisfação do cliente, entregando *software* de valor continuamente e de forma adiantada, e aceitar que os requisitos mudem, mesmo quando o desenvolvimento acabar, de modo que o processo se adeque à mudança e ofereça vantagem competitiva ao cliente. Para que tudo isso dê certo, os indivíduos precisam estar motivados e isso pode ser conquistado através da transmissão de confiança no seu trabalho. Essa confiança é transmitida provendo o ambiente e suporte necessários para que as funções de cada integrante da equipe sejam bem exercidas. Outra forma simples de transmitir confiança e mitigar problemas de relacionamento é transmitir informações através de conversas diretas, de modo a produzir o mínimo de ruído de comunicação possível. Agindo dessa forma, o ambiente de desenvolvimento se torna

sustentável. As pessoas e os processos envolvidos são capazes de seguir adiante indefinidamente, sendo que a excelência técnica e o bom *design* possuem um papel fundamental nesse sentido. Por serem auto-organizáveis, os times ágeis devem estar sempre atentos para se manterem efetivos, ajustando seu comportamento à situação atual.

O *Scrum* possui artefatos para tornar ágeis os processos nele inseridos. Um deles, é o *backlog* do produto (Figura 1): uma lista completa e detalhada das descrições (ou requisitos) do produto, que vai sendo refinada conforme o produto avança. O dono do produto é quem mantém essa lista, por ordem de prioridade. É uma lista dinâmica, pois itens são adicionados, removidos e suas priorizações são alteradas à medida que o produto, o cliente e a equipe envolvida vão se tornando mais bem conhecidos.

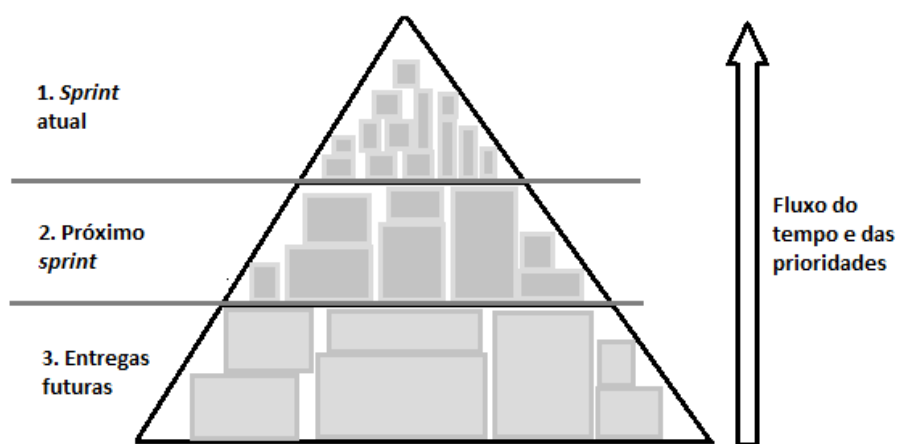


Figura 1. Exemplo gráfico do backlog do produto.

O *Sprint* é um outro artefato do *Scrum*, podendo ser descrito como um período onde as modificações acontecem de forma incremental e iterativa. Desenvolver de forma incremental significa construir uma parte do sistema de cada vez. Desenvolver de forma iterativa significa partir do pressuposto que o sistema deve ser feito todo de uma vez e que é impossível ou improvável conseguir construir corretamente um requisito logo na primeira tentativa, de forma que outras modificações serão necessárias conforme o *feedback* do cliente, até que o produto esteja pronto. Cada uma dessas duas formas de desenvolvimento, separadamente, possui seus próprios problemas. O *Scrum* resolve isso combinando-as e formando esse conceito, o *sprint*, através do qual é possível atender ao princípio da metodologia ágil: conseguir entregar *software* funcionando com alguma novidade de valor em um intervalo regular de tempo.

Outro artefato do *Scrum* é o gráfico *Burndown*, que é uma técnica para o rastreamento do andamento do *sprint*. Revela o quanto já foi feito e o quanto ainda falta fazer com relação ao total de trabalho que foi alocado no *sprint*. O gráfico da Figura 2 mostra duas linhas, em função do tempo. Uma delas desce, reta, partindo de um ponto alto do gráfico (onde está o total da carga de trabalho alocada para o *sprint*, em medida de tempo) até o ponto mais baixo do eixo vertical. Essa linha mostra como as requisições poderiam ser totalmente entregues, considerando uma taxa de entregas constante. A outra linha mostra as requisições que estão sendo, de fato, entregues no decorrer do *sprint*. As duas linhas dispostas na mesma imagem possibilitam visualizar se as entregas estão ocorrendo em uma frequência satisfatória ou não. O esperado é que a linha que

representa o que já foi entregue não se distancie demais da linha que representa a taxa de entrega ideal, como mostra a Figura 2, com todas as requisições entregues no vigésimo dia, para um *sprint* de 20 dias e 100 horas de trabalho alocado.

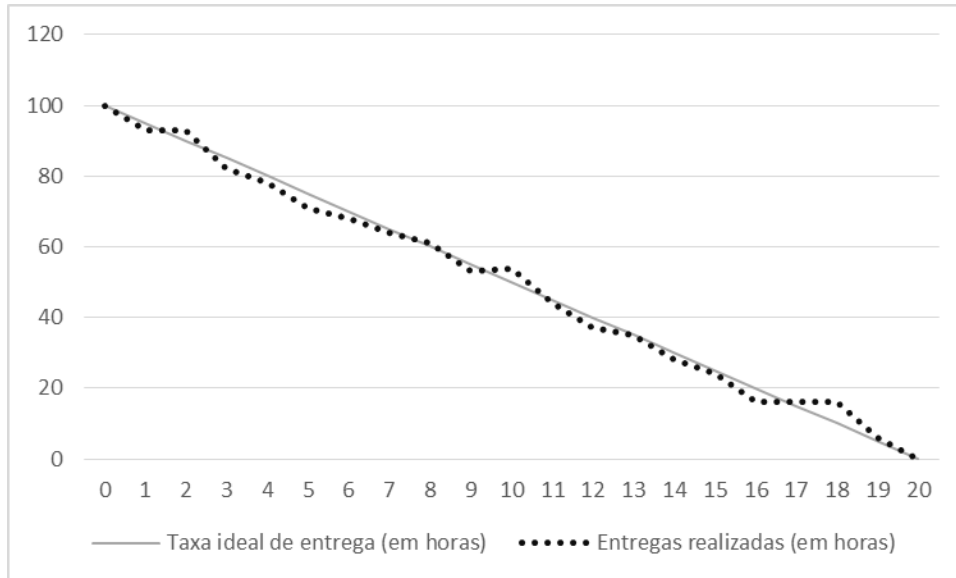


Figura 2. Exemplo de gráfico *Burndown* com taxa de entregas próxima a ideal.

Uma taxa de entrega que se distancie demais da linha de referência, estando muito acima dela (Figura 3), indica um ou mais problemas. Interessante observar que algo que merecia a atenção do *ScrumMaster* começou a ocorrer a partir do 5º dia de trabalho. Um *sprint* superdimensionado (ou seja, que teve mais carga de trabalho do que a equipe é capaz de executar) pode ser a causa para um gráfico que apresente esse padrão.

Outras causas possíveis são: excesso de interferência externa (demandas chegando por fora do *sprint*, fazendo com que as solicitações planejadas precisem esperar em função de solicitações extras que estão chegando), número de desenvolvedores insuficientes alocados no *sprint*, alta taxa de reprovação de requisitos entregues, faltas recorrentes ao expediente de trabalho e outros motivos possíveis que o *ScrumMaster* deverá identificar.

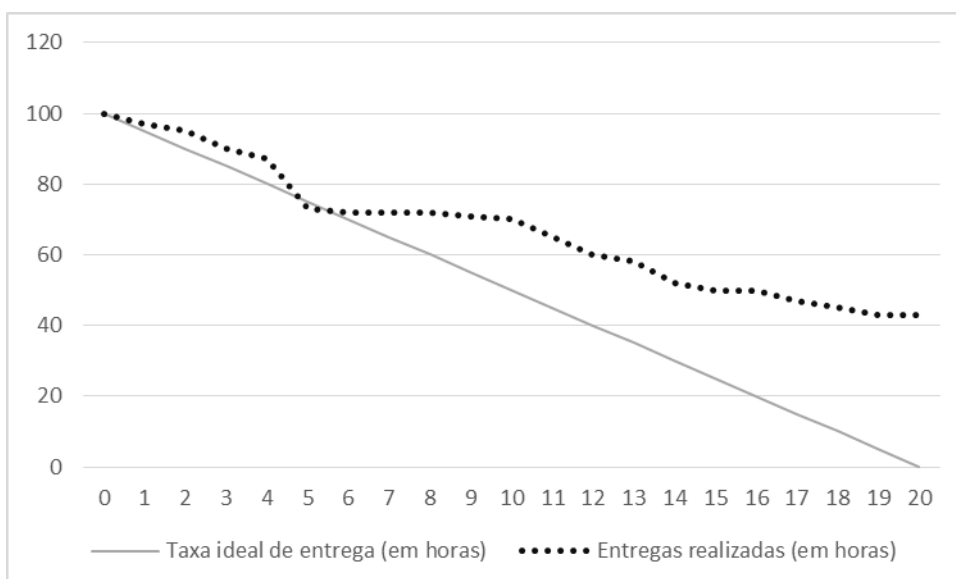


Figura 3. Exemplo de gráfico *Burndown* com taxa de entregas inferior à ideal.

Por sua vez, uma taxa de entrega que esteja muito abaixo da linha de referência (Figura 4), também pode indicar problemas a serem resolvidos para os próximos *sprints*. Percebe-se que o *ScrumMaster* não interveio por volta do 7º dia de trabalho, deixando a equipe ociosa ou trabalhando fora do *Sprint*. O gráfico pode se mostrar dessa forma quando, por exemplo (e ao contrário da situação anterior), o *sprint* está subdimensionado, ou seja, definido com uma carga de trabalho menor do que a equipe é capaz de executar, tornando o processo de construção do produto menos eficiente do que poderia ser. Ou ainda quando contém muitos requisitos que não agregaram valor ao produto porque foram cancelados ou descontinuados (o que pode ocorrer por qualquer motivo, mas não de forma muito recorrente) e mesmo assim foram considerados como concluídos, dentre outros possíveis fatores.

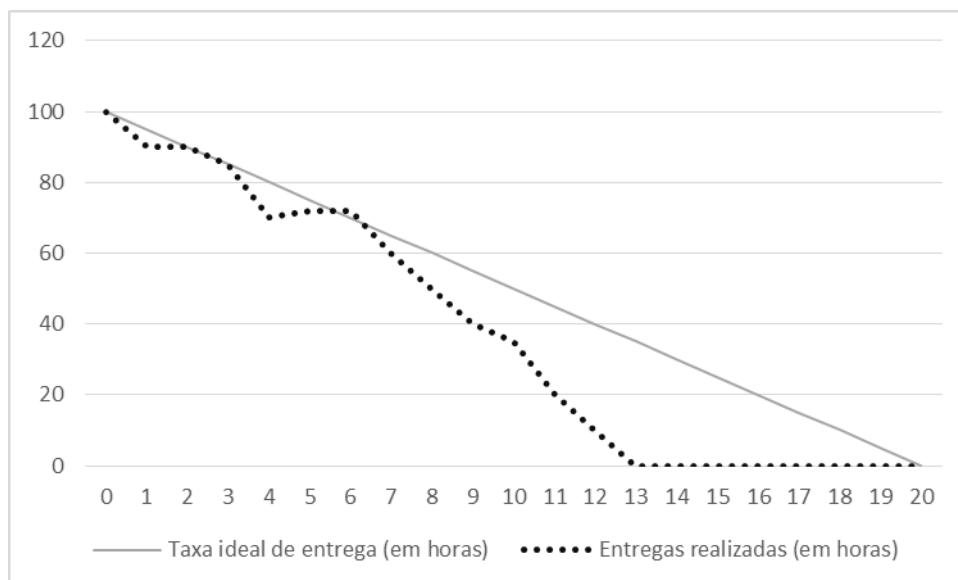


Figura 4. Exemplo de gráfico *Burndown* com taxa de entregas superior a ideal.

Situações em que a equipe consegue chegar à entrega das requisições no tempo estipulado para o *sprint*, mas com muito distanciamento da linha de referência em alguns momentos, também devem ser analisados com cautela para evitar futuras surpresas indesejáveis. Ou seja, toda situação que dispare algum alerta indica a necessidade da intervenção do *ScrumMaster* já no *sprint* corrente, ajustando-o às condições vigentes ou trabalhando para que a situação mude a favor do *sprint*. É importante aprender com o momento atual para estar apto a planejar o futuro com mais eficiência sempre.

3. Estudo de Caso

A motivação da empresa pela adoção do *Scrum* foi decorrente de um esforço iniciado pela gerência e por pessoas em cargos de liderança para retomar o controle das modificações que estavam sendo realizadas no produto. O produto em questão é um ERP (*Enterprise Resource Planning*), um tipo completo de sistema para apoio à decisão gerencial, pois contempla os mais diferentes módulos para esse fim, como produção, logística, finanças, entre outros, integrando todas as informações em tempo real (Souza e

Silva 2015). Esse *software* sofria modificações diárias e em grande volume, em sua maioria caracterizadas por correções de falhas, sendo que muitas delas eram oriundas de modificações também recentes. Em resumo, a empresa se mantinha basicamente para corrigir suas próprias falhas. Em sistemas ERP, há comumente uma pressão extra, que são as alterações legais para adequação do *software* à realidade fiscal, que está sempre a mudar e, naquele momento, isso não era diferente. Devido ao tempo gasto com retrabalho, os clientes estavam recebendo menos alterações do que demandavam.

A intervenção veio com a contratação de um consultor técnico, que iniciou um levantamento sobre a situação da empresa naquele momento e as práticas de desenvolvimento que adotava até então. As principais observações feitas e que poderiam ser atribuídas como causas do problema de ineficácia e ineficiência ao produzir *software* foram:

- não havia controle eficiente das mudanças no *software*, ou seja, não havia ciência de qual mudança viria nos dias a seguir. As próximas mudanças seriam, em boa parte, correções das mudanças recém-realizadas;
- o gerente do setor e o dono da empresa alternavam seu tempo entre tarefas gerenciais e de desenvolvimento de *software*, sem planejar o seu dia-a-dia a fim de realizar essas tarefas de forma equilibrada;
- para o controle das versões liberadas do ERP, parte da equipe, a que lidava com linguagem orientada a objetos, utilizava o *Subversion* (SUBVERSION 2015) para controlar versões, e a outra parte da equipe, que lidava com linguagem estruturada, utilizava um sistema de controle de versão mais antigo, o *Microsoft Visual SourceSafe* (VISUAL SOURCESAFE 2005). Ou seja, não havia reuso de subrotinas entre as equipes, o que é necessário, já que os módulos do ERP são integrados por processos e compartilham recursos;
- o sistema usado para o controle das solicitações de mudanças, chamado *Service Center*, autoral, estava em estágio de desenvolvimento e necessitava de suas próprias melhorias;
- o *backlog* do produto existia (mas não com esse nome), e era armazenado em um banco de dados tipo SQL Server sob a forma de registros de solicitações de mudança. Muitas dessas solicitações estavam em duplicidade por não terem sofrido revisão adequada quando foram abertas, o que as tornava potenciais candidatas à não execução por algum desenvolvedor ou ao seu cancelamento, caso entrassem em execução, ou mesmo a serem executadas e se tornarem fonte de mais falhas;
- inadequação a padrões de projeto que pudessem melhorar a manutenibilidade e o reuso de código;
- tentativas frustradas por parte da empresa em alocar esforços na migração do sistema feito em linguagem estruturada, antigo e legado, para um novo sistema, orientado a objetos, cujo desenvolvimento já estava em curso. Nesse sentido, foram idealizados futuros treinamentos sobre padrões de projetos, com o intuito de capacitar os desenvolvedores a criar um sistema melhor durante o processo de migração mas, até então, nunca haviam sido realizados.

Terminado o levantamento e após algumas reuniões, o consultor propôs o uso do *Scrum* como método para mitigar esses problemas. A metodologia ágil foi apresentada publicamente e houve a compreensão por parte de todos os envolvidos de que o *Scrum* como descrito por seus criadores não é a solução perfeita para nenhuma empresa e que, portanto, havia a necessidade de adequá-lo para aumentar sua conformidade. Por ter sido feita dessa forma, a transição para uma nova maneira de trabalho, relatada como sendo muitas vezes difícil devido à resistência que pode ocorrer por parte dos integrantes da empresa (COHN 2011), não foi, nesse caso, um problema a ser superado em nenhum nível hierárquico ou setor.

As requisições do produto são chamadas ainda hoje na empresa de RDMs (requisições de mudanças). Nesse trabalho, essa sigla não será usada, sendo substituída pelo termo solicitação de mudança, ou requisição. Para que seja possível avaliar a eficácia do *sprint*, foi importante definir o que seria uma solicitação entregue. Ficou definido na empresa chamar de concluídas as mudanças que foram entregues ao final do *sprint* e não necessariamente aprovadas pelo Controle de Qualidade, que é o setor da empresa responsável pelos testes internos do produto. Caso a mudança gerasse uma falha, fosse oriunda dos testes internos ou de clientes, sua correção era solicitada em uma nova solicitação de mudança, nem sempre corretamente associada àquela que a originou.

Os papéis de dono do produto, *ScrumMaster* e desenvolvedores foram definidos. Convencionou-se chamar o tempo estimado para a conclusão de uma solicitação de mudança como HH (ou homem-hora). As primeiras semanas foram dedicadas a disseminar a ideia entre os colaboradores e a estabelecer prioridades na execução das solicitações. Feito isso, aproximadamente 26 dias após a primeira reunião com o consultor, o *sprint* 1 teve início, cuja descrição foi “liberação de RDMs em atraso”.

As informações geradas para a avaliação do *Scrum* e sua aplicação na empresa foram obtidas através da análise das solicitações de mudança executadas nos *sprints*. Para qualquer organização que aplique o *Scrum*, o *sprint* bem-sucedido é aquele que teve suas solicitações dadas como concluídas pela equipe de desenvolvimento, independentemente dos parâmetros estabelecidos para considerar uma solicitação de mudança como tal. Solicitações não concluídas indicam algum problema, seja de dimensionamento de tempo para as tarefas, equipe não integrada entre si, usuário não integrado à equipe, falta de foco e muitas outras possíveis causas (COHN 2011). Todavia, na empresa estudada, mesmo se todas as solicitações forem consideradas como concluídas ao término do *sprint*, o Controle de Qualidade poderá ainda, nos dias seguintes (ou seja, durante o seguimento dos *sprints* seguintes), reprovar aquelas nas quais encontrar defeito ou inconformidade, podendo elas serem provenientes de qualquer *sprint* (anteriores ou atual). Isso é muito importante entender porque um fator chave para determinar o sucesso de um *sprint* é se o *software* resultante foi liberado de forma funcional e contendo valor para os usuários ou clientes do sistema. Ou seja, na empresa estudada, a informação sobre o sucesso de um *sprint* vai sendo refinada à medida que mais e mais de suas solicitações de mudança vão sendo testadas pelo setor de Controle da Qualidade.

Por isso, a metodologia aqui utilizada consiste em correlacionar o volume e a qualidade do trabalho concluído nos *sprints* analisados com os eventos ocorridos na

empresa que possam estar associados aos resultados obtidos a nível de *software*, sejam eles satisfatórios ou não. Há o interesse, inclusive, em diferenciar os resultados que possivelmente impactaram na qualidade final do software, mas que não podem ser diretamente relacionados ao advento do *Scrum*.

Para tal, foram extraídos dados sobre os *sprints* e sobre as condições da empresa durante o período em que estavam em curso. Esses dados podem ser categorizados de duas formas diferentes, quanto à sua origem:

- origem gerencial: dados resultantes do trabalho analítico do gerente do setor de Controle de Qualidade da empresa, que esteve no exercício dessa função durante todo o período apurado no estudo. Consiste em um conjunto de arquivos digitais que foram liberados por esse funcionário ao término da maioria dos *sprints*. Contêm métricas e dados quantitativos relacionados ao que foi concluído (e testado) nos *sprints*;
- origem de banco de dados: valores obtidos a partir do repositório oficial da empresa, que é um banco de dados SQL Server. O banco de dados foi alimentado ao longo do tempo por dois sistemas, ambos autorais da empresa, um chamado *Service Center*, e o outro, *Gerenciador*, cujas funcionalidades mais relevantes a esse trabalho são, respectivamente: gestão do fluxo de trabalho (registro de solicitações de mudança e agenda de eventos) e gestão dos recursos humanos (admissão e desligamento de funcionários e mudanças de cargos). Dessa forma, esse repositório forneceu dados históricos sobre o andamento dos *sprints* e também sobre eventos relacionados à gestão de pessoas na empresa.

Alguns dados obtidos do repositório SQL Server tiveram uma finalidade mais específica, que foi a de servir como fonte de dados para o *software Give Me Views v1.0* (TAVARES et al. 2015). Esse *software* foi usado por ter sido criado para o apoio na análise visual de mudanças em projetos de *software* realizadas pela indústria, facilitando as tomadas de decisão sobre mudanças futuras por parte dos desenvolvedores, sendo um projeto de mestrado do programa de Ciência da Computação da UFJF.

A empresa estudada possui sua própria classificação de solicitações de mudanças. O *GiveMe Views* possui um filtro de dados onde deve-se classificar as solicitações de mudança conforme tipos de manutenção pré-definidos, sendo eles: manutenções corretivas, manutenções adaptativas ou manutenções evolutivas. Logo, para a análise dos *sprints* da empresa, o padrão de classificação da empresa foi associado aos tipos do *GiveMe Views* da seguinte maneira:

- manutenções corretivas: erro no sistema;
- manutenções adaptativas: alteração legal;
- manutenções evolutivas: sugestão de melhorias, complemento de tarefa, orçamento e solicitação de novo recurso.

Outras classificações de mudança realizadas pela empresa não foram atribuídas a nenhum tipo, tais como: liberação de módulos, documentação do *help* e treinamento interno; assim sendo, as solicitações classificadas com essas formas foram desconsideradas na *GiveMe Views* porque não estão associadas a mudanças no sistema ERP, que é o objeto de estudo.

Apesar do interesse mostrado pela empresa em adotar o *Scrum*, as adaptações ao modelo (ou a falta de cuidado na montagem do *sprint*) trouxeram problemas, como será visto a seguir. A equipe convencionou que cada *sprint* duraria quinze dias corridos. O *sprint* 1 foi aberto com as seguintes características:

- 158 solicitações de mudanças adicionadas;
- 11 desenvolvedores alocados;
- gerente do setor de desenvolvimento conciliava também tarefas de *ScrumMaster* e desenvolvedor;
- dono do produto possuía também os papéis de desenvolvedor, diretor da empresa e sócio fundador, conciliando as tarefas relacionadas a cada um deles;
- total de 314 de HH total atribuído ao *sprint*.

Estudando esses dados, destacaram-se alguns pontos interessantes. Onze desenvolvedores é um número maior que a média para os padrões atuais do *Scrum*. Equipes menores ou maiores do que a faixa de 4 a 9 membros reportam menor chance de sucesso, conforme pesquisas recentes (SCRUM ALLIANCE 2015). Contudo, há um problema ainda mais evidente no valor do HH total atribuído ao *sprint*: havia muito mais quantidade de solicitações alocadas no *sprint* do que recursos humanos para mensurá-las. Dessa forma, muitas solicitações acabaram por ficar sem HH definido. O *sprint* 1, na verdade, foi apenas um repositório do que precisava ser feito de mais urgente, e não um período no tempo onde as tarefas que seriam executadas estavam planejadas.

Ou seja, até esse momento, percebe-se que havia um entendimento equivocado sobre o *Scrum*. Não ficou claro para a equipe que o *sprint* precisa ser composto de solicitações passíveis de serem concluídas, e que todas deveriam ter sido previamente mensuradas pela equipe quanto ao tempo previsto para executá-las; ou é possível deduzir que o volume de correções urgentes era tão grande que impediu o planejamento correto do *sprint*. Mas isso não justificaria, de qualquer forma, esse equívoco. Afinal, tendo em vista que o trabalho do *ScrumMaster* estava dividido entre programar e montar o primeiro *sprint*, o que ocorreu foi que não restou tempo suficiente para executar corretamente a segunda função. E, por ter sido escalado para agir como desenvolvedor no *sprint* 1, o *ScrumMaster* gastou novamente muito tempo com desenvolvimento, deixando suas atribuições com o *Scrum* em segundo plano. Das 88 horas de seu expediente de 10 dias, 81 horas gastou apenas executando solicitações de mudança, como um desenvolvedor normal do *sprint* 1.

Ter um *ScrumMaster* que concilia tarefas de desenvolvedor é sempre muito arriscado (COHN 2011) mas, em casos como esses, onde o volume de tarefas de desenvolvimento urgentes era grande, não conseguir ter um *Scrum* bem acompanhado pelo *ScrumMaster* era mesmo um problema muito provável de ocorrer. Devido a esses resultados, estava clara a necessidade de desvincular a tarefa de *ScrumMaster* da tarefa de desenvolvedor e atribuir a responsabilidade sobre cada uma delas a pessoas diferentes.

Em consonância a isso, há o fato de que o dono do produto também desenvolvia função dupla. Mas esse fato, diferentemente do anterior, não deve ser apontado como uma das causas do não cumprimento do *sprint*. Por ser uma equipe iniciante no *Scrum*, o *ScrumMaster* é muito solicitado, mas o dono do produto, nesse momento, nem tanto.

Como a equipe ainda não é eficiente no *Scrum*, não há muitas questões a indagar a ele. Muito tempo é gasto aprendendo sobre o *Scrum* e como segui-lo, conforme foi adotado. Com o tempo, à medida que a equipe começa a se preocupar menos com o *Scrum* e mais com o produto, a tendência é que o dono do produto seja mais requisitado (COHN 2011). Ou seja, apenas nos próximos *sprints* essa situação pode ter causado problemas. Nas considerações finais deste trabalho ela será reavaliada.

Pode-se observar na Figura 5 que, apesar do dimensionamento incorreto, o *sprint* chegou próximo de ser cumprido. Contudo, isso não justifica a forma como foi montado, pelos motivos já explicados anteriormente. Por exemplo, do total de 158 solicitações de mudanças adicionadas nesse *sprint*, apenas 1 solicitação de mudança comportou, em si, 90 horas de HH previsto para sua execução. Se o valor do HH total do *sprint* foi de 314, como já mencionado também, isso significa que apenas uma solicitação de mudança ocupou 28% do tempo alocado para todo o *sprint*, mesmo representando menos de 1% da quantidade total de solicitações nele contidas. Trabalhar dessa forma não é trabalhar com *Scrum* pois, se o *Scrum* prevê que o trabalho seja feito de forma incremental, é necessário dividir as tarefas em partes menores para que o acompanhamento seja efetivo. Além do mais, uma solicitação de mudança grande reduz as chances que a equipe consiga entregar algo de valor periodicamente ao cliente. O gráfico mostrado na Figura 5, como consequência disso, mostra duas linhas distantes, o que aponta problemas no *sprint*, como já explicado anteriormente nesse trabalho.

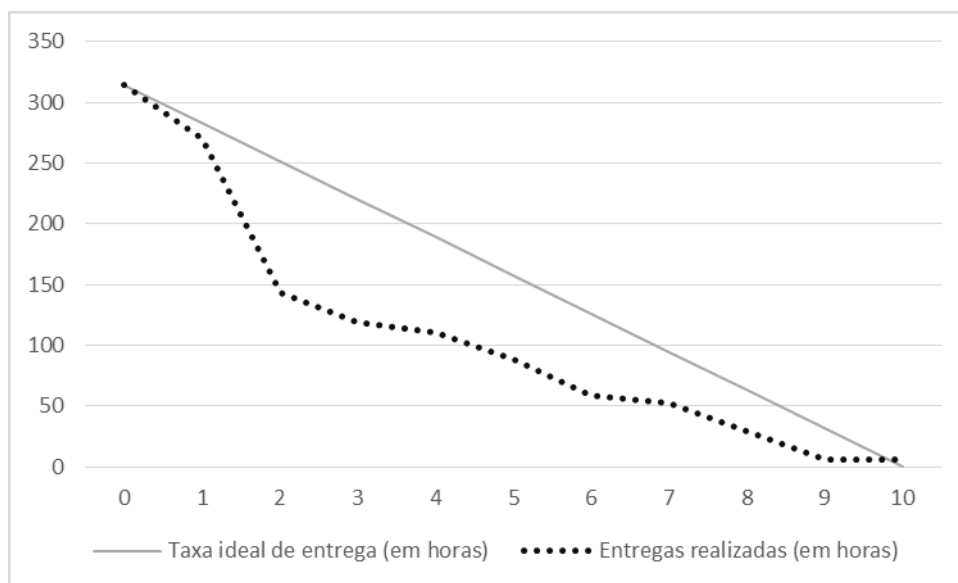


Figura 5. Gráfico Burndown do sprint 1, onde os sprints começaram subdimensionados.

Há de se considerar também que mesmo as solicitações que tiveram o HH definido não o tiveram com base em uma estimativa da equipe, mas do *ScrumMaster* ou do dono do produto, apenas. Essa não é uma prática ágil. O *backlog* do produto deve ser discutido com a equipe, assim como o tempo a ser gasto com as solicitações que serão incluídas no *sprint* (COHN 2011).

Não houve apuração do resultado do *sprint* 1 após o seu término. Seguindo com o estudo mais à frente no tempo, a fim de observar como o *Scrum* progrediu, é visível que, até o *sprint* 10, ainda não havia avaliação do término dos *sprints* de nenhuma forma. As iterações eram feitas no produto, mas não havia qualquer análise com relação

ao que havia sido concluído pela equipe de desenvolvimento, nem sobre o que já havia sido testado pela equipe de teste. O que não era concluído, ou ficava sem solução, ou era repassado automaticamente para o *sprint* seguinte. Somente a partir do término do *sprint* 11 em diante o gerente do setor de Controle de Qualidade passou a liberar planilhas que continham informações visuais que ajudaram a equipe a acompanhar o andamento. Essas planilhas continham o volume total de liberações aprovadas nos testes da equipe de qualidade, por *sprint*, assim como o total de solicitações de mudança adicionadas aos *sprints*.

A Figura 6, gerada a partir desses dados, mostra um gráfico oscilante. É interessante observar que, em alguns *sprints*, entre o 48 e o 58 principalmente, a equipe de desenvolvimento esteve próxima de concluir todas as solicitações de mudança por *sprint*. Contudo, o fator mais importante a ser analisado é que em nenhum *sprint* o número de solicitações concluídas chegou perto do número de solicitações aprovadas pelo setor de qualidade, o que indica um alto índice de reprovação. Sabendo que o setor de qualidade considera como reprovada nos testes qualquer solicitação que ao menos uma vez não atenda aos critérios de conformidade ou apresente um software com falha, entende-se pelo gráfico que, se há um índice de reprovação alto ao mesmo tempo em que há uma taxa de entrega próxima do ideal, então, isso significa que a equipe de desenvolvimento reexecutou muitas vezes a mesma solicitação até que ela fosse aceita como concluída pela equipe de qualidade. Com isso, entende-se que as solicitações acabavam levando mais tempo para serem entregues do que o planejado.

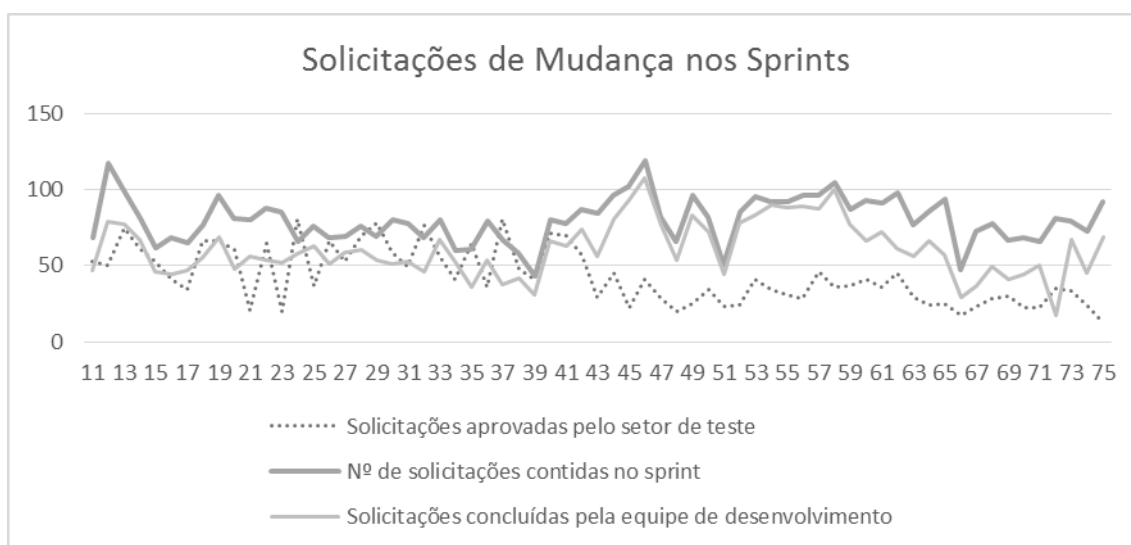


Figura 6. Relação entre solicitações de mudanças inseridas, concluídas e aprovadas, por *sprint*.

De fato, a Figura 7 comprova essa teoria. Em todos os *sprints* analisados, o tempo efetivamente gasto para conseguir concluir algumas (não todas) solicitações de mudança excede muito o tempo que havia sido estimado para elas. Um fator relacionado ao *Scrum* que justifica em parte essa proximidade entre o número ideal de conclusões e o número de conclusões de fato realizadas após o *sprint* 52 (vide Figura 6) é a informação extraída do *Service Center* que mostra que, a partir do final da data de início desse *sprint*, as reuniões diárias (*daily scrums*), previstas pela teoria do *Scrum*, começaram a ser agendadas para se iniciarem todos os dias às 9 horas e 15 minutos, com a

participação de todos os desenvolvedores e apresentação do gráfico *Burndown* ao final de cada *sprint*, mostrando um esforço por parte da equipe em melhorar o controle sobre o que estava sendo executado no *Scrum*. Essa prática se mantém até os dias atuais.

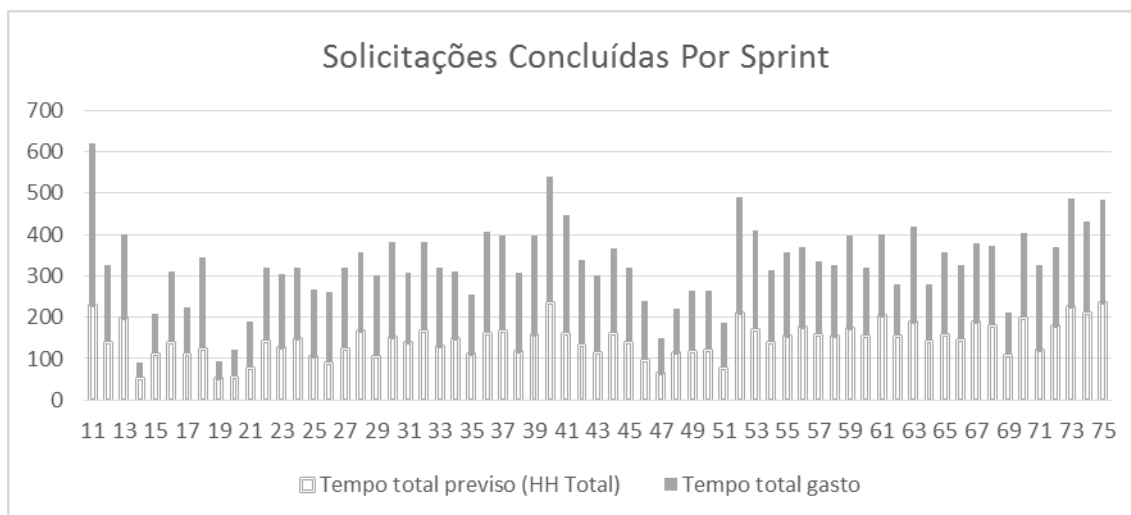


Figura 7. Comparativo entre tempo previsto para a conclusão das solicitações e o tempo efetivamente gasto para concluí-las.

Alguns eventos importantes ocorridos no período analisado, não diretamente relacionados às práticas *Scrum*, certamente também influenciaram nos resultados apresentados. Esses eventos serão comentados a seguir.

No *sprint* 12, um novo desenvolvedor foi contratado pela empresa e lhe foi atribuída a responsabilidade de melhorar o *Service Center*, adicionando a ele novos *status* possíveis para as solicitações de mudanças, algo já almejado pela empresa desde antes do *Scrum*. Antes, alguns processos não eram bem documentados. A solicitação de mudança avançava no *sprint*, saindo da situação de “em execução pelo desenvolvedor” para “em teste pelo Controle de Qualidade” diretamente, e logo que possível já era declarada como “concluída”, independente da ocorrência de reprovações no processo, por exemplo. Outro *status* não documentado era o “aguardando *merge*”, que era (e ainda é) importante para as solicitações aprovadas onde houve alteração em código-fonte que já estava sendo trabalhado com o uso de *branches*, que é um recurso do sistema controlador de versão que possibilita que a modificação realizada por um desenvolvedor seja isolada da modificação que está sendo realizada pelos outros (SUBVERSION 2015). Todas essas situações eram controladas manualmente, por fora do *Service Center*. Tendo isso em vista, esse desenvolvedor, com a ajuda da equipe, inseriu novos *status* para as solicitações que permitiram à empresa controlar o fluxo do trabalho com muito mais segurança. O resultado dessa mudança se reflete em melhores taxas de aprovação do *sprint* 12 ao 20 (Figura 6).

Foi observada pela gerência a necessidade de atribuir a função de *ScrumMaster* a outro integrante da equipe durante o *sprint* 13. Já foi apresentado neste trabalho que o término do *sprint* 1 já apontaria essa necessidade se tivessem sido geradas informações gerenciais a partir dele, na época. Isso foi necessário exatamente porque não estava sendo possível atribuir à mesma pessoa as funções de desenvolvedor e orientador de *Scrum*, ficando o antigo *ScrumMaster*, portanto, apenas com a função de desenvolvedor. A Figura 7 mostra que, no *sprint* 14, subsequente a esse evento, o número de

solicitações de mudanças foi visivelmente inferior. Isso pode ter ocorrido devido ao esforço do novo *ScrumMaster* em se adaptar à nova função, ou mesmo porque decidi incluir nesse *sprint* apenas as solicitações que haviam sido previamente discutidas pela equipe, ao invés de inseri-las sem nem mesmo atribuir HH, como ocorreu no *sprint* 1, apenas por serem urgentes.

A empresa iniciou, no *sprint* 22, um esforço para extinguir seu controlador de versão mais antigo, o *Microsoft Visual Sourcesafe*. Ao chegar no *sprint* 28, esse sistema havia sido totalmente descontinuado, tendo sido substituído integralmente pelo *Subversion*. Juntamente com essa mudança, veio a utilização de *branches* por todos os membros da equipe de desenvolvimento (SUBVERSION 2015), ao invés de somente parte dela, como ocorria até então. Ou seja, antes da implantação do *Subversion*, alguns arquivos do código-fonte só podiam ser alterados por um desenvolvedor de cada vez, já que o recurso de *branch* do *Sourcesafe* não era utilizado, apesar de existir. Trabalhar com *branches* trouxe inúmeras vantagens. Antes, havia o risco de uma modificação ser incorporada a outra de forma consciente, mas inevitável, já que o código-fonte que estava sendo modificado podia estar sendo compartilhado entre vários desenvolvedores. Essa era uma situação de trabalho totalmente inviável e sua substituição pelo uso de *branches* contribuiu para a qualidade final do produto. Como possível resultado, o número de solicitações aprovadas pelo Controle de Qualidade nunca foi tão alto por tanto tempo, indo do *sprint* 24 ao 43 (Figura 6). Na verdade, a quantidade de solicitações aprovadas chegou a superar a de concluídas, o que indica que algumas correções realizadas pelos desenvolvedores, utilizando esse método de trabalho eliminaram, de uma só vez, mais de uma solicitação de mudança.

Já no *sprint* 46 foram realizadas algumas trocas de pessoal no setor de Controle de Qualidade. A Figura 6 mostra que, nesse momento do tempo em diante, houve uma mudança na realidade do *Scrum* em um fator diretamente relacionado aos testes: o volume de reprovações, que aumentou muito. Isso, que analisado isoladamente é um problema, quando colocado em contexto pode ser entendido como um acontecimento positivo, já que as reprovações contribuíram para solicitações quase totalmente entregues ao final dos *sprints* seguintes ao 46, sendo que esse fenômeno perdurou por mais ou menos 10 *sprints* consecutivos. A dedução a ser feita é a de que menos código defeituoso chegando no cliente implicam em menos solicitação urgente de correção (o que de fato ocorria e atrapalhava o andamento dos *sprints*). O resultado positivo desse evento de troca de pessoal no setor de qualidade pode ser o motivo do sucesso mostrado no gráfico de *Burndown* do *sprint* 47 (Figura 8), gerado apenas 2 semanas após o término do *sprint* 46.

Recentemente a empresa progrediu muito com os sistemas mais novos, que são escritos sobre o paradigma da orientação a objetos. Os desenvolvedores desses novos sistemas estão trabalhando com técnicas de desenvolvimento ORM (*object-relational mapping*), através de um *framework* próprio. Dessa forma, hoje, nos sistemas novos, o desenvolvedor está cada vez menos trabalhando diretamente com SQL e cada vez mais com uma interface de programação que gera as *queries* SQL por ele. Em adição a isso, o *software* legado da empresa hoje está totalmente integrado aos novos sistemas através de um componente desenvolvido internamente que faz a ligação entre os dois, tornando possível o compartilhamento em duas vias de telas e sub-rotinas, ou seja, agora, os

módulos do ERP estão totalmente integrados entre si. Os resultados dessas mudanças ainda não foram evidentes por serem muito recentes, mas são promissores.

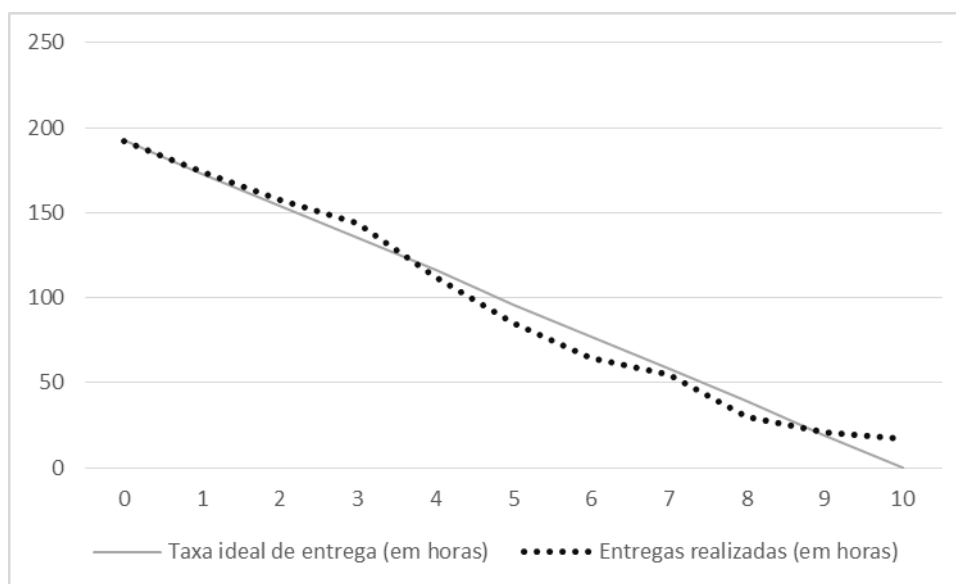


Figura 8. Burndown do sprint 47: um dos melhores resultados de sprint na trajetória do Scrum na empresa.

Contudo, e para finalizar o estudo, nota-se que mesmo com todos esses acontecimentos importantes, benéficos ou não, e relacionados diretamente ao *Scrum* ou não, a relação entre o volume de solicitações corretivas, adaptativas ou evolutivas e o total de solicitações abertas no geral praticamente não mudou com o tempo nem para melhor, nem para pior (veja Tabela 1).

Tabela 1. Métricas geradas pelo GiveMe Views.

Dados anteriores ao <i>sprint</i> 1	Dados anteriores ao <i>sprint</i> 75
Total de solicitações: 2615	Total de solicitações: 9373
Solicitações corretivas: 2000 (77%)	Solicitações corretivas: 6912 (74%)
Solicitações adaptativas: 91 (3%)	Solicitações adaptativas: 419 (4%)
Solicitações evolutivas: 524 (20%)	Solicitações evolutivas: 2042 (22%)

4. Considerações Finais e Trabalhos Futuros

Ao analisar a trajetória do *Scrum* na empresa parceira, observaram-se falhas no uso do *Scrum*, algumas delas prevalecendo ainda hoje. Essas falhas impedem que os *sprints* sejam concluídos com sucesso. Para que sejam, será necessário por parte de todos os integrantes um maior compromisso com os princípios ágeis.

Não haverá *sprint* terminado com todas as solicitações concluídas se não houver planejamento e monitoramento eficientes. Hoje, desde o *sprint* 46, a empresa já realiza o *planning poker*, que é um modelo de previsão de tempo baseado na experiência da equipe e utilizado, portanto, para gerar o HH das solicitações de mudança. Nada semelhante havia ocorrido até esse *sprint*, e isso (combinado à troca do time de

testadores de *software*, mencionado anteriormente) pode ter contribuído de alguma forma para a melhor taxa de conclusões ocorrida nessa época (Figura 6). Esse acontecimento mostra a importância de implementar técnicas eficientes de planejamento e controle.

Para obter mais eficiência, as organizações envolvidas com o *Scrum* podem viabilizar algumas mudanças em suas práticas, a fim de integrar toda a equipe nesse mesmo propósito. Uma delas é disseminar a ideia da posse coletiva do código. Essa forma de pensar e programar pressupõe que todos os desenvolvedores devem se sentir donos dos produtos no processo de desenvolvimento. Alguma especialização por parte deles é aceitável, mas a ideia de associação exclusiva de parte do sistema a poucos desenvolvedores (ou até mesmo a um apenas) não é uma boa prática. A empresa estudada neste trabalho vive esse cenário, já que o dono do produto também é um desenvolvedor ativo nos *sprints*. Dessa forma, a ideia de dono do produto pode se confundir com a ideia de dono do código. Incentivar a posse coletiva consiste exatamente em desvincular o código criado de quem escreveu as linhas, criando em cada membro da equipe um senso de responsabilidade pelo programa como um todo. Outra técnica interessante é a programação em pares, que ajuda, inclusive, na ideia de posse coletiva, por favorecer a disseminação do conhecimento sobre o código. Foi observado que, pelo menos durante algum período, a taxa de reprovação existente na empresa analisada favoreceu a qualidade do produto final, mas também foi dito que, para isso, está sendo necessário que a equipe desenvolvedora reexecute a mesma solicitação várias vezes. Programar em pares pode ajudar a minimizar o índice de reprovações e, ao mesmo tempo, trazer a taxa de conclusões de solicitações para níveis aceitáveis nos *sprints*. Um programador ajudando o outro implica na disciplina dos dois, pois quando um não está agindo de forma disciplinada ou atenta, o outro o ajuda a fazê-lo, e isso resulta em mais qualidade. Não é necessário, contudo, mudar completamente a forma de programar: algumas horas de programação pareada por dia já devem surtir resultado.

Associar solicitações de correção de falhas às suas causas também é imprescindível, já que, fazendo isso, será possível identificar e classificar suas origens (estejam elas em um programador da equipe que está cometendo muitos defeitos no software, ou em uma rotina que precisa ser refatorada) para que sejam tomadas as decisões estratégicas cabíveis. Ainda, sobre desenvolvedor que comete erros, uma forma interessante de evitar esse fenômeno é revisando seu código antes mesmo que ele possa liberá-lo para a equipe de testes. A programação pareada ajudaria nesse sentido, mas ter o código revisado pela equipe, ou parte dela, antes de concluir uma solicitação é mais uma forma de evitar problemas futuros. Como há muitos olhos críticos sobre o código, a excelência no desenvolvimento e a codificação dentro dos padrões estabelecidos pela empresa são favorecidos.

Uma forma ainda mais direta de monitorar o *Scrum* é através da Avaliação Comparativa de Agilidade, que está disponível gratuitamente na *web* (COMPARATIVE AGILITY 2015). Baseada em respostas individuais, essa pesquisa pode ser projetada para ser respondida por um *ScrumMaster*, um *coach*, ou um consultor experiente, que represente a equipe. As respostas dadas serão agregadas e poderão ser comparadas com o banco de dados da *Comparative Agility*, onde várias parametrizações são possíveis. Todavia, as avaliações mais importantes podem ser aquelas criadas pelas próprias

organizações. Empresas com grandes iniciativas *Scrum* como a *Ultimate Software*, *JDA Software*, *Yahoo!* e *Salesforce.com* seguiram esse caminho (COHN 2011).

O importante nesse sentido é que a equipe se mantenha alerta e se ajuste ao que for necessário, sempre observando os princípios ágeis. Esses princípios devem, na verdade, ser observados antes mesmo da implementação do *Scrum*, ou de forma independente a ele. Os eventos de troca de pessoal, aplicação de novas técnicas de desenvolvimento e melhoria ou atualização dos sistemas internos de controle, apesar de não terem contribuído para diminuir o volume solicitações de correção de erros (como visto com as métricas geradas pelo *GiveMe Views*), foram importantes para compor o cenário atual da empresa, que hoje possui códigos muito mais fáceis de manter, opera com maior tranquilidade devido à baixa urgência das solicitações (apesar do grande volume de falhas que ainda persiste) e sem a necessidade de solicitar horas extras frequentes aos funcionários. Provavelmente essas mudanças poderiam ter ocorrido mais cedo se a empresa já estivesse seguindo os princípios ágeis a mais tempo. Com isso, pode-se deduzir que ser ágil é uma ideia que pode e deve ser colocada em prática independentemente de quando se dará o início dos *sprints* e da geração dos gráficos de *Burndown*.

A fim de mitigar as dificuldades encontradas na execução do *Scrum*, a principal tarefa a ser exercida é a mensurar com atenção o que está ocorrendo na empresa ao longo de sua aplicação. Como explicado na introdução deste trabalho, não mensurar o *Scrum* faz com que ocorra uma descrença quanto à sua eficiência. Existem métodos de combate a isso e a avaliação não precisa ser exata, porque não há uma forma de se medir *software* de tal maneira. Isso, no entanto, não significa que não há formas aproximadas de se fazê-lo. Soluções relativamente precisas para se medir resultados envolvem análise de dados quantitativos simples como, por exemplo, extrair dados sobre volume e data de chamadas registradas pelo suporte da empresa que alegam defeitos no sistema para, em seguida, confrontá-los com dados dos *sprints* que as antecederam, em busca de um problema que possa ter passado despercebido nas reuniões diárias, tais como: impacto de ausências ou excesso de contingente de desenvolvedores em algum *sprint*, indicando talvez, para a primeira situação, uma falha na concessão de folgas, férias, ou um excesso de faltas ao expediente de trabalho; desmotivação de um ou mais integrantes da equipe, o que demanda trabalhar o moral dos integrantes afetados; falha recorrente de comunicação, devido possivelmente a algum fator que esteja causando um afunilamento na propagação de informação entre setores; enfim, qualquer problema que não tenha sido detectado a tempo de prevenir a falha. Se os períodos em que houve maior reporte de falhas coincidirem com as semanas seguintes ao término de um ou mais *sprints*, talvez faça sentido para sua equipe acreditar que o *Scrum* não está funcionando. Afinal, nem todos estão em uma posição tal que os permita compreender bem a situação. As mudanças de comportamento que o *Scrum* demanda podem torná-lo um alvo mais fácil para críticas do que os discretos problemas que realmente podem ser a causa das falhas reportadas. Portanto, cabe ao *ScrumMaster*, e aos outros líderes envolvidos, identificar esses problemas, resolvê-los, aprender com eles e planejar os próximos *sprints* em função disso. A viabilização da identificação de problemas relacionados ao desenvolvimento do produto e a possibilidade de apontar causas para eles produzem o cenário ideal: os descrentes vão (cada vez mais com o tempo) entendendo que as técnicas do *Scrum* são importantes e que valem a pena serem aplicadas. Ou seja, o

Scrum, quando é bem monitorado e frequentemente reajustado às condições do momento (o que é um princípio do desenvolvimento ágil), cria possibilidades de análise que antes não existiam, favorecendo a diminuição de falhas no produto final; o que, como consequência natural, produz mais confiança no próprio *Scrum* por parte da equipe que o compõe, aumentando ainda mais sua eficiência, diminuindo novamente o índice de falhas e, assim por diante, criando um ciclo autossustentado.

Entendida a importância do monitoramento e planejamento, fica evidente que a falta desse hábito foi decisiva para os resultados aquém das expectativas no estudo de caso mostrado neste trabalho.

A empresa avaliada segue ativamente com o *Scrum*. Atualmente, tem executado um trabalho sobre as solicitações de mudança que foram abertas a mais de um ano, cujo objetivo é dar alguma sequência a elas, seja adicionando-as a um *sprint* ou realizando o seu arquivamento (como poderá acontecer com aquelas que se originaram de orçamentos cujo pagamento não foi efetivado, por exemplo). Hoje, devido ao controle de versão mais eficiente, já é mais fácil vincular as falhas reportadas pelos clientes às solicitações de mudança que as originaram. Já a migração das linhas de código escritas em linguagem estruturada para linguagem orientada a objetos tem acontecido a passos mais modestos, porém constantes. Os líderes do setor de desenvolvimento planejam ministrar os cursos sobre padrões de projetos idealizados inicialmente, e a diretoria geral declarou recentemente a intenção de manter o dono do produto apenas como sócio, removendo-o da função de desenvolvedor.

Referências

Cohn, M. “Desenvolvimento de Software com Scrum”. São Paulo: Bookman, 2011. 496 p.

COMPARATIVE AGILITY: <https://comparativeagility.com/>. Acessado em novembro, 2015.

Cruz, F. “Scrum e PMBOK Unidos no Gerenciamento de Projetos”. Rio de Janeiro: Brasport, 2013. 382 p.

MANIFESTO ÁGIL. Manifesto para o Desenvolvimento Ágil de Software. Disponível em: <http://www.manifestoagil.com.br/>. Acessado em novembro, 2015.

Mittal, N. “The Burn-Down Chart: An Effective Planning and Tracking Tool”. Disponível em: <https://www.scrumalliance.org/community/articles/2013/august/burn-down-chart-%E2%80%93-an-effective-planning-and-tracki>. Scrum Alliance, 2015.

MOUNTAIN GOAT SOFTWARE: Disponível em: <http://www.mountaingoatsoftware.com/agile/scrum>. Acessado em novembro, 2015.

SCRUM ALLIANCE: <http://www.pm-progetti.it/download/scrum-alliance-state-of-scrum-2015.pdf>. Acessado em novembro, 2015.

Souza, J., Silva, D. (2015) “Cultura Organizacional: O Fator Chave para o Sucesso da Implantação de um Sistema ERP”. Disponível em: <http://177.107.89.34:8080/jspui/bitstream/123456789/313/1/SouzaSilva.pdf>. Acessado em novembro, 2015.

SUBVERSION: <http://subversion.apache.org/>. Acesso em novembro, 2015.

Tavares, J., David, J., Araújo, M., Braga, R., Campos, F. e Carneiro, G. (2015) “*GiveMe Views*: Uma Ferramenta de Suporte a Evolução de Software Baseada na Análise de Dados Históricos”. Disponível em:
<http://www.lbd.dcc.ufmg.br/colecoes/sbsi/2015/009.pdf>. Acesso em novembro, 2015.

VISUAL SOURCESAFE: [https://msdn.microsoft.com/pt-br/library/ms181038\(v=vs.80\).aspx](https://msdn.microsoft.com/pt-br/library/ms181038(v=vs.80).aspx). Acesso em novembro, 2015.