ANÁLISE DE DINÂMICA DE SISTEMAS DO PROCESSO SCRUM

Felipe Madureira Fonseca

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro
Junho de 2016

ANÁLISE DE DINÂMICA DE SISTEMAS DO PROCESSO SCRUM

Felipe Madureira Fonseca

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Geraldo Bonorino Xexéo, D.Sc.

_____

Prof. Jano Moreira de Souza, Ph.D.

_____

Prof. Rodrigo de Toledo, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 2016

Fonseca, Felipe Madureira

Análise de dinâmica de sistemas do processo SCRUM / Felipe Madureira Fonseca. – Rio de Janeiro: UFRJ/COPPE, 2016.

XII, 105 p.: il.; 29,7 cm.

Orientador: Geraldo Bonorino Xexéo.

Dissertação (Mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2016.

Referências Bibiliográficas: p. 73-79.

1. *Scrum* 2. *System Dynamics.* I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Agradeço primeiramente à minha mãe, Regina, de força inigualável, que mesmo em frente a todas as dificuldades, se mantém firme e pronta para fornecer apoio a todos que precisam.

Agradeço à minha esposa, Flavinha, pelo amor incondicional, pela paciência e por todos os momentos que passamos juntos. Nossa nova jornada está apenas começando.

Agradeço ao meu Pai, Oswaldo, que mesmo não estando sempre presente, está sempre pronto para me apoiar, suportar e orientar todas as difíceis decisões que a vida nos traz a todos os momentos.

Agradeço ao meu irmão, Gustavo, pelo suporte e companheirismo em muitos momentos.

Agradeço a todos os amigos da GPE, companheiros durante muito tempo. Agradeço em especial a Luiz Tomaz, Ester Lima e José Augusto Rodrigues pelas orientações e por gentilmente terem cedido seu tempo para ajudar na composição desse trabalho. Agradeço também a Ricardo Barros, companheiro e parceiro dentro e fora do ambiente de trabalho e ao Bruno Osiek por me ajudar a não deixar a peteca cair.

Agradeço ao meu orientador Prof. Geraldo Xexéo por todo seu apoio, confiança e suporte ao longo desse período.

Agradeço ao Prof. Jano e ao Prof. Rodrigo de Toledo, membro da banca, por aceitar participar da apresentação dessa dissertação de mestrado.

Agradeço a todos os meus amigos, presentes nos momentos bons e ruins.

Agradeço a todos que, direta ou indiretamente, influenciaram na pessoa que sou hoje e tornaram possível a realização desse sonho.

Agradeço também ao CNPq pelo apoio financeiro.

Meus mais sinceros agradecimentos a todos estes.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANÁLISE DE DINÂMICA DE SISTEMAS DO PROCESSO SCRUM

Felipe Madureira Fonseca

Junho/2016

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

A agilidade tem se mostrado um fator importante para o crescente sucesso no desenvolvimento de software. Dentro desse paradigma, o *Scrum* se mostra como o *framework* mais utilizado nas empresas. O estudo aprofundado de processos baseados nesse desenvolvimento poderá elevar a compreensão da dinâmica do *framework*, aumentando a produtividade, a eficácia e o valor gerado.

Nesse trabalho apresentamos modelos em System Dynamics para representar as diferentes etapas de um processo baseado nesse *framework*, buscando entender as particularidades, variáveis e relacionamentos. O modelo foi submetido a validação de especialistas além de ter sido submetido a dados reais de execução coletados a partir da ferramenta de gestão ágil *ScrumHalf*.

Foram realizados experimentos para compreender melhor a sensibilidade do processo em relação as suas variáveis e também foi explorado a capacidade preditiva do modelo verificando que com poucas iterações é possível obter boa acurácia da capacidade de entrega da equipe.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SYSTEM DYNAMICS ANALYSIS FOR SCRUM-BASED PROCESS

Felipe Madureira Fonseca

June/2016

Advisor: Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

Agile Software development has been a major factor in the growing success in software development. Within this paradigm, Scrum is shown as the most widely used framework in companies. The in-depth study of Scrum-based processes could increase the understanding of the framework's dynamics, increase productivity, efficiency and value generated.

In this work we present models in *System Dynamics* to represent different stages of a process based on this framework, seeking to understand the characteristics, variables and relationships. The model was subjected to validation specialists as well as being subjected to actual data collected run from scrumhalf agile management tool.

Experiments were conducted to better understand the sensitivity of the process regarding its variables and has also explored the model's predictive ability verifying that with few iterations is possible to obtain good accuracy of team deliverability capacity.

Sumário

Lista de Figuras

Lista de Tabelas

1.Introdução

1.1.Motivação

Um grande desafio de desenvolvimento de sistemas é que eles sejam entregues no prazo, dentro do custo, com qualidade e agregando valor ao cliente. Em 2012, 43% dos projetos foram entregues fora do prazo, acima do custo e/ou com menos funcionalidades que as necessárias. Além disso, 18% foram cancelados ou nunca utilizados. Isso significa que menos de 40% dos projetos foram entregues conforme o planejado caracterizando a área de desenvolvimento de software como uma área que possui muito espaço para melhoria.(“Chaos Manifesto”, 2013)

Com o problema em mente, duas vertentes (*Augustine’s Laws, Sixth Edition*, 1997)(Reel, 1999) buscaram entender as razões desse cenário. A linha mais atual traz à tona os problemas de gerenciamento, de comunicação entre os envolvidos e dificuldade em gerenciar as mudanças. Acredita-se, portanto, que solucionar esses problemas ou minimizá-los melhoraria a performance dos projetos de desenvolvimento de software.

Baseado nas dificuldades encontradas, em 2001 um grupo de especialistas em desenvolvimento de software se uniu para escrever o manifesto ágil. Esse documento se tornou um marco para o desenvolvimento de novas metodologias para criação de sistemas. Exposto nesse documento existem 4 fundamentos e 12 princípios que adotam uma inversão de valor em determinados quesitos quando comparados com os processos de desenvolvimento de software anteriormente praticados. (Beck et al., 2001)

Baseado nesse manifesto, diversas técnicas, *frameworks* e processos de desenvolvimentos foram criados, buscando atender aos princípios. Ao longo do tempo, o *framework Scrum* ganhou destaque, sendo usado por 58% das empresas que se utilizam dessa metodologia e 10% utilizam um híbrido entre o *Scrum* e o XP, totalizando uma adoção de ⅔ das empresas (“10th Annual State of Agile™ Report”, 2015).

O panorama dos projetos de software melhorou significativamente, como mostram diversos estudos sobre o assunto ao longo do tempo. Esses estudos, além das estatísticas, discorrem sobre os motivos dessa melhora. Entre melhoria das ferramentas, otimizações, envolvimento do usuário está também a adoção de processos ágeis

(Version One, 2013). O interesse por essa área é crescente e diversos trabalhos foram publicados na área desde então.

Dado o cenário descrito, a busca por compreender melhor essa área e como o desenvolvimento ocorre dentro um processo baseado em *Scrum* é importante para conseguir ainda mais melhorias no desenvolvimento. Entretanto, muitas alterações de melhoria no processo e hipóteses só podem ser validadas ao final de um determinado período, fazendo com que não se saiba se a alteração aumentará ou diminuirá a eficiência do processo no momento de sua adoção. A validação rápida e eficiente dessas hipóteses traria ainda mais benefício para o desenvolvimento.

1.2.Solução proposta

Para alcançar o objetivo de validar rapidamente as hipóteses em busca de uma maior eficácia no uso das metodologias, muitos pesquisadores utilizam a técnica de simulação. Essa técnica tem como objetivo mimetizar o ambiente a ser estudado através de modelos, variáveis e relacionamentos entre eles sendo possível estimar o comportamento real esperado dado um determinado cenário. Assim sendo, as alterações podem ser realizadas nesse ambiente controlado e têm-se o resultado esperado que será obtido ao se aplicar a mesma no ambiente real.

Os estudos de software baseados em simulação não são novidades no meio acadêmico, sendo o trabalho mais marcante data do início dos anos 90 (Abdel-Hamid & Madnick, 1991). Desde então, diversos trabalhos têm sido publicados utilizando esse paradigma, com diferentes focos e variadas técnicas.

O objetivo dessa dissertação é apresentar modelos que representem as diferentes etapas do *framework Scrum*, como o desenvolvimento, planejamento e a revisão. Por motivos de coerência e facilidade de transferência do conhecimento, a dinâmica de sistemas foi escolhida para a representação e simulação, dado que se encontra presente em 47% dos trabalhos publicados até então, sendo, portanto, a linguagem mais popular no meio (França & Travassos, 2013).

Como maneira de validação estrutural do modelo, uma versão inicial do trabalho foi submetida a análise de especialistas na metodologia ágil e com vasta experiência no desenvolvimento de software e suas sugestões e alterações do modelo estão documentadas, trazendo como resultado o modelo final desse trabalho.

Além da análise subjetiva do trabalho, os modelos foram submetidos a uma base de dados real da execução, levando em conta os indicadores considerados como essenciais, as saídas reais foram comparadas com a saída esperada do modelo, verificando se a mesma se encontrava dentro do esperado e mostrando se a estimativa baseada nesse trabalho se fazia possível através dos modelos apresentados.

Por fim, foi realizado, em cima do modelo validado, uma análise de sensibilidade, para compreender melhor quais são os fatores que mais afetam a entrega de produto durante o processo de desenvolvimento baseado em *Scrum*. De posse desse experimento, é possível compreender melhor o que deve ser aprimorado em seu processo para que se consiga uma maior entrega de pontos ao final do desenvolvimento do produto.

1.3.Organização do trabalho

Esse trabalho está dividindo em 7 capítulos, incluindo o de introdução, que foi feita a motivação para a realização do trabalho e descrita a abordagem da solução abordada, além de um apêndice.

No capítulo 2 é realizado uma breve revisão sobre o que é agilidade, em especial o *framework Scrum* e uma pesquisa sobre trabalhos relacionados ao que está sendo apresentado.

No capítulo 3 são apresentados os modelos propostos de maneira ampla e também de maneira detalhada com a finalidade de explicar os detalhes de sua concepção. Além disso, é também apresentada a validação estrutural feita com especialistas.

No capítulo 4 é feita uma contextualização dos dados coletados, dos indicadores propostos, de correlações encontradas que não foram utilizadas no experimento e uma breve discussão sobre comparação de produtividade entre equipes ágeis.

No capítulo 5 experimentos são realizados para compreender a sensibilidade do modelo aos indicadores e os resultados obtidos são discutidos.

No capítulo 6 temos a conclusão do trabalho e sugestões de trabalhos futuros a serem realizados

As referências bibliográficas se encontram no capítulo 7.

No apêndice são encontrados os dados de todos os projetos avaliados e o resultado da execução de todos os experimentos.

2.Referencial Teórico

2.1.Agilidade

Como visto anteriormente, a taxa de fracasso nos projetos de desenvolvimento de software estava muito alta. Nesse contexto, um grupo de especialista que posteriormente passou a ser conhecido como a Aliança Ágil, elaborou um manifesto (“Manifesto for Agile Software Development”, 2001). Esse documento foi então considerado como o marco inicial do movimento ágil, apesar dos métodos e práticas já existirem e serem utilizadas antes dessa data (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). Nesse manifesto, estão apresentados os quatro valores que devem ser os guias para a mentalidade de um agilista. São eles:

- Indivíduos e interações mais que processos e ferramentas;
- Software funcionando mais que documentação abrangente;
- Colaboração do cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

A agilidade entende que, apesar dos itens mencionados na direita possuírem valor, os da esquerda são mais valorizados. Além dessa mudança de visão, o Manifesto Ágil também apresenta doze princípios (“Principles behind the Agile Manifesto”, 2001):

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
- Mudanças de requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.

- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
- O método mais eficiente e eficaz de transmitir informação para e entre uma equipe de desenvolvimento é através de conversa face a face.
- Software funcionando é a medida primária de progresso.
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
- Contínua atenção à excelência técnica e bom design aumenta a agilidade.
- Simplicidade - a arte de maximizar a quantidade do trabalho não realizado - é essencial.
- As melhores arquiteturas, requisitos e projetos emergem de equipes auto organizáveis.
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então sintoniza e ajusta o seu comportamento de acordo.

A partir desse momento, os processos, técnicas, ferramentas ou metodologias que atendiam ou buscavam atender alguns dos princípios e compartilhavam dos valores estabelecidos nesse documento passaram a ser conhecidas como ágeis e, os demais métodos, passaram a ser considerados como tradicionais. De acordo com *Highsmith* (*Agile Software Development Ecosystems*, 2002), a origem da diferença entre ágil e tradicional está na forma de lidar com as incertezas em relação as funcionalidades necessárias para o software, principalmente no início do projeto.

Enquanto os chamados métodos tradicionais buscam prever, os ágeis buscam se adaptar (Murugaiyan & Balaji, 2012). Por causa de sua natureza iterativa e o emprego de ciclos contínuos e curtos de desenvolvimento, implementação e testes a equipe de desenvolvimento pode alterar o curso do trabalho e se adaptar aos novos cenários e exigências que surgem ao longo do projeto (Highsmith & Cockburn, 2001). Graças a essa visão, os *softwares* produzidos se tornam mais enxutos e evitam funcionalidades superficiais e focando apenas no essencial (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

2.2.Scrum

Por se tratar do *framework* de estudo nesse trabalho, se faz necessário um maior detalhamento das etapas, papéis, características, artefatos e filosofias dessa maneira de se realizar o desenvolvimento de software. O ambiente em que os projetos são executados costumam ser complexos, tendo seus requisitos em uma região de média ou baixa certeza acompanhada de média ou baixa concordância da necessidade (*Agile Project Management with Scrum*, 2004). O desenvolvimento tradicional possui seu custo de retrabalho aumentado em cada etapa do seu ciclo de vida: um erro identificado na fase de levantamento de requisitos pode custar até 100 vezes menos que a identificação do mesmo na etapa de manutenção ("Defect Prevention: Reducing Costs and Enhancing Quality", 2010). Nesse ambiente, de complexidade e mudança de requisitos aliado ao alto custo de retrabalho, a metodologia *Scrum* ganhou força e vem sendo usada em diversas empresas, ocupando 73% das empresas que utilizam paradigma ágil em seu desenvolvimento.

O *framework* é empírico e está em constante evolução tendo como seu principal documento de orientação o Guia do *Scrum*, disponível em diversas línguas ("Scrum Guide", 2013). Sua primeira versão foi lançada em 2010, tendo alterações incrementais em 2011 e 2013, mas o processo foi concebido no início dos anos 90 ("Scrum History", 2013). Nele é possível encontrar os papéis, fases, terminologia, artefatos e o funcionamento geral.

2.2.1.Papéis

Scrum Master

A figura do *Scrum Master* é normalmente caracterizada como um líder servo. Isso se caracteriza por ele ser uma figura de liderança e orientação para as práticas ágeis e teoria do *Scrum* e ao mesmo tempo alguém que serve ao time ajudando na facilitação das reuniões, nas resoluções dos impedimentos e em blindar o time de distrações externas e focar em aumentar a eficiência do mesmo.

Essa posição exige uma grande capacidade facilitação e habilidades interpessoais para gerenciar conflitos, identificar problemas e resolvê-los, buscando a harmonia e o aumento da capacidade de entrega do time. O conhecimento do *framework* e da agilidade deve ser notório visto que ele é o guardião dos princípios e dos valores e referência quando alguma questão controversa é colocada. Possui papel fundamento nas reuniões como facilitador, garantindo que os temas sejam debatidos de forma eficiente e eficaz, evitando fuga do tema e atrito entre as pessoas.

Existem diversas facetas relacionadas ao trabalho do *Scrum Master* dentro da equipe. Quando está a serviço do *Product Owner*, seu papel é de auxiliar ensinando o conceito de agilidade, auxiliando com técnicas para manuseio adequado do *Product Backlog* e ajudando a maximizar o valor. Ele ajuda o time de desenvolvimento facilitando a auto-organização e estimulando a multidisciplinaridade. Há também a responsabilidade de remover os impedimentos que interrompem o progresso do time. Por fim, perante a organização ele é o responsável explicar a importância da adoção do *framework* e planejar a implementação dentro da cultura organizacional existente.

Product Owner

A pessoa responsável por representar e defender os interesses e representar os *stakeholders* dentro do projeto no ambiente do *Scrum* é o *Product Owner* (PO). Sua missão é definir as características do produto e gerenciar o *Product Backlog*, adicionando, removendo, alterando e priorizando os itens que julgar necessários para produzir as funcionalidades mais importantes primeiro, considerando o valor e o retorno do investimento (ROI) que o mesmo trará ao cliente final.

Além da administração do *Product Backlog* e da representação dos interesses dos *stakeholders*, também cabe a essa pessoa validar o que foi desenvolvido durante a Sprint na reunião de revisão que será explicada posteriormente. É importante que ele mantenha o artefato que contém as histórias em lugar visível e que seja claro para que qualquer pessoa consiga entender.

A função de *Product Owner* deve ser desempenhada por uma única pessoa e não por um grupo. Essa recomendação está no guia e existe para que haja uma responsabilidade em cima do andamento do produto e que as decisões possam ser tomadas de forma mais fácil e ordenada (“Common Product Owner Traps - Scrum

Alliance", 2010). Apesar de não ser um comitê, o *Product Owner* poderá ser apoiado por múltiplas pessoas contanto que fique claro que o mesmo é soberano nas decisões que tangem o *Product Backlog* e que suas decisões sejam respeitadas.

Time de desenvolvimento

O time de desenvolvimento é a equipe de profissionais responsável pelo trabalho que realmente será entregue durante a *Sprint*.

A auto-organização e a multidisciplinaridade do time são características essenciais para uma boa formação. Ninguém fora do time, nem mesmo os outros papeis relacionados ao *Scrum* podem definir como o item do *Product Backlog* será transformado em incremento. Não existem títulos e os indivíduos deverão ser especialistas em determinados assuntos, mas também generalistas no desenvolvimento como um todo. A responsabilidade da entrega é do time inteiro e não de apenas um membro.

2.2.2.Eventos

Sprint

As *Sprints* são entendidas como ciclos de desenvolvimento que consistem em planejamento, desenvolvimento, revisão e retrospectiva. A duração dessa iteração deve ser definida previamente e consiste em uma janela de tempo fixa e inalterável e é conhecido como *timebox*. Caso a equipe não seja capaz de entregar o acordado no tempo previsto, uma negociação deve ocorrer entre time e *Product Owner* para ver se alguma funcionalidade ou ajuste poderá ser feito na próxima Sprint. Por outro lado, se as funcionalidades foram poucas, o PO e a equipe poderão, de comum acordo, inserir novas histórias. Essas práticas não são recomendadas, mas são consequências de um processo não ajustado. Com o passar do tempo e com a retrospectiva, a equipe tende a estabilizar sua capacidade de entrega e esses eventos passam a ser mais raros.

É importante que o *timebox* da *Sprint* seja preservado bem como não deverão haver intervalos entre elas. O fim de uma *Sprint* representa o início de outra. Esse ciclo

se mantém até que o cliente esteja satisfeito com o produto final ou que o projeto seja encerrado.

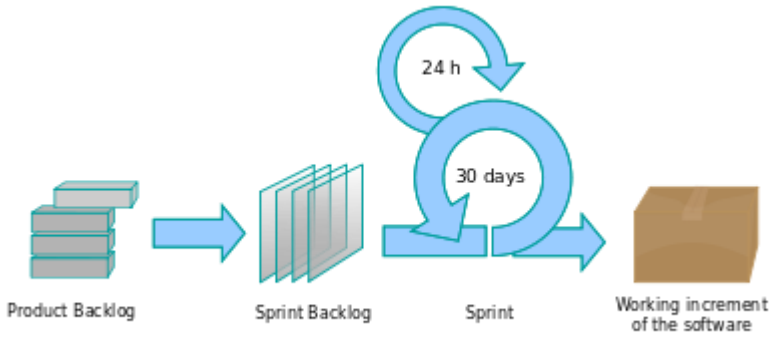

Figura 1. Fluxo do *Scrum* ("Scrum (software development)", 2016)

Planejamento da Sprint

Essa reunião se dá no início de cada Sprint e antes do guia de 2013 era dividida em duas partes ("Scrum Guide", 2013). Anteriormente na primeira etapa era negociado os itens que iriam compor o trabalho na *Sprint* e na segunda, a equipe realizava a divisão dos itens em tarefas que representavam o trabalho a ser realizado para concluir aquela funcionalidade solicitada.

Na versão mais moderna do guia, esse evento foi alterado para ser único, com a introdução formal da meta da *Sprint* nessa etapa. A reunião agora está focada em dois tópicos: primeiramente definir o que pode ser feito nessa iteração baseado na expectativa do *Product Owner* e na capacidade do time de desenvolvimento. Após esse tópico estabelecido, a equipe que define as tarefas para que a funcionalidade seja transformada em incremento. Nessa etapa, ao detalhar o trabalho necessário, a equipe poderá discutir com o *Product Owner* caso haja muito ou pouco trabalho, redefinindo os itens que foram escolhidos.

A etapa mais importante desse evento é a criação e compreensão da meta da *Sprint.* Ela representa um guia para todo o time do motivo da iteração e dos itens que estão sendo desenvolvidos. Caso a equipe perceba que o trabalho que está sendo desenvolvido está caminhando para não atender a meta, o PO deve ser notificado para que o *Sprint Backlog* seja alterado.

Reunião Diária

Ao longo da iteração, diariamente a equipe deve se reunir para monitorar o andamento do trabalho. Ao contrário do senso comum, essa reunião não é uma prestação de contas para o gerente ou para o chefe, visto que não há essa figura. Esse momento é da equipe e para a equipe, buscando manter a transparência e inspecionar continuamente o trabalho para que qualquer problema possa ser prontamente notificado e resolvido.

A reunião deve durar cerca de 15 minutos no máximo e por isso é feita preferencialmente com seus participantes em pé para que não se alongue. O fato de não estar sentado faz com que as pessoas se cansem e reunião não se estenda. O objetivo é responder a três perguntas para que o time fique sabendo como está andando o trabalho como um todo. A primeira pergunta consiste no trabalho que foi feito desde a última reunião para que o time entenda o que foi realizado no dia. A segunda consiste em comentar o que pretende ser realizado até a próxima reunião fazendo com que o time tenha uma expectativa do andamento do trabalho e até mesmo possa coordenar esforços para um melhor trabalho. Por fim, os membros devem expor quais os impedimentos que tiveram em seus trabalhos para que os demais fiquem cientes desses problemas.

Revisão da Sprint

Ao final do *timebox* planejado, o trabalho realizado ao longo da Sprint deve ser revisado pelo *Product Owner*. Poderão participar dessa reunião todos os interessados e afetados pelo projeto, mas cabe ao PO a palavra final em relação à aprovação ou reprovação do item.

Independentemente do estado em que se encontra, um item de backlog deve ser avaliado e apresentado ao PO para que o mesmo apresente seu veredicto em relação ao item. O incremento poderá ser aprovado, caso o *Product Owner* encare que a funcionalidade está de acordo com o seu interesse e irá gerar valor ao produto final ou reprovado, considerando que o item não atendeu as expectativas e necessidades da funcionalidade desejada.

Os itens serem apresentados para o PO mesmo que não concluídos é uma boa prática para que se possa ter uma ideia do trabalho realizado e do caminho que está sendo tomado naquela determinada funcionalidade. Essa prática evita um retrabalho futuro e dado o estado do trabalho poderá até mesmo ser aprovado dado que um item de *backlog* é negociável.

Retrospectiva

Essa é a etapa final do trabalho da *sprint* e não está diretamente ligada ao desenvolvimento do produto, mas sim com a melhoria contínua do processo de trabalho. Nessa reunião, a equipe é encorajada a realizar um retrospecto do trabalho realizado e buscar pontos de melhoria e consolidar boas práticas. Busca-se, neste momento, tornar o trabalho mais gratificante e eficaz, construindo uma equipe mais produtiva, maximizando a capacidade de entrega do time.

Apesar de não especificar como a reunião deve ser realizada, diversas técnicas foram desenvolvidas para tentar extrair o máximo de elementos a serem melhorados. O objetivo de ter uma variedade de abordagens é garantir o comprometimento e o interesse do time em participar e adaptar o método de trabalho, evitando que as ações levantadas não sejam adotadas ou que as pessoas não vejam mais valor nessa etapa.

2.2.3.Artefatos

Product Backlog

O *Product Backlog* é uma lista ordenada dos requisitos necessários para a finalização do produto e é a única fonte onde é possível encontrar esses requerimentos. A administração do conteúdo, da ordenação e da acessibilidade desse artefato é de responsabilidade total e única do *Product Owner*. A lista não é fixa e nunca é considerada acabada. A medida em que o projeto vai sendo executado a lista vai sendo alterada em conteúdo e ordenamento conforme as novas necessidades do produto e do ambiente para o qual ele está sendo desenvolvido.

Os itens que se encontram no topo da lista normalmente estão mais bem definidos, mais detalhados e mais claros sobre o que deve ser desenvolvido. Itens de

prioridade mais baixa costumam ser épicos (“Agile User Stories, Epics and Themes- Mike Cohn - Scrum Alliance”, 2014) contendo uma descrição em alto nível e sem muitos detalhes. A medida que esses itens sobem de prioridade, uma maior clareza se faz necessária para que a equipe possa desenvolvê-los.

Apesar do conteúdo e da ordenação ser de responsabilidade exclusiva do *Product Owner*, a estimativa do esforço do desenvolvimento de cada item deve ser dada pela equipe. O papel do *PO* nesse caso é ajudar a equipe a entender os requisitos e escolher o caminho entre as opções oferecidas pelo time, mas a palavra final do esforço deve ser dada por que iram realizar o trabalho.

Sprint Backlog

Uma vez que a iteração é iniciada, uma parte do *Product Backlog* é escolhida em acordo durante a etapa de planejamento e irá compor o *Sprint Backlog*. Esse artefato representa o que será trabalhado dentro da *Sprint*, além de um plano para entregar o incremento e atingir a meta. Essa lista também é ordenada, sendo que os itens mais acima representam os que possuem mais valor e a ordem deve ser respeitada.

O plano para a finalização dos itens é descrito em tarefas e deve ser visível e claro a todos do time. A medida que novo trabalho é julgado necessário pela equipe, novas tarefas são adicionadas e, se uma tarefa é considerada obsoleta é retirada desse artefato. Essas manobras são de exclusividade da equipe de desenvolvimento. Durante o *Daily Scrum*, a equipe discute o trabalho realizado e que será realizado em volta do *Sprint Backlog*.

A atualização do conjunto de histórias desse artefato durante o andamento da *Sprint* não é recomendada. Eventualmente, é possível que alguma história seja adicionada caso sobre tempo ou removida caso não faça mais sentido, desde que não interfira na meta planejada para essa iteração. No caso de a meta estar obsoleta, o recomendado é cancelar a *Sprint* e refazer todo o ciclo novamente.

A soma de todos os itens que foram concluídos e aceitos pelo *Product Owner* após a *Sprint* e o valor dos incrementos de todas as *Sprints* anteriores é chamado de Incremento. Esse incremento deve estar utilizável e de acordo com a definição de pronto mesmo que não vá incorporar o produto de forma imediata.

2.3.Modelagem e simulação em processo de desenvolvimento de software

A modelagem e simulação de processo de desenvolvimento de software não é uma área nova, sendo observado em uma revisão *quasi-sistematica* recente (França & Travassos, 2013) a existência de 108 artigos publicados focados em estudo baseados em simulação na área da engenharia de software, sendo o mais antigo datado de 1986 com o número de publicações crescendo a cada ano. O trabalho mais referenciado na área é o modelo proposto por Abdel-Hamid e Madnick em seu livro (Abdel-Hamid & Madnick, 1991). Ele aborda o desenvolvimento de software considerando 7 sub-sistemas, entre eles recursos humanos e testes. Esse trabalho se tornou referência para diversos trabalhos na área e popularizou a linguagem de dinâmica de sistemas para a simulação desse fim. Além dessa publicação, outras revisões da literatura foram encontradas nessa área.

Artigos que lidam com a agilidade também foram encontrados, apesar de não muito numerosos. Apesar do autor não extrapolar o assunto, acredita-se ser por causa da agilidade ser um domínio recente na engenharia de software e por falta de dados coletados nessa área, ao contrário da área de gerenciamento do projeto de software que possui práticas consolidadas e vasto material divulgado, além de ser a região do modelo mais difundido.

Inicialmente, para embasar o trabalho, é preciso entender como funciona a modelagem, seus benefícios, seus principais problemas e como ela é feita. De acordo com Anu Maria (Maria, 1997), modelagem é o processo de construir um modelo e esse, por sua vez, é uma representação da construção e trabalho de algum sistema de interesse. O modelo tende a ser uma visão parecida, mas simplificada do ambiente que ele representa. Um bom modelo deve possuir um bom balanceamento entre simplicidade e realismo e é recomendado que o mesmo seja construído e sua complexidade adicionada de forma iterativa.

Os modelos podem ser divididos entre mentais e explícitos. O primeiro tipo representa as percepções que um determinado indivíduo possui em relação às interações entre partes do sistema e o comportamento que ocorre devido a essas ligações. Esses

modelos, por estarem limitados a capacidade humana de gerenciar um grande número de fatores, são normalmente simples. A simplificação ocasionada por essa limitação pode gerar problemas na interpretação de determinados eventos e dificuldades em associar eventos e resultados de forma correta. Além disto, o principal problema reside no fato deste tipo de modelo existir no conhecimento tácito, dificultando sua transmissão e compartilhamento.

Ao ser transformado em uma linguagem que possa ser compartilhada com outras pessoas há a criação de um modelo explícito, representando o modelo mental daquele indivíduo sobre o tema. De posse desse registro, a pessoa poderá atualizar sua representação mental a partir dessa nova evidência. As decisões não são tomadas a partir dos modelos explícitos (Jay W. Forrester, 1993) mas servem para aprimorar o modelo mental do indivíduo. Do ponto de vista prático, os modelos explícitos possuem diversas vantagens como a validação e difusão do conhecimento, fazendo com que o mesmo possa ser aprimorado, revisado e compartilhado entre os interessados.

A técnica mais utilizada para estudos de software baseados em simulação é a dinâmica de sistemas (Jay Wright Forrester, 1961). Em uma recente revisão da literatura a técnica foi encontrada em 51 dos 108 artigos e foi utilizada em 37 dos 88 modelos de simulação identificados nos trabalhos (França & Travassos, 2013). Sua popularidade é confirmada ainda por outra revisão focada na área de simulação de processo de software, sendo a segunda mais popular a utilização de eventos discretos (Zhang, Kitchenham, & Pfahl, 2008).

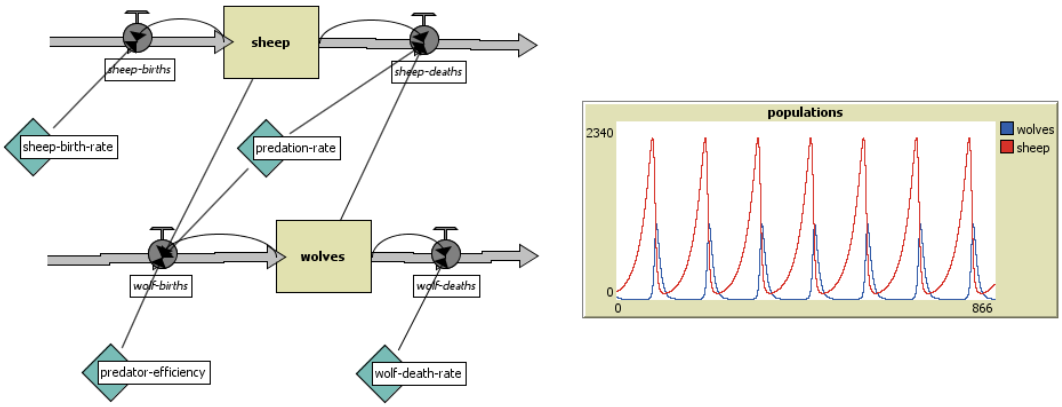

Figura 2. Exemplo de modelo em dinâmica de sistemas (Wilensky, U, 1999)

Na figura 2 podemos observar um modelo básico e didático que é muito utilizado para explicar as características da dinâmica de sistemas. Nele podemos observar dois repositórios: lobos e ovelhas. Cada um desses possui uma vazão de entrada, a partir do conceito de nascimento e de saída, representando a morte de indivíduos. A medida em que a população de lobo cresce, a taxa de mortalidade de ovelhas aumenta, devido ao fator de predação. Com menos ovelhas, a taxa de nascimento desses indivíduos também diminui, influenciando na taxa de nascimento de lobos já que haverá escassez de alimentos. Dessa forma, temos um *feedback* representado e mapeado no modelo.

A dinâmica de sistemas possui a capacidade de representar comportamentos endógenos, sendo que o modelo deve ser construído a fim de representar a estrutura do sistema e ao ser simulado temos a demonstração do seu comportamento (Jay W. Forrester, 1993). Partindo do paradigma da iteração e do *feedback* relacionado ao *Product Backlog* podemos compreender como o comportamento dos fatores internos influenciam no comportamento visível do processo.

Uma outra característica que é importante observar nessa técnica é a possibilidade de compreender causas e consequências distantes no tempo. Em sistemas complexos, como o desenvolvimento de um *software*, decisões e combinados só podem ter sua eficácia aferida após um determinado tempo de experiência. Por se tratar de um sistema de diversas variáveis, o passar do tempo dificulta a compreender e perceber a real origem de problemas e consequências dos atos dado que explicações conflitantes podem surgir (Karl E. Weick, 2015).

Por fim a modelagem utilizando essas técnicas está associada apenas a equações ou informações numéricas. Apesar das fórmulas serem o que efetivamente é utilizado para o cálculo dos valores, elas podem ser criadas a partir do modelo criado pelo projetista. Ao invés de pensar na matemática, o foco poderá ser na estrutura e no relacionamento das variáveis permitindo representar comportamentos genéricos e situações de forma mais prática e simples (Merrill, Collofello, Urban, & Faltz, 1996).

Dado todo esse cenário, utilizar a dinâmica de sistemas se mostra perfeitamente adequado para a situação. A técnica de eventos discretos, no entanto, não está descartada para esse tipo de trabalho (Ross, 1990). A mesma também possui capacidade de representar processo de desenvolvimento e é a segunda mais utilizada para esse fim conforme mostrado. Para esse trabalho focado em *Scrum* e com poucos artigos desse

tipo na área, a dinâmica de sistemas se mostra mais adequada por ter mais publicações na literatura, ampliando a divulgação e a capacidade de alcance do trabalho realizado.

2.4.Trabalhos Relacionados

Conforme dito anteriormente, o trabalho pioneiro nessa área, mesmo que não esteja ligado ao desenvolvimento ágil diretamente é o livro do Abdel-Hamid e Madnick (Abdel-Hamid & Madnick, 1991). Mesmo sem tratar diretamente de agilidade, seus diversos subsistemas e suposições são utilizadas como base para diversos trabalhos como veremos nessa seção. A etapa de qualidade, por exemplo, é utilizada em uma pesquisa de agilidade para entender o impacto da duração de uma iteração com a quantidade de defeitos e o esforço total do trabalho para o desenvolvimento de um sistema.

A utilização de dinâmica de sistemas no campo de metodologia ágil está muito ligada a melhor entender como os processos funcionam e avaliar a efetividade desse tipo de abordagem para o desenvolvimento de software (Cocco, Mannaro, Concas, & Marchesi, 2011). A maioria dos trabalhos dessa área está ligada a práticas da Programação Extrema (Extreme programming) ou apenas a métodos ágeis genéricos. Mesmo sendo a metodologia mais popular, trabalhos puramente sobre o *Scrum* são praticamente inexistentes (Cocco et al., 2011).

Alguns trabalhos da área buscam realizar a comparação entre o desenvolvimento tradicional com o novo paradigma utilizando diferentes indicadores. Custo, tempo, produto entregue e densidade de defeitos são números comuns a serem utilizados para comparar as duas abordagens. Um trabalho recente (Cocco et al., 2011) propõe um modelo único para comparar a performance entre 3 diferentes abordagens para um mesmo cenário. Diferentes variáveis controlam os atrasos e as taxas para simular a maneira como cada processo trabalha. O modelo é bem simplificado e parte de suposições que limitam a capacidade do mesmo mas deve ser encarado como um ponto de partida a ser considerado para estudos futuros. Por se tratar de um trabalho inicial, sua conclusão se baseia apenas nos atrasos e quanto produto foi entregue em um determinado tempo, evidenciando a característica iterativa do *Scrum* em relação ao modelo em cascata tradicional.

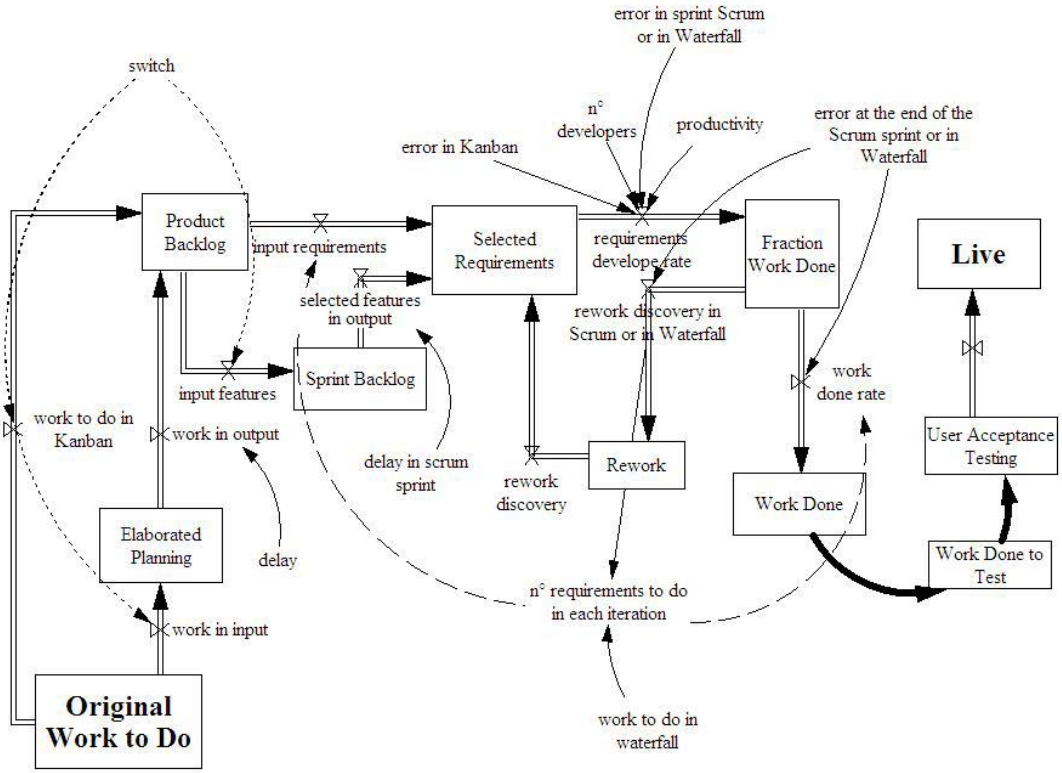

Figura 3. Modelo simplifcado de comparação entre abordagens (Cocco et al., 2011)

Um conceito muito estudado pelos trabalhos da área e que está presente nos princípios do manifesto ágil é a entrega contínua de software. Os benefícios e problemas gerados a partir desse conceito são alvos de estudo constante na área, principalmente para uma comparação com o desenvolvimento tradicional que não utiliza esse paradigma. Um trabalho recente da área aborda diretamente esse assunto (Akerele, Ramachandran, & Dixon, 2013). Entretanto, nenhum modelo é fornecido, sendo apenas abordado como a pesquisa será realizada.

No contexto da Programação Extrema, a prática mais estudada utilizando a abordagem de simulação parece ser a programação em par. O Pair Programming pode ser definido como dois programadores trabalhando colaborativamente no mesmo algoritmo, design ou tarefa, sentados lado a lado em um computador (“Pair Programming”, 1999). Essa prática foi apontada como fator de sucesso em 17,8% dos projetos que adotaram essa abordagem em seus projetos baseados em programação extrema (Cockburn & Williams, 2001) e considerada um método eficaz de

desenvolvimento de software (*Constantine on Peopleware*, 1995)(Cockburn & Williams, 2001).

Wu, Minghui and Hui Yan (Wu & Yan, 2009) propõe um modelo e realizam um estudo sobre o impacto da utilização da técnica de pair programming em diferentes cenários. O modelo utilizado para a simulação da XP foi evoluído a partir do trabalho de Abdel-Hamid e simplificado para condizer com a leveza do método em relação ao processo tradicional mapeado pelo trabalho original. Muitos subsistemas descritos foram simplificados para a representação.

Outro trabalho abordando a técnica de Pair Programming foi publicado em 2003 e estuda questões mais profundas da técnica, como a compatibilidade entre os desenvolvedores, um fator muito importante dado que algumas pessoas trabalham melhor utilizando essa técnica do que outras, a quantidade e o tempo de refatorações realizadas. Esse trabalho busca entender como um projeto pode ser designado para funcionar melhor utilizando essa técnica e em quais cenários é ou não vantajoso a prática (Lui & Chan, 2003).

Posteriormente, um modelo mais robusto e um estudo mais completo foi produzido por Glaiel et al denominado de Agile Project Dynamics (APD)("Agile Project Dynamics", 2012). Nele, os autores mapeiam o que chamam de "Genoma ágil" considerando 7 aspectos que seriam análogos aos genes existentes. Um processo desse paradigma, por sua vez, seria formado por um ou mais componentes, gerando uma variedade grande de possibilidades. Para os experimentos, o autor altera os genes para simular diferentes metodologias ágeis. A figura abaixo mostra uma visão simplificada do modelo proposto.

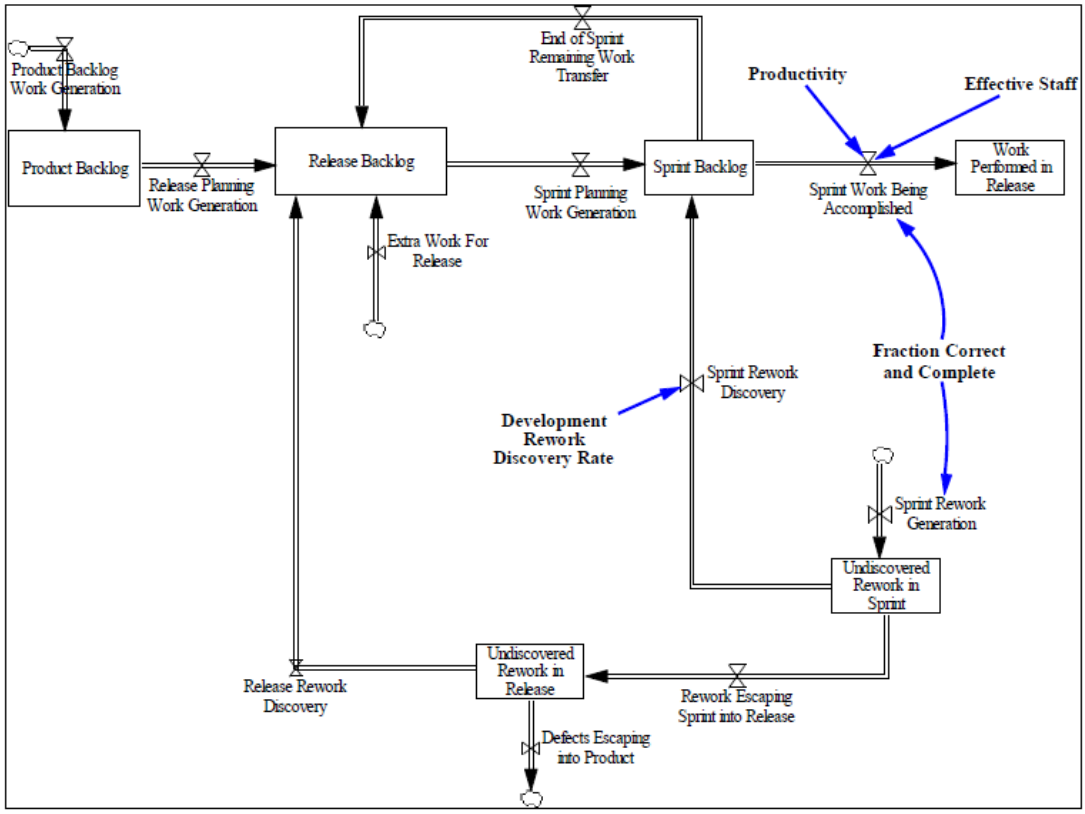

Figura 4. Versão simplificada do modelo APD (“Agile Project Dynamics”, 2012)

O modelo proposto novamente surgiu apoiado na modelagem do processo tradicional do artigo referência na área. Junto com outros estudos e diferentes abordagens, um modelo genérico foi criado e, a partir da ativação ou desativação de determinadas alavancas que representam os genes, diferentes abordagens podem ser simuladas a partir do modelo.

Nesse trabalho, a agilidade é estudada como um todo e um modelo completo é criado, possibilitando a adoção ou não de algumas técnicas. Essa flexibilidade permite ao modelo se adaptar a diferentes práticas que os agilistas podem praticar, permitindo um estudo mais profundo do impacto da adoção e refinamento de determinadas etapas do processo, independente da metodologia utilizada

2.5.Considerações finais

No trabalho buscamos aprofundar o modelo para as especificidades do *Scrum*, detalhando as fases e as particularidades de cada etapa desse *framework*. Dessa

maneira, podemos compreender melhor como funciona a dinâmica do mesmo em sua essência e podemos efetuar análise e otimizações que não são possíveis em modelos genéricos como os trabalhos anteriores. Um exemplo dessa diferença está ligado ao retrabalho gerado pela entrega da sprint: em um modelo genérico, como observamos anteriormente, apenas um retrabalho seria gerado e devolvido para o *Product Backlog*, o que, no Scrum, esse fenômeno pode ter sido causado por um impedimento que surgiu em uma história que a impediu de ser finalizada e não atingiu o conceito de pronto ou pela reprovação da história que, apesar de ter tido seu desenvolvimento completo, não atingiu as expectativas do *Product Owner*. São dois cenários com diferentes causas e devem ser tratados de forma diferente para a solução. Essas especificidades são importantes e serão tratadas ao longo do trabalho com mais detalhes.

3.Modelos

3.1.Considerações iniciais e definições

Nessa parte do texto iremos estabelecer algumas definições e conceitos que serão adotados para a modelagem final do trabalho. Essa etapa é importante para que o modelo possa ser compreendido corretamente e, assim, expandido para novas situações sem comprometer o trabalho.

3.1.1.Impedimentos

O tópico de impedimento é tratado com superficialidade no guia do *Scrum*. Portanto, sua real definição fica à mercê da prática e do entendimento de cada profissional quanto a utilizar esse tipo de sinalização. Inicialmente, para a definição de uma palavra buscamos a mesma no dicionário. Impedimento está ligado diretamente a "Aquilo que impede" também podendo ser encarado como um obstáculo, um estorvo ("impedimento: Dicionário Português Online: Moderno Dicionário da Língua Portuguesa", [s.d.]). Esse entendimento, no entanto, continua deixando a interpretação em aberto: como o guia não é claro, então, o que afinal está sendo impedido? É possível encarar diversos pontos que podem estar sofrendo com obstáculos.

Um entendimento possível é que o que está sendo impedido é a capacidade do time como um todo entregar o sistema solicitado. Por essa visão, um impedimento é qualquer coisa que impeça o time de entregar valor, desde escritório muito barulhento a falta de ferramentas e requisitos necessários para o desenvolvimento correto ("Impediments are holding back the team", 2011). Até mesmo coisas que não estão no controle, como desastres naturais, devem ser levantadas como impedimentos adotando essa visão ("What are impediments?", 2013).

Por outro lado, é possível encarar como impedimentos os que são apresentados por algumas pessoas como bloqueadores. Esse termo, apesar de não estar no guia, pode ser entendido como um tipo específico de impedimento que não está apenas limitando, mas sim bloqueando o andamento da entrega de valor ("Impediments vs. blockers",

2016). Pensando no caso em que o objeto que está enfrentando um obstáculo é a própria entrega de valor e não a capacidade do time, esses conceitos acabam sendo vistos como sinônimos. Independentemente da interpretação, esse é um caso mais grave que deve ser abordado de maneira mais urgente, já que o trabalho com o qual a equipe se comprometeu na *sprint* está ameaçado diretamente.

Outra discussão que surgiu ao longo da pesquisa é em que momento um item de *backlog* está no estado de impedido. Nesse caso, foram encontradas duas possibilidades possíveis: a história está efetivamente impedida quando o impedimento é vislumbrado pela equipe ou somente no momento em que não seja possível realizar o andamento da mesma. No primeiro caso, o desenvolvimento da história poderia continuar normalmente enquanto seu estado estaria sinalizando impedimento. No segundo, somente no momento em que nenhum trabalho possa ser realizado nesse item o estado seria alterado. Nesse caso, de acordo com a maneira de trabalho da equipe que foi realizado o estudo e conceito de história impedida dado pelo sistema, o primeiro conceito é mais adequado. Em suma, para o conceito estabelecido, história impedida é entendida como uma história que não pode ser finalizada, independentemente de poder ser desenvolvida ou não.

Por último, foi preciso definir a transição de estados para estruturar o modelo. Conforme discutido, uma história impedida poderá continuar com seu desenvolvimento caso a interrupção não seja total em relação ao trabalho restante. A questão que cabe agora é se uma história não iniciada poderia ser alterada para um estado de impedimento sem antes receber o estado de em execução. Ao longo das entrevistas isso foi um tema de grande debate com duas visões diferentes. A primeira delas argumentava que, apesar de não iniciada, a existência de um impedimento fazia com que essa história tivesse o estado de impedida e, a resolução desse impedimento, consequentemente, faria o item retornar ao estado de não iniciado. Em outro ponto de vista, apesar do impedimento existir naquela história, se o desenvolvimento não estiver sido iniciado, o estado se manteria inalterado. Nessa visão, apenas uma história com o desenvolvimento iniciado poderia ser considerada impedida (Tomaz, L, 2015). Caso contrário, seu estado permaneceria não iniciada mesmo que o *Scrum Master* já estivesse trabalhando na resolução. Nesse caso, somente histórias iniciadas poderiam se tornar impedidas e, portanto, não haveria transição entre histórias não iniciadas e impedidas.

Para ilustrar as transições de estado e as possibilidades de andamento, a figura 3 ilustra o diagrama de estado pensados para um item de *backlog* em desenvolvimento dentro de uma *sprint*.

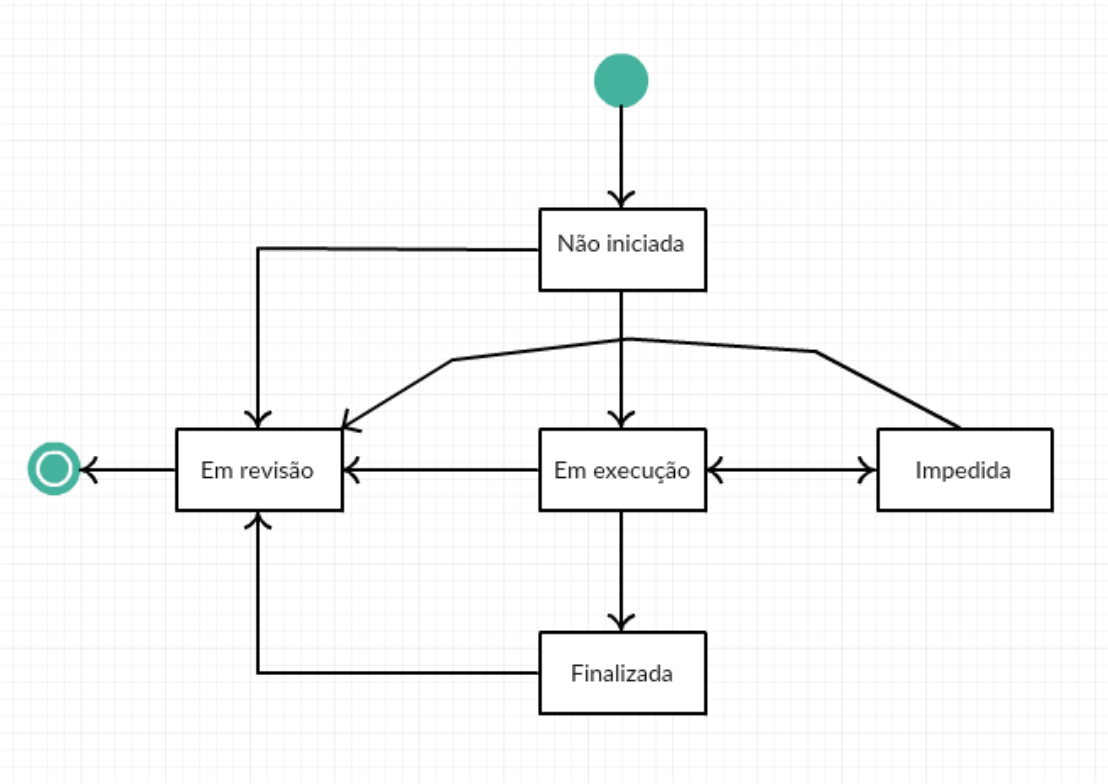

Figura 5. Diagrama de estados de um item de *backlog para o modelo*

Conforme descrito, qualquer item de *backlog* do produto completado e pronto, ou seja, no estado de Finalizado, deverá ser revisado. Todos os demais são retornados ao *Product Backlog*, onde serão analisados novamente pela equipe para receber uma nova estimativa e pelo *Product Owner*, para receber uma nova prioridade ou até mesmo ser excluído, a critério do dono do produto. Para o modelo, o cancelamento de uma *Sprint* pode ser encarado como uma finalização abrupta.

A partir do diagrama de estados, da experiência obtida na execução dos projetos e das entrevistas com especialistas, foi criado um modelo simulável que representa a dinâmica de uma iteração.

3.2.Modelo completo

Como parte desse trabalho o *framework Scrum* foi extensivamente estudado e suas etapas e os eventos foram identificados para que um modelo representando um processo de desenvolvimento baseado em *Scrum* pudesse ser apresentado. O modelo se apresenta separado em 3 áreas diferentes identificadas e detalhadas mais à frente nesse capítulo.

As áreas se relacionam a partir de fluxo de pontos de histórias entre elas da seguinte forma: os pontos que foram inseridos no planejamento são transferidos para o desenvolvimento da Sprint a partir da reunião de planejamento, que preenche o *Sprint Backlog* a partir da velocidade observada. Ao final do *timebox*, os pontos são transferidos para a revisão onde serão aprovados e comporão o produto final ou reprovados e voltarão para a etapa de planejamento.

O *timebox* está ligado às duas transferências na medida em que determina o início e o fim de uma iteração. A velocidade é responsável por orientar a quantidade de pontos que serão inseridos no desenvolvimento. As áreas, variáveis e comportamentos descritos podem ser encontrados na figura 4 abaixo.

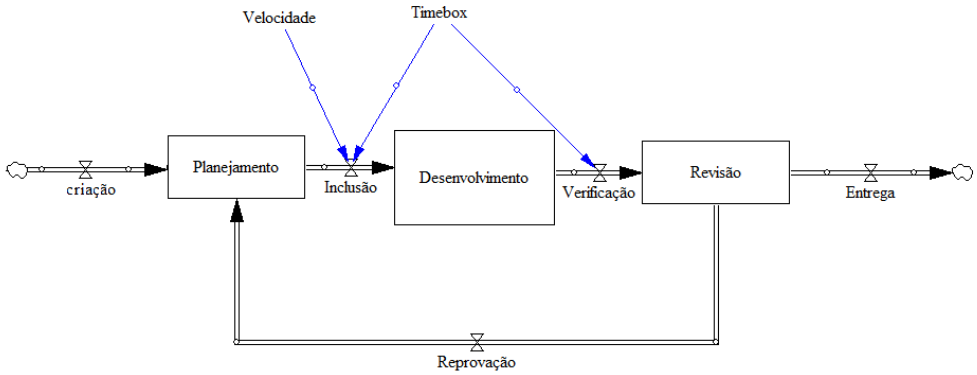

Figura 6. Versão simplificada do modelo proposto

Dentro de cada área existe mais especificidades e variáveis que irão controlar o comportamento de cada etapa. Ao longo desse capítulo iremos descrever em mais detalhes cada área mencionada nessa etapa e poderá ser compreendido mais claramente o modelo proposto.

3.3.Modelo do Planejamento

A dinâmica do *product backlog* está relacionada ao escopo do sistema de software a ser desenvolvido. Essa responsabilidade, no *Scrum*, é atribuída ao PO, sendo ele o soberano em adição, exclusão e modificação das histórias que ali existem. Apesar dos demais membros não possuírem poder para realizar alterações nesse artefato, os stakeholders podem opinar e sugerir melhorias e alterações dos itens que ali estão descritos. Para modelar esse comportamento, foi sugerido o seguinte modelo:

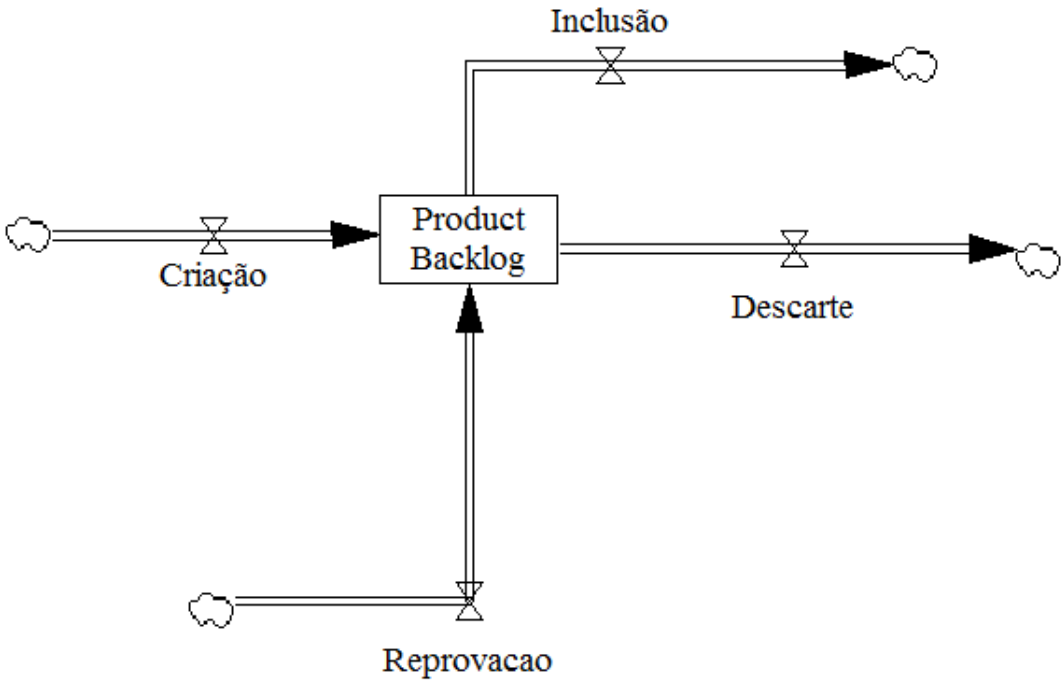

Figura 7. Modelo representando a dinâmica do *Product Backlog*

O artefato do *Product Backlog* é alimentado através da criação, fazendo com que pontos de histórias sejam inseridos a partir do aval do *Product Owner*. Não necessariamente esse requisito surgiu diretamente do dono do produto: os *stakeholders* podem e devem opinar sobre as necessidades, mas elas só irão compor o artefato após a autorização do *PO*. Inicialmente, como será discutido no item 3.7, o modelo continha outro reservatório dedicado a sugestões. Por não ser tratar do processo genérico e sim

de uma especialização, foi retirado do modelo ficando apenas como uma sugestão de extensão.

A medida em que o contexto muda e a necessidade do negócio também, pontos são retirados do *Product Backlog*. Como visto, é um artefato dinâmico e elástico, tendo seu conteúdo constantemente revisto e ajustado para atender ao cliente. Nesse ponto, o modelo não trata da noção de prioridade entre os itens, visto que a simulação se baseia na quantidade de pontos entregues e a prioridade está relacionado ao valor do item e ao retorno que o mesmo dará ao investimento.

A taxa de inclusão em *sprint* está diretamente relacionada ao timebox e a velocidade da equipe. Inserir itens em uma iteração já iniciada não é recomendado pelo *Scrum* e por isso é uma prática que tenta ser evitada. Apesar dos esforços, eventualmente se faz necessário a adição fora do timebox e isso pretende ser representado na taxa a ser calculada. Além disso, a relação entre velocidade e a inclusão de itens na sprint está relacionada com a etapa de planejamento da iteração, onde há uma negociação entre PO e equipe para se definir quanto esforço será alocado na Sprint, normalmente tendendo a estar nas proximidades da capacidade de desenvolvimento médio do time. A alteração desses valores pode existir e se deve a diversos fatores externos, como pressão para concluir o prazo final do projeto, ausência de membros da equipe desenvolvedora, incerteza sobre a meta a ser alcançada ou até mesmo um acordo informal entre os papéis. Esses eventos não estão representados no modelo, mas podem justificar alterações abruptas de valores.

3.4.Modelo do Desenvolvimento da História

Os itens agrupados no *Sprint Backlog* que foram selecionados para serem entregues ao final da iteração deve ter seu trabalho e progresso monitorados. Ao longo da mesma, um comportamento foi observado e representado no modelo.

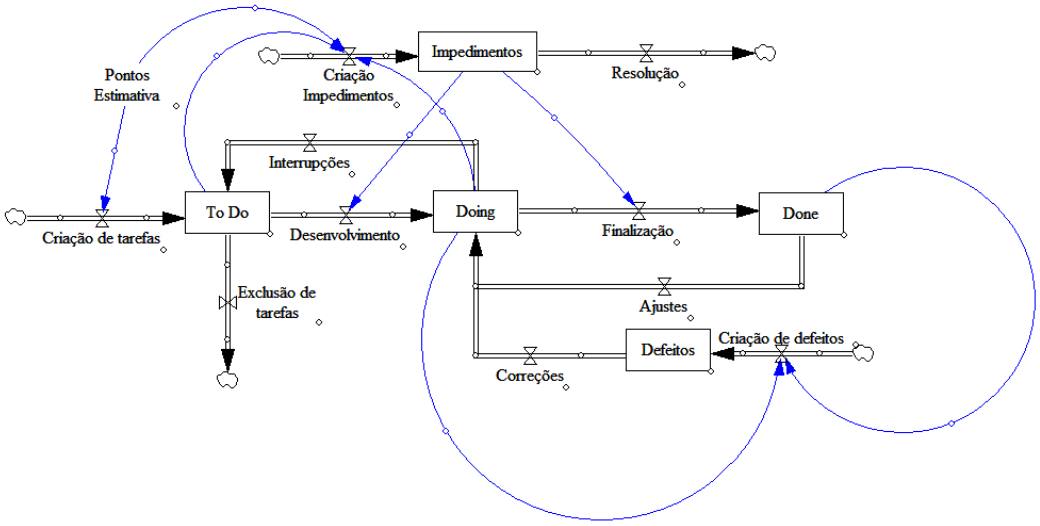

Figura 8. Modelo do desenvolvimento de uma história

Para o início do desenvolvimento, se faz necessário quebrar a história em atividades que serão feitas, denominadas tarefas. Essa etapa é feita pela equipe de desenvolvimento com a finalidade de explicitar os passos necessários para a finalização daquele entregável e para monitorar o andamento. Apesar de ter o seu momento de realização durante o planejamento, a criação de tarefas é atemporal e pode ser realizada a qualquer momento da iteração. Inicialmente, as tarefas são criadas e armazenadas em um repositório de “A Fazer”, representando o trabalho restante a ser desenvolvido.

A medida em que o trabalho é iniciado, as tarefas vão sendo transferida para o próximo reservatório, conhecido como “Fazendo”. Esse repositório é uma simplificação e poderá ser estendido conforme a necessidade do projeto. Diferentes fases podem ser incorporadas para representar estágios de completude da tarefa. O time de desenvolvimento pode encarar como necessária a distinção entre histórias codificadas e testadas e, para isso, expandir essa fase em duas. O modelo segue o caso mais simples e permite a extensão do mesmo.

Finalmente, ao terminar um desenvolvimento de uma tarefa, a mesma é transferida para o reservatório de “Finalizado”. Essa movimentação representa que aquela atividade foi realizada até sua completude. Diferentes critérios podem ser adotados para um trabalho ser considerado finalizado, porém, independentemente dos mesmos, essa etapa demonstra o fim dessa tarefa.

Em algumas ocasiões, as tarefas podem retroceder fases. A tarefa poderá ser interrompida por algum evento externo e o desenvolvedor encontrar como melhor solução retorná-la a fase de “A fazer”. Diversos motivos podem ocasionar esse evento, como por exemplo uma melhor análise da atividade e um eventual descobrimento de falta de *expertise* ou de material disponível para o trabalho. Também é possível que a tarefa retorne a fase de desenvolvimento para algum determinado ajuste fino para que a mesma seja finalizada. Por esses motivos, há um fluxo contrário às fases, sendo considero eventos de exceção.

Durante a iteração, podem ser criados ou encontrados defeitos na história que está sendo desenvolvida. Idealmente esses problemas são identificados de maneira diferente das tarefas normais para fins de medição. Esses defeitos são sinalizados e resolvidos pela equipe de desenvolvimento como qualquer outra atividade. Quando encontrados dentro da Sprint, esses eventos são corrigidos normalmente e não são considerados para a contagem de densidade de defeitos.

Ao longo do desenvolvimento, existem eventos que impedem o início ou a finalização de uma determinada tarefa ou história. Esses eventos são tratados no guia *Scrum* como impedimentos e são responsabilidade do *Scrum Master* resolvê-los. Os fatores que propiciam o surgimento de impedimentos não estão contemplados no modelo visto que são das mais diversas naturezas e fogem ao escopo do modelo. Esses impedimentos refletem na taxa de desenvolvimento, impedindo que uma tarefa seja iniciada, e na taxa de finalização, inviabilizando terminar o trabalho.

É razoável pensar que a complexidade da história terá influência no número de tarefas e serem criadas e dos impedimentos que surgirão. Isso será verificado posteriormente com a análise dos dados obtidos pelas execuções de projetos, mas consta no modelo como hipótese. Itens com complexidade mais alta tendem a ter mais tarefas a serem executadas e, consequentemente, aumenta a possibilidade de surgimento de defeitos e de impedimentos.

3.5.Modelo do Desenvolvimento da Sprint

Ao contrário do modelo do desenvolvimento da história, o foco da Sprint está relacionado às histórias e não as tarefas. Esse diagrama busca verificar o comportamento da iteração que é a forma que o *Scrum* prega para a entrega do trabalho.

O bom andamento de uma *sprint* é o objetivo do um bom processo de desenvolvimento baseado nesse *framework*. Planejar com valor, cumprir meta e entregar com qualidade os itens selecionados é essencial e premissas do *framework*. O modelo a seguir representa a dinâmica dos itens durante o desenvolvimento da iteração.

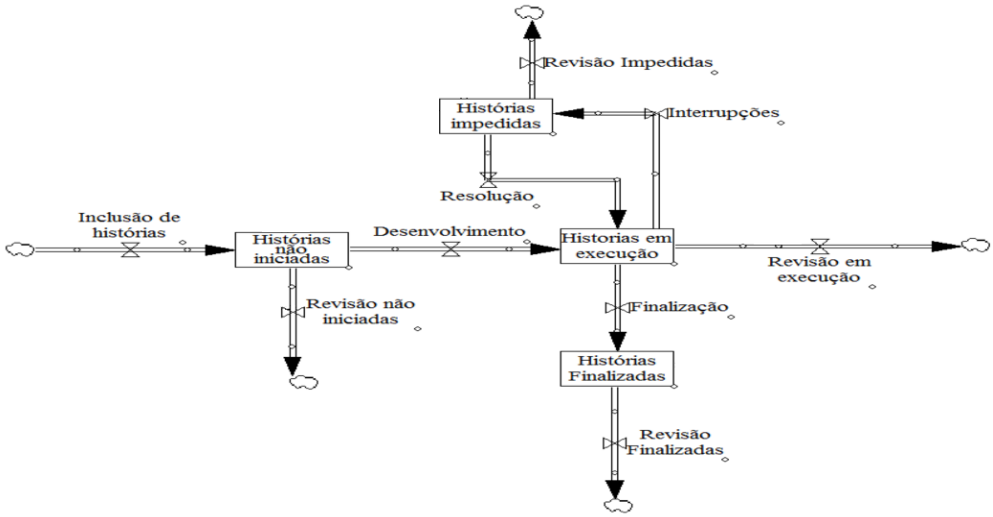

Figura 9. Modelo representando o andamento de uma *sprint*

A Inclusão de histórias é a mesma vazão apresentada no modelo do *Product Backlog*, denominada de "Inclusão em *sprint*". A entrada de itens é realizada primeiramente durante a fase de planejamento 1 da iteração. A partir daí, excepcionalmente, é possível adicionar ou remover itens sem que afete a meta.

O desenvolvimento é a taxa em que as histórias são iniciadas. Conforme vimos nos demais modelos, o conceito de em execução é estabelecido com o início da realização de alguma tarefa planejada para o item em questão. O estado da história é alterado caso todas as tarefas estejam na fase de finalizado, sendo assim considerado uma história finalizada ou caso todas as tarefas se concentrem na fase de A fazer, sendo assim uma história não iniciada.

As histórias impedidas representam um estado anormal de um item. Esse caso é descrito como uma história que possui algum obstáculo que impeça a execução de qualquer trabalho restante no item, ou seja, não pode ser finalizada. Um caso onde a história possui impedimentos mas possui trabalho que pode ser executado independentemente da resolução do mesmo é considerado uma história em execução para o modelo. Uma história impedida é o caso de existir um impedimento para a história ou todas tarefas restante possuirem um impedimento associado. Dessa forma,

são itens que não poderão ter seu trabalho terminado até que a resolução do problema seja efetuada.

Apesar de haver semelhança entre o andamento da *sprint* e o desenvolvimento de uma história, a granularidade e o resultado do modelo são bastante distintos. Enquanto esse modelo abrange histórias o outro é focado em tarefas, que são subdivisões do trabalho do item e, portanto, de menor granularidade. Além disso, o bom andamento individual de uma história não necessariamente implica em uma *sprint* de sucesso.

3.6.Modelo da Revisão

Na data programada, as histórias que foram incluídas na *sprint* serão submetidas a avaliação do *product owner*. Esse evento é conhecido como revisão e deve ser realizado ao final de cada *sprint*. As histórias deverão ser aceitas ou reprovadas de acordo com o critério de pronto estabelecido. Esse critério é extremamente importante e um guia tanto para a equipe quanto para o *Product Owner* estabelecerem um acordo sobre a finalização do item desenvolvido.

Independentemente da finalização das histórias, o *timebox* deve ser respeitado. Todas as histórias presentes no *Sprint Backlog* devem ter seu status apresentado ao *Product Owner*. As que foram finalizadas, devem ser apresentadas funcionando no sistema, para que o dono do produto possa avaliar seu comportamento, funcionalidades, tempo de resposta, usabilidade e quaisquer outros critérios que sejam relevantes.

No modelo a seguir, temos a representação da dinâmica observada ao longo dessas reuniões.

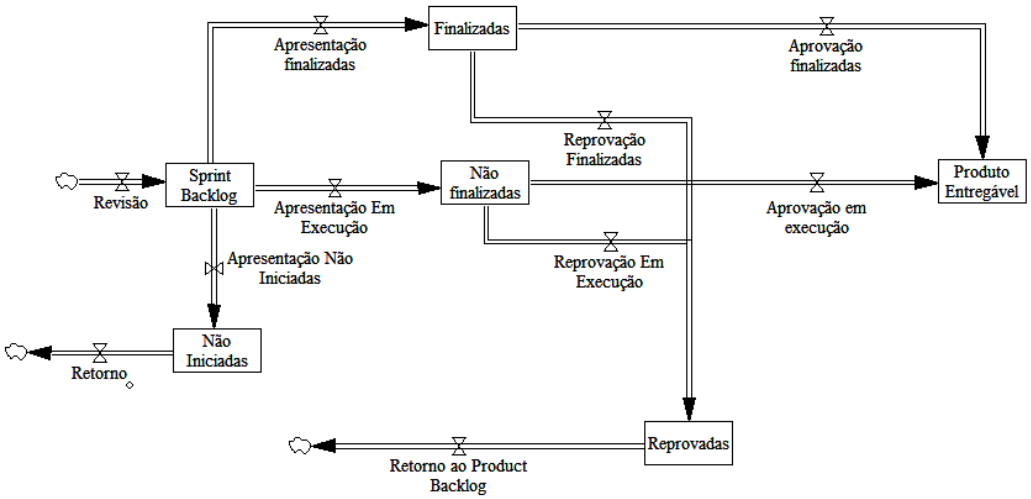

Figura 10. Modelo representando a revisão

Como podemos ver, o *Sprint backlog* é todo levado para avaliação do *Product Owner,* mas em diferentes status. Histórias não iniciadas são aquelas onde não existe nenhum trabalho em execução ou finalizado no momento da revisão e, portanto, são retornadas ao *Product Backlog* diretamente. Elas podem representar uma estimativa de velocidade maior do que a realidade ou uma dificuldade não esperada no desenvolvimento das demais histórias da iteração. As causas dessa sobra devem ser analisadas pela equipe para que sejam determinadas e, assim, evitar que esse problema persista.

As histórias que terminaram em execução são aquelas que algum trabalho foi finalizado ou até mesmo apenas iniciado. As mesmas podem ser apresentadas a critério da equipe. É uma prática recomendada, mesmo que o desenvolvimento do item não tenha sido finalizado, pois permite que o *Product Owner* avalie a funcionalidade e possa evitar que erros possam ser cometidos antes da sua finalização. Em alguns casos, esses itens podem até ser aprovados, já que o trabalho que a equipe julgava restante não era necessário aos olhos do dono do produto. É esperado que essa taxa seja mais baixa que em histórias finalizadas, mas não há impeditivo da existência da mesma.

Finalmente as histórias finalizadas são as que possuem todas as suas tarefas finalizadas, ou seja, não há nenhum trabalho restante a ser feito aos olhos da equipe. É esperado uma taxa elevada de aceitação desse tipo de histórias e pode ser um indicador de qualidade para representar a sintonia entre o trabalho da equipe e do *Product Owner*. Uma baixa taxa de aprovação desse status da história pode representar problemas sérios de comunicação entre a equipe desenvolvedora e o dono do produto.

Independentemente do estado da história, há a possibilidade de a mesma não ser aceita pelo *Product Owner*. Entende-se que o status da mesma é relevante para ela ser ou não reprovada e por isso as taxas são diferentes. Os itens que não irão compor a entrega da Sprint são reservados e é desejável para o processo que uma justificativa seja fornecida, possibilitando a equipe entender os problemas que existiram e evitar repeti-los. As histórias reprovadas poderão voltar a compor o *Product Backlog* a critério do dono do produto. Vale ressaltar que o esforço de realização de uma história que foi reprovada não necessariamente se mantém idêntico. O mesmo poderá ser menor, igual ou até mesmo maior devido a eventos externos ou a melhor explicação dos requisitos.

3.7. Validação

Para um resultado mais robusto e útil se faz necessário validar o modelo em relação ao que ele se propõe apresentar. Infelizmente, não é possível provar que um modelo seja totalmente correto (Robinson, 1997). Apesar disso, realizar a validação do mesmo faz com que aumente a confiança no resultado e facilite a aceitação do mesmo para uso a ponto de ser útil para outros tomadores de decisão que estão avaliando o domínio.

A validação de um modelo consiste em assegurar que esse instrumento reflete adequadamente o comportamento que pretende se representar a partir do mundo real. Para validar modelos de simulação o mais comum é realizar consultas com especialistas que entendam do domínio abordado pelo modelo (Robinson, 1997). Essa etapa visa consolidar a validade conceitual do modelo mostrando que é razoável e coerente para o cenário que está sendo dirigido.

Para uma visão abrangente, foram entrevistadas 3 pessoas cujos papéis ao longo de sua experiência no desenvolvimento de software fossem diferenciados para uma visão mais ampla do que está sendo abordado. Os papéis diferentes buscam uma complementação no conhecimento de cada etapa do processo já que o *Product Owner* poderá ter uma visão diferenciada da dinâmica do *Product Backlog* de um membro da equipe de desenvolvimento. Não é correto, entretanto, afirmar que as visões apresentadas representem o universo completo desses papéis dado o número pequeno de representantes.

Os 3 especialistas solicitados foram:

- Luiz Fernando Cardoso Tomaz, M.Sc., CSM/CSD: Graduado em ciência da computação pela Universidade Federal do Rio de Janeiro e mestre pelo PESC/UFRJ, possui experiência de mais de 10 anos como programador e líder técnico de equipes. Possui mais de 6 anos de experiência com o *framework Scrum* e é professor do Centro Universitário Serra dos Órgãos, tendo participado de mais de 20 bancas e orientado mais de 10 alunos. Foi membro integrante de diversos dos projetos estudados nesse trabalho e questionado para apresentar uma visão representativa dos membros da equipe de desenvolvimento.

- Ester Lima de Campos, M.Sc., CSP/CSM/CSPO: Graduada em Engenharia Eletrônica e Computação pela Universidade Federal do Rio de Janeiro e mestre pelo PESC/UFRJ, é especialista em metodologias ágeis e atua há mais de 10 anos na área de desenvolvimento de software. Ministra treinamentos sobre o *Scrum* e atua em consultoria de adoção da metodologia em empresas. É também professora do MBA da Poli/UFRJ na disciplina de Metodologias Ágeis e Qualidade. Foi *Scrum Master* de quase a totalidade dos projetos estudados e foi abordada para representar a visão do papel de *Scrum Master* no processo.

- José Augusto Rodrigues Neto, D.Sc., CSP/CSM/CSPO: Graduou-se em Sistemas de Armas pela Escola Naval, pós-graduado em Análise de Sistemas pela PUC-Rio, mestre em *Operation Research* e em Computer Science pela *Naval Postgraduate School* e Doutor em Engenharia de Sistemas e Computação pela COPPE/UFRJ. É autor de dezenas de trabalhos científicos publicados no Brasil e no Exterior e possui mais de 20 anos de experiência e atuação em projetos de TI e consultorias. Possui sólida experiência de negócio e agilidade, sendo *Product Owner* responsável pela *ScrumHalf*. Sua entrevista foi realizada para abordar uma análise do PO sobre o processo.

Todos os modelos foram apresentados aos especialistas e debatidos sobre se estavam realmente representando o processo baseado no *framework Scrum*. Inicialmente os modelos apresentavam algumas particularidades que foram alteradas após a validação com os especialistas. Essas alterações serão tratadas nessa seção.

Inicialmente, o modelo do *Product Backlog* contava com dois repositórios: um representava as histórias que foram aceitas pelo *Product Owner* representando o *Product Backlog* conforme descrito no guia do *Scrum* e outro um repositório colaborativo onde seriam coletadas sugestões de toda a equipe e de *stakeholders* e, posteriormente, aprovadas pelo *Product Owner* para compor o escopo do desenvolvimento do produto. Essa particularidade foi inspirada no funcionamento do *ScrumHalf* ("ScrumHalf", 2010). Ao longo da validação, foi levantado que esse comportamento não é relacionado ao *Scrum* propriamente dito e deveria ser tratado como uma extensão possível, mas não deveria compor o modelo inicial do estudo que visa compor um processo genérico baseado nesse *framework*.

Outra alteração que foi realizada após o processo de validação e das informações coletadas foi o entendimento e definição do impedimento e de histórias impedidas. Esse tema é tratado no item 3.1.1 e foi decidido, para esse trabalho em especial, que o conceito de história impedida está relacionado ao fato da mesma não poder ser finalizada e seu desenvolvimento já ter sido iniciado. Inicialmente, esse conceito não estava estabelecido e uma história não iniciada poderia ir diretamente para o estado de impedida e idealmente também retornar para não iniciada após seu obstáculo ter sido removido. Essa alteração foi incorporada ao modelo e o conceito de impedimento fixado para evitar divergências de entendimento.

Além dos comentários descritos acima o modelo foi considerado coerente e representativo de um processo genérico de desenvolvimento baseado no *framework Scrum.* Isso, vale ressaltar, não é definitivo que o modelo esteja realmente adequado e finalizado. A exposição do trabalho ao mundo científico, além da validação inicial realizada, é uma das etapas para consolidação do mesmo como definitivo na representação e simulação desse paradigma.

3.8.Extensões do modelo

É importante observar que o modelo apresentado nesse trabalho não visa abraçar por si só todas as variâncias e processos que se baseiam no *framework Scrum*. Por isso, apesar de entender que o processo básico é atendido, não há problemas em alterar o modelo e incluir ou retirar etapas que representem melhor a realidade do processo utilizado em cada caso.

Extensões e adaptações são permitidas pelo modelo para que o mesmo se adeque a maneira de trabalho e particularidades de cada processo. Nessa seção falaremos de algumas adaptações comuns que podem ser realizadas e de que maneira elas devem ser feitas para um melhor aproveitamento do modelo.

O *Scrum* trata o *Product Backlog* como um artefato único em que se reúnem os requisitos necessários para o desenvolvimento do produto. Ao longo do tempo, conforme o sistema vai se consolidando e as funcionalidades começam a ser usadas podem ser encontrados defeitos e problemas no produto que foi entregue. O retrabalho encontrado, para ser corrigido, deverá seguir o caminho habitual do desenvolvimento de uma *feature*: ser inserido no *product backlog*, priorizado, estimado, incluído na *sprint*, desenvolvido e resolvido. Entretanto, nesse caso, a taxa de adição de itens a esse artefato estará demonstrando não um problema de requisitos, mas sim de qualidade do desenvolvimento.

Para melhor representar essa dinâmica, uma proposta de extensão consiste em dois repositórios que serão alimentados de maneira independente. Essa separação será realizada para poder entender o crescimento de itens de *backlog* que são defeitos (bugs) ou que são *features*. Dessa maneira buscamos entender como o escopo do sistema cresce, além de compreender a qualidade do produto que está sendo gerado. Uma taxa muito alta de defeitos pode representar um problema de qualidade e evidenciar problemas mais sérios no desenvolvimento. Um crescimento acentuado de *features* pode ser considerado normal para produtos de escopo aberto, mas preocupante em outros casos. Essa separação se torna crucial para uma melhor análise do comportamento do *Product Backlog*.

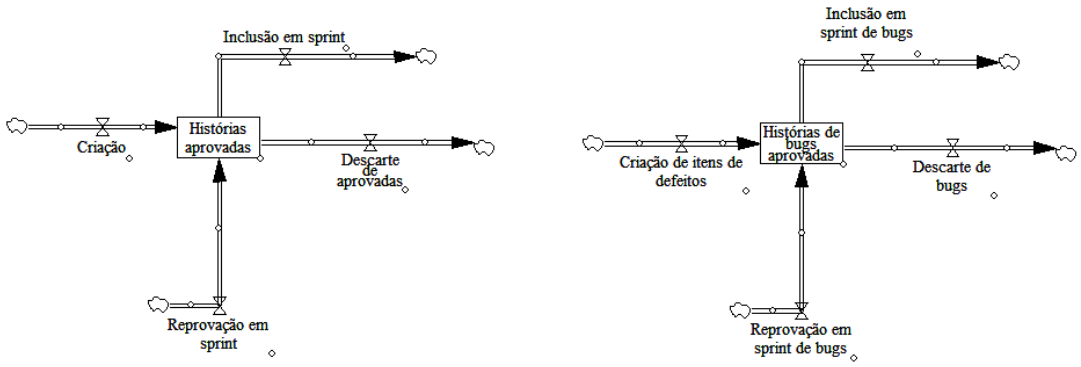

Figura 11. Modelo separado em tipos diferentes de itens

Podemos ver que a extensão apresentada não representa grandes alterações ao tradicional, sendo somente replicado para representar diferentes tipos de histórias. Apesar de estruturalmente idênticos, as taxas e curvas para cada caso podem ser bem diferentes, representando diferentes cenários e fazendo com que essa divisão seja necessária para compreender a situação real do projeto. A separação desses itens gera um grande ganho de entendimento e é apenas um exemplo, limitado pela base de dados que está sendo trabalhada. Uma categorização dos defeitos também seria interessante para tentar encontrar a simular problemas que serão encontrados ao longo do projeto, como sugerido por Kalinowski e Marcelo Costa (Marcelo Nascimento Costa, 2008). Uma equipe que possui pouca experiência com construção de interfaces poderá apresentar uma taxa de defeito maior nesse quesito do que outra. O comportamento aqui descrito poderá ser estendido para diversos outros itens e demais métricas poderão ser obtidas para refinar o modelo.

4.Estudo de caso

4.1. Projetos analisados

Os projetos aqui analisados foram desenvolvidos na mesma empresa por equipes multidisciplinares e auto organizadas, conforme recomenda o guia do *Scrum* (“Scrum Guide”, 2013). Os projetos se diferenciavam em relação a escopo, experiência dos desenvolvedores e também poderiam ser desenvolvimento de produto com cliente externo ou interno, no caso de serviços.

O projeto que criou o sistema de gerenciamento ágil utilizado para a coleta de dados dessa dissertação também foi desenvolvido nessa empresa. Os detalhes do desenvolvimento não podem ser expostos a público com o objetivo de manter a privacidade dos dados. No apêndice é possível encontrar uma breve descrição textual, além de informações referentes a duração dos projetos e outros indicadores.

É sabido que uma breve descrição do projeto não é suficiente para uma contextualização do ambiente em que o mesmo foi desenvolvido. Por isso, na próxima seção, iremos abordar uma maneira de fornecer de maneira mais detalhada o ambiente em que foi e estavam sendo desenvolvidos os projetos acima descritos.

4.2.Contextualização

Apesar de não possuir nenhum nível de maturidade nos modelos MPS-BR ou CMMI se deseja obter uma referência da empresa e dos projetos para caracterizar os dados obtidos da execução dos projetos. Independentemente das práticas adotadas e acrescidas ao *framework* padrão, foram utilizados o *alpha states cards* do SEMAT (“What is SEMAT?”, 2014) para realizar a avaliação da engenharia de software praticada nos projetos.

A metodologia e a avaliação são independentes da metodologia usada e focada nos *Kernels* buscando compreender em que nível cada um deles se encontra. Além da independência do paradigma utilizado, cada área poderá possuir uma avaliação diferente dependendo dos resultados alcançados. A avaliação é feita a partir de um

checklist disponível na própria página(“Progress Poker Flyer”, 2014). A classificação de cada projeto foi realizada em dupla, com o auxílio das cartas e da listagem de requisitos necessários.

Os *alphas* são retirados da especificação do *Object Management Group* ESSENCE (“Essence”, 2014). Os *kernels* são divididos em áreas e se relacionamento entre si conforme a figura 6 abaixo.

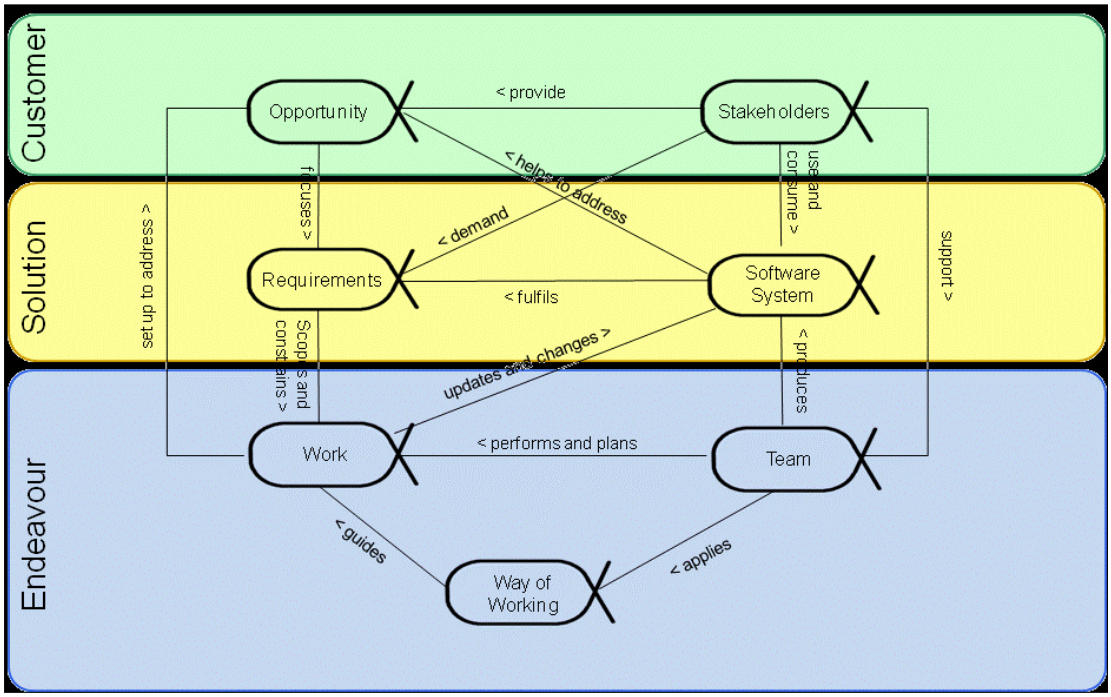

Figura 12. Os *kernels*, as áreas e seus relacionamentos

A definição de cada área e *kernel* está descrita como:

- *Costumer* (Cliente): Nessa área está contido tudo o que é relativo ao uso e à exploração do sistema de software a ser produzido.
    - *Opportunity*: É definido como o conjunto de circunstâncias que fazem o desenvolvimento de um sistema ou a mudança de algum existente apropriado. Pode possuir 1 dos 6 estados abaixo.
        - Identificada: Uma oportunidade de negócio, social ou comercial foi identificada e pode ser suprida com uma solução baseada em software.
        - Solução necessária: A necessidade de uma solução baseada em software foi confirmada.

        - Valor estabelecido: O valor de uma solução bem-sucedida está estabelecido.
        - Viável: Está acordado que uma solução pode ser produzida em um tempo e custo baixo o suficiente para atender a oportunidade.
        - Endereçada: Uma solução foi produzida e está demonstrado que atende a oportunidade.
        - Benefício atingido: A operação ou venda da solução está criando benefícios tangíveis.
    - *Stakeholders*: É definido como a (s) pessoa (s), o (s) grupo (s) ou a (s) organização (ões) que afetam ou são afetadas pelo sistema de software a ser desenvolvido. Pode possuir 1 dos 6 estados abaixo.
        - Reconhecidos: Os *stakeholders* foram identificados.
        - Representados: Os mecanismos de envolvimento dos *stakeholders* foram acordados e o representante dos *stakeholders* foi nomeado.
        - Envolvidos: O representante dos *stakeholders* estão ativamente envolvidos no trabalho e cumprindo suas responsabilidades
        - De acordo: O representante dos *stakeholders* estão de acordo.
        - Satisfeitos com a entrega: As expectativas mínimas do representante dos *stakeholders* foram atingidas
        - Satisfeitos no uso: O sistema atingiu ou superou as expectativas mínimas dos *stakeholders*.
- *Solution* (Solução): Nessa área está contido tudo o que é relativo ao desenvolvimento e à especificação do sistema de software.
    - Requisitos: É definido como o quê o sistema deve atender para satisfazer os *stakeholders* e a oportunidade. Pode possuir 1 dos 6 estados abaixo.
        - Concebidos: A necessidade de um novo sistema foi acordada.
        - Delimitados: O propósito e a extensão do novo sistema são claros.

        - Coerentes: Os requisitos provêm uma descrição consistente das características essenciais de um novo sistema.
        - Aceitáveis: Os requisitos descrevem um sistema que é aceitável pelos *stakeholders.*
        - Endereçados: Requisitos suficientes foram descritos para satisfazer a necessidade de um novo sistema de forma aceitável pelos *stakeholders*.
        - Cumpridos: Os requisitos que foram descritos satisfazem a necessidade do novo sistema.
    - *Software System*: É definido como o sistema feito de *software*, hardware e dados que fornecem o valor para a execução do *software*. Pode possuir 1 dos 6 estados abaixo.
        - Arquitetura selecionada: Uma arquitetura foi selecionada que atende os riscos técnicas e qualquer restrição aplicável à organização.
        - Demonstrável: Uma versão executável do sistema está disponível para demonstrar que a arquitetura escolhida é viável para o caso e testes são suportados.
        - Usável: O sistema está usável e demonstra todas as características de qualidade de um sistema operacional
        - Pronto: O sistema (como um todo) foi aceito para o ambiente de produção
        - Operacional: O sistema está em uso em um ambiente de produção.
        - Aposentado (*Retired*): O sistema não é mais suportado.
- *Endeavour* (Esforço): Nessa área está contido tudo o que é relativo ao time e à maneira que o trabalho é abordado.
    - Time: É definido como o grupo de pessoas que está efetivamente engajado no desenvolvimento, manutenção, entrega ou suporte do sistema de software em questão. Pode possuir 1 dos 5 estados abaixo.
        - Semeado: A missão do time é clara e o conhecimento necessário para executá-la é possuído.

        - Formado: O time está composto com comprometimento suficiente para começar o trabalho.
        - Colaborando: Os membros do time estão trabalhando como uma única unidade.
        - Realizando (*Performing)*: O time está trabalhando eficientemente e efetivamente.
        - Suspenso: O time não é mais responsável para executar o trabalho necessário no sistema
    - *Work*: É definido como a atividade que envolve esforço mental e físico com o objetivo de alcançar um resultado. Pode possuir 1 dos 6 estados abaixo.
        - Solicitado: O trabalho foi solicitado
        - Preparado: As condições para o início do trabalho foram atingidas
        - Iniciado: O trabalho foi iniciado
        - Sob controle: O trabalho está andando bem, os riscos estão sob controle e a produtividade está em um nível suficiente para atingir um resultado satisfatório
        - Concluído: O trabalho para produzir os resultados desejados foi concluído.
        - Encerrado: Todas as tarefas, inclusive de revisão do trabalho e reuniões de encerramento, foram realizadas e o trabalho está oficialmente finalizado.

    - Maneira de trabalhar: É definido como o conjunto de práticas e ferramentas escolhidos para serem usados pelo time como guia e suportar o trabalho necessário. Pode possuir 1 dos 6 estados abaixo.
        - Princípios estabelecidos: Os princípios, restrições e formas da maneira de trabalho foram estabelecidos
        - Fundações estabelecidas: As práticas chaves e ferramentas que formam a maneira de trabalhar foram selecionadas e estão prontas para o uso
        - Em uso: Alguns membros estão usando e adaptando a maneira de trabalhar

- No lugar: Todos os membros estão usando a maneira de trabalhar acordada para concluir seu trabalho
- Trabalhando bem: A maneira de trabalha está funcionando bem para o time
- Aposentado (*Retired*): A maneira de trabalhar não é mais usada pelo time

O objetivo dessa etapa é contextualizar o ambiente em que os dados foram adquiridos para facilitar futuros estudos ou comparações de novas execuções, simulações ou desdobramentos do trabalho apresentado. Um trabalho a ser desenvolvido é a comparação entre os índices, fatores e relacionamentos obtidos entre projetos com diferentes níveis de maturidade. É esperado, por exemplo, que a taxa de conclusão de histórias seja mais alta em times mais robustos e que a taxa de aprovação aumente com a melhoria da comunicação. Essas hipóteses, criadas apoiadas no bom senso, poderão ser comprovadas ou refutadas com mais experimentos em diferentes ambientes.

Para o ambiente em questão, dois projetos foram avaliados utilizando a metodologia e o ambiente do time foi questionado para a empresa como um todo, buscando descobrir o alinhamento da empresa com as práticas adotadas no *scrum*. Para obtenção dos dados, foi utilizado a técnica do *Progress Poker* (“Progress Poker Flyer”, 2014), da maneira tradicional com a adoção do *checklist* completo encontrado no guia de referência.

Dois membros de cada projeto foram selecionados aleatoriamente e questionados como informado no guia sobre a posição de cada estado e, além disso, foram instruídos a informar quais itens do *checklist* não foram atendidos e por isso o processo não pode ser considerado em um nível acima. Dessa forma, além conhecer o estado atual de cada *kernel* ainda é possível entender onde o processo está pecando, tornando a informação mais completa. Esse processo foi repetido até que o consenso entre os membros fosse atingido, tanto em relação ao nível quanto aos itens não atendidos.

Os demais projetos, no entanto, não puderam ser contextualizados dessa forma dado que já estavam com a equipe dissolvida, eram antigos, já estavam terminados ou o desenvolvimento estava interrompido. Para esses casos, não foi possível determinar o estado de cada alpha e por isso não há maiores informações de contexto em relação a

eles. Vale ressaltar também que há uma volatilidade ao longo do projeto do estado e, portanto, mesmo que o time tenha atingido o nível de *Performing*, não é garantido que o mesmo esteve nesse estado ao longo de todo o projeto e que não houve nenhuma interrupção posterior que faz o time retroceder alguns níveis. A contextualização é importante mas vale ressaltar que a mesma é apenas um snapshot de um determinado momento do projeto, podendo variar ao longo do mesmo e por isso, apresentar alguma informação de aparente inconsistência.

Além dos projetos atuais, os participantes em conjunto foram questionados em relação a adoção do *Scrum* na empresa, buscando repassar as experiências nos diversos projetos que passaram e na observação da adoção das práticas e princípios defendidos pelo *framework*. Para essa análise, apenas dois *alphas* foram considerados possíveis de serem avaliados, dado a diversidade do negócio, dos clientes, da natureza e da etapa em que cada projeto se encontrava. Sendo assim, apenas a classificação de *Way of Working* e de Team foi possível de ser trabalhada, obtendo-se um resultado de *In Use* e *Formed* respectivamente

Para os projetos individuais, os resultados são apresentados na forma do gráfico de "aranha", sendo o mesmo utilizado na página oficial da técnica sendo considerado, assim, uma forma efetiva de visualização da informação. Na figura 13 podemos ver o resultado relacionado ao projeto A.

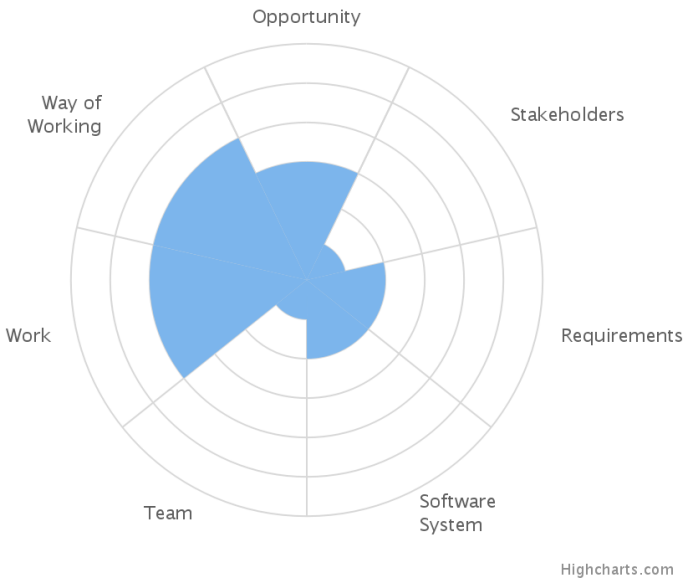

Figura 13 - Avaliação do projeto A através do *Progress Poker*

Os valores obtidos foram valor estabelecido em oportunidade, reconhecido em *stakeholders*, delimitados em requisitos, demonstrável em *software system*, semeado no time, sob controle em trabalho e no lugar para *way of working*. Nesse projeto, a necessidade do negócio não era transparente para a equipe e muitas vezes não se sabia exatamente o problema que o requisito estava tentando resolver.

Para o projeto B, obtivemos o seguinte resultado visualizado na figura 14 abaixo.

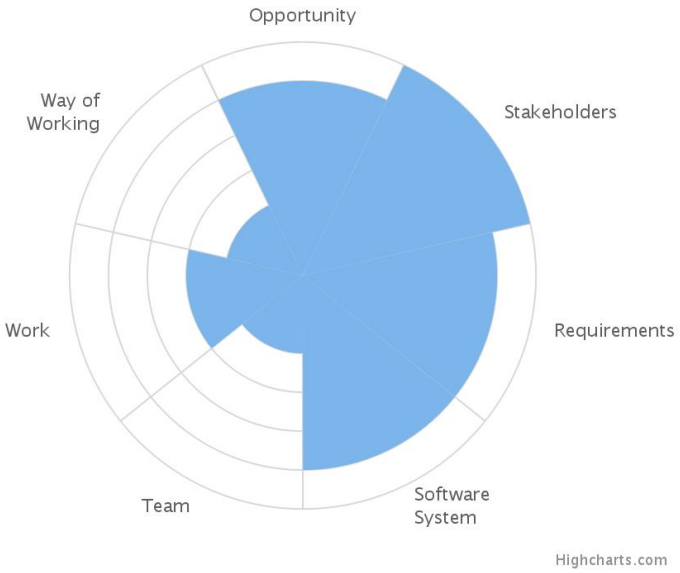

Figura 14 - Avaliação do projeto B através do *Progress Poker*

No projeto B, foi obtido o grau de endereçado em oportunidade, satisfeito no uso em *stakeholders*, endereçado em requisitos, operacional em *software system*, formado em time, iniciado em trabalho e fundamento estabelecido em *way of working*. Como podemos perceber, o projeto estava pecando na região do esforço. Nossa análise é que isso se devia a mudanças recentes dentro do time que ainda estava estabilizando. O trabalho, apesar de já ter sido iniciado e entregue, não atingiu o critério de ter a estimativa reavaliada para refletir a performance do time e, por isso, só atingiu ao estado de Iniciado.

Essa seção busca contextualizar além de uma descrição solta o contexto em que os dados foram coletados. Essa informação é importante para que o estudo possa ser replicado em diversos contextos e percebermos a diferença entre eles, bem como perceber se há estabilidade e coerência entre projetos executados em diferentes ambientes com o mesmo grau obtidos nesses setores. É possível também estudar o

impacto da melhora de cada um desses indicadores com o melhor andamento do projeto executado.

4.3.Coleta de dados

A coleta de dados foi realizada mediante acesso ao banco de dados que contêm os indicadores que estão previstos no modelo. A pesquisa abrange 6 projetos, sendo 2 deles considerados de escopo fechado e 4 de escopo aberto. Todos os projetos foram desenvolvidos pela empresa categorizada, ou seja, sem intervenção de terceiros no código.

Os dados considerados inconsistentes ou incorretos foram retirados do estudo, como por exemplo, *Sprint* de apenas 1 dia de duração ou histórias não iniciadas que foram aprovadas. Nesses casos, os dados relativos àquela etapa do desenvolvimento são desconsiderados para a obtenção dos indicadores. Esses eventos são considerados como erro do sistema ou na execução do mesmo. Além disso, existem dados que não foram possíveis serem obtidos por conta da não existência no mesmo na época, como por exemplo a contagem de tempo alocado no desenvolvimento de cada tarefa. Essa funcionalidade só foi inserida no sistema após o início da coleta de dados e, portanto, a informação não foi recuperada para algumas *sprints* iniciais de determinados projetos. A última categoria de dados que foram analisados e corrigidos são os *outliers* no tempo de desenvolvimento de uma tarefa. Se tornou uma situação comum o desenvolvedor encerrar o desenvolvimento da tarefa, mas não mover a mesma para a etapa de finalizada e, além disso, não realizar a pausa do desenvolvimento quando o mesmo se ausentava da empresa ou estava encarregado de outra demanda. Além dos motivos de falha humana, poderia acontecer também o problema de falha no sistema, impossibilitando a coleta precisa desse indicador.

4.4.Obtenção dos indicadores

Os indicadores propostos pelo modelo podem ser coletados a partir de dados obtidos ao longo da execução de projetos. No caso do estudo, todo o desenvolvimento estava apoiado na ferramenta de gestão do desenvolvimento ágil *ScrumHalf*

(“ScrumHalf”, 2010). A partir dos dados que foram inseridos em um banco de dados relacional foi possível capturar os indicadores necessários para realizar a simulação.

O modelo permite que a série utilizada para a entrada de dados possua qualquer formato e, portanto, não está restrito a apenas uma forma de representação de uma distribuição normal, conforme foi utilizada nesse trabalho. Para fins de estudo, todas as taxas retiradas do projeto foram coletadas buscando a média e o desvio padrão e utilizando uma geração randômica na distribuição normal utilizando esses valores.

A seguir será feito uma descrição dos indicadores coletados de cada projeto:

- Criação: Esse indicador representa a taxa em que os pontos necessários que representam o escopo do produto são adicionados aos *Product Backlog*. Em alguns projetos, o *Product Backlog* terá um valor inicial e pouco será acrescido ao mesmo. Essa taxa representa quantos pontos estão sendo incorporado a esse artefato.

- Descarte: De maneira análoga a criação, o descarte representa a taxa em que os pontos são retirados do escopo do produto. Na agilidade, mudanças são bem-vindas e por isso é esperado que haja uma renovação do escopo do trabalho a ser desenvolvido. A taxa representa a quantidade de pontos que são retiradas do *Product Backlog.*

- Duração: A duração representa o tempo decorrido das iterações dentro do projeto. Idealmente, o tempo é determinado no projeto e fixo até que haja uma decisão posterior de mudança. No entanto, devido a imprevistos, feriados ou eventos e decisões alheias ao projeto, pode existir alguma variação na duração da Sprint. Essa taxa representa a duração da iteração no projeto.

- Velocidade: A cada iteração, uma determinada quantidade de pontos é escolhida para ser desenvolvida. Na fase de planejamento, há uma negociação entre *PO* e equipe que, se baseando na velocidade histórica e em outros fatores da Sprint, decidem quais funcionalidades e quantos pontos serão desenvolvidos. O valor coletado representa a quantidade

de pontos inseridas no *Sprint Backlog* a cada iteração.

- Inicialização: Uma vez com a o *Product Backlog* preenchido, a equipe deverá iniciar o desenvolvimento. Existe uma boa prática relacionada ao não iniciar muitas histórias sem finalização. Essa taxa está relacionada na quantidade de pontos que a equipe inicia o desenvolvimento.

- Finalização: No desenvolvimento ágil, terminar uma história existe um significado especial associado a definição de pronto. A taxa de finalização se refere a quantidade de pontos que a equipe é capaz de transformar em pronto.

- Impedimentos: Ao longo do desenvolvimento, obstáculos e problemas podem impedir que os pontos sejam finalizados. A taxa coletada representa a quantidade de pontos que estão impossibilitados de terem seu desenvolvimento finalizado.

- Resolução: Analogamente, a medida em que os pontos são impedidos, há um trabalho paralelo de resolução desses impedimentos buscando possibilitar a finalização dos mesmos. A taxa coletada representa os pontos que são desbloqueados, ou seja, estão disponíveis para serem finalizados.

- Aprovação Finalizadas: As histórias que são submetidas a aprovação em estado finalizado normalmente são incorporadas ao produto final. Excepcionalmente, algumas histórias podem não estar adequadas ou apresentar problemas e não serem aprovadas nessa etapa mesmo com o desenvolvimento finalizado. Essa taxa representa a porcentagem de aprovação dos pontos que foram finalizados dentro do período de desenvolvimento.

- Aprovação Não Finalizadas: Histórias não finalizadas originalmente não iriam compor o produto final. No entanto, ao apresentar o item, o

Product Owner pode considerar satisfeito com o atual desenvolvimento, postergar as pendências e aceitar os pontos apresentados mesmo assim. Essa taxa representa a porcentagem de pontos aprovados que não tiveram seu desenvolvimento finalizado.

Os indicadores coletados de cada projeto estão localizados no apêndice desse trabalho. Esses valores podem ser utilizados em outros trabalhos, posteriormente, para replicar os resultados encontrados e realizar outros experimentos desejados. É esperado que esses valores possam servir de auxílio e que aumente a base de dados da área em relação a medição de projetos executados, criando uma base mais significativa para futuros estudos.

4.5.Correlações encontradas

Durante a coleta de indicadores necessários para a simulação do modelo também foram feitos estudos de algumas propriedades que foram observadas durante a execução dos projetos. Algumas das observações são intuitivas, mas são reforçadas com a obtenção de dados que corroboram com o cenário imaginado. Observamos, portanto, alguns fenômenos que não foram utilizados para a simulação do modelo, mas que são informativos e por isso considerados relevantes para compor este trabalho.

O tamanho de um item de *backlog* é um assunto amplamente discutido na comunidade ágil. Uma história pequena demais pode não gerar valor quando entregue sozinha, causando um problema de dependência que deve ser evitado. Por outro lado, itens de grande escala dificultam a entrega rápida e faz com que princípios do ágil sejam feridos. Encontrar o tamanho ideal, um equilíbrio entre independente o suficiente e com valor, é um trabalho árduo e de difícil realização mesmo com diversas técnicas existentes atualmente para isso.

Pontuar histórias com valores altos está associado a riscos. O primeiro risco está na discrepância de valores da escala amplamente divulgada e utilizada nos projetos estudados. O *pseudo-fibonacci* e a natureza *fuzzy* da escolha dos números por si só já gera um erro de estimativa maior associado aos grandes números do que aos pequenos (“Incerteza nos Pontos de História”, 2011). Uma história de pontuação mais alta oferece uma variância maior e mais perigosa para sua finalização no tempo planejado.

Nesse momento não iremos discutir sobre como conseguir o tamanho ideal de uma história, mas sim fazer uma análise de todos os itens desenvolvidos e seu desfecho ao final de uma iteração de desenvolvimento. Foi coletado o índice de finalização e de aprovação de cada item agrupado por seu projeto, para que a unidade de medida fizesse sentido dentro do contexto. Como resultado, esperamos uma correlação entre o número de pontos atribuídos a uma história e sua aprovação. Por se tratar de um valor subjetivo, não é possível obter o valor final que aquele item realmente entregou, apenas se foi ou não aprovado em revisão.

Como etapa de filtragem dos dados, foram retiradas as histórias de valor de estimativa 0 pois elas muitas vezes eram utilizadas como marcadores ou como controle de atividades foram do projeto. Como esse valor está associado a essa prática na empresa, as histórias de valor 0 não representam exatamente um requisito de esforço mínimo a ser realizado e, por isso, foram desconsiderados para essa etapa.

	Pontuação							
Proj.	0.5	1	2	3	5	8	13	**CdC**
SH	2.63%	14.73%	12.78%	31.25%	36.53%	45.16%	47.37%	0.88
A	0%	5.41%	5.49%	10.41%	12.57%	31.52%	31.25%	0.94
B	14.55%	12.35%	14.52%	22.92%	37.04%	50%	57.14%	0.96
C	12.2%	36.17%	9.52%	40%	50%	100%	*	0.92
D	12.5%	30.43%	28.57%	30.77%	*	*	*	0.68
E	9.09%	15.79%	13.95%	22.73%	27.78%	36.36%	*	0.97

Tabela 1 – Percentual de reprovação de histórias por pontos estimados

Podemos observar a partir dos dados obtidos uma correlação alta em todos os projetos da relação entre pontos de história e índice de reprovação. Esse fenômeno não pode ser interpretado como uma verdade em todos os projetos que se baseiam em *Scrum*, visto que o universo dos dados é pequeno e é possível que exista um viés da organização, dados que são todos da mesma pessoa jurídica. No entanto, é interessante encontrar um resultado que corrobora com o esperado pela teoria que, por oferecerem mais riscos e pela possibilidade de enfrentar um maior número de problemas e terminar inacabado, histórias de maior pontuação terão um índice menor de aprovação e de finalização.

É importante também observar que uma alta correlação não é evidência de causa e efeito. Portanto essa análise não nos faz concluir que o aumento do índice de reprovação de histórias se deve ao aumento de sua pontuação. Conforme discutido anteriormente, um maior risco associado a um maior esforço necessário para finalização do item pode levar a uma diminuição da chance da aprovação do item, mas, os dados coletados não são uma prova dessa hipótese.

4.6.Produtividade das equipes

Em termos de estimativa de esforço, a agilidade pratica maneiras diferentes de realizar as estimativas. Desde diferentes técnicas até variantes de uma mesma, a abordagem tende a ser um balanço entre acurácia e esforço desprendido para realizar a tarefa. No livro mais famoso publicado, variações e maneiras de realizar essa etapa são descritas, elicitando benefícios e dificuldades de cada uma delas, além de explicar a teoria por trás de cada uma delas. Nos projetos estudados, a estimativa foi realizada a partir de pontos de histórias utilizando a técnica do *Planning Poker* (James W Grenning, 2002).

Ponto de história é uma unidade de medida utilizada para expressar o tamanho de uma história, item de *backlog*, *feature* ou qualquer outro tipo de trabalho a ser realizado. Um número é associado ao trabalho a ser realizado e não possui muita informação sozinho. O importante é que, relativamente com os demais itens a serem trabalhos, os pontos associados a essa *feature* podem ser comparados a outro, criando assim uma visão geral do trabalho a ser realizado e, a partir da velocidade de entrega desses pontos, quanto tempo o time completará o trabalho que foi designado ao mesmo.

A técnica de utilizar pontos de histórias, por não possui um padrão, dificulta a comparação de produtividades entre diferentes equipes. A comparação direta de pontos para fins de análise estatística não deve ser feita porque os critérios utilizados por cada equipe para definir sua unidade de esforço variam e, portanto, uma equipe pode considerar uma história com o esforço maior que a outra devido a unicidade dos seus critérios.

Como prática comum para comparação entre capacidades e produção de equipes e também para fins de estimativa de tamanho de software foi criado a Análise de Pontos de Função (APF) (Maxwell & Forselius, 2000). Essa unidade visa fornecer uma medida

objetiva e comparável para o controle, a gerência e o planejamento do desenvolvimento de software. Através de diversos fatores. Idealmente, para a comparação entre produtividades de equipe, uma medida normalizada como os pontos de função é preferível e, para o contexto ágil, a utilização de planejamento e estimativas flexíveis são desejáveis, para evitar a paralisia pela análise(James W Grenning, 2002).

Na filosofia ágil, software funcionando é mais importante do que documentação abrangente. Partindo dessa ideia, é razoável supor que a maior parte do esforço imaginado para um item de *backlog* esteja ligado diretamente ao desenvolvimento do software, maximizando assim o principal indicador de desempenho. É esperado, portanto, que quanto maior a pontuação da história mais código seja produzido para satisfazer o requisito. Em metodologias tradicionais, o esforço poderia ser redirecionado para documentação ou artefatos não entregáveis de software, o que tende a não ser o caso na abordagem ágil.

Uma tabela de conversão direta entre pontos de função e linhas de códigos para diversas linguagens pode ser usada para uma análise retroativa (QSM, 2009). A partir dessa tabela podemos dizer qual a produção daquela *sprint* associando a quantidade de linha de códigos. Recomenda-se, no entanto, que a análise de pontos de função seja realizada a partir da descrição da funcionalidade e evitar essa conversão.

No contexto do trabalho, a análise posterior dos pontos de função se mostraria pouco confiável devido a diversas tecnologias e linguagens utilizadas, ao escopo das histórias desenvolvidas e a reutilização e alteração de componentes comuns. Por esse motivo, optamos em não utilizar a informação da base de código-fonte para normalização dos pontos e comparação entre projetos. Outra abordagem foi utilizada que se considerou mais adequada para o caso.

A abordagem utilizada para possibilitar a comparação entre a produtividade das equipes é a transformação de pontos de histórias para homem-hora. Essa unidade é amplamente discutida na área de engenharia de software e é conhecido suas falhas e problemas, como a diferença de capacidade no trabalho criativo e eficiência da hora de diferentes profissionais. Entretanto, na falta de melhor unidade de medida, foi realizado o levantamento de quantas horas de desenvolvimento foram gastas para diferentes histórias na forma de gráfico de caixa.

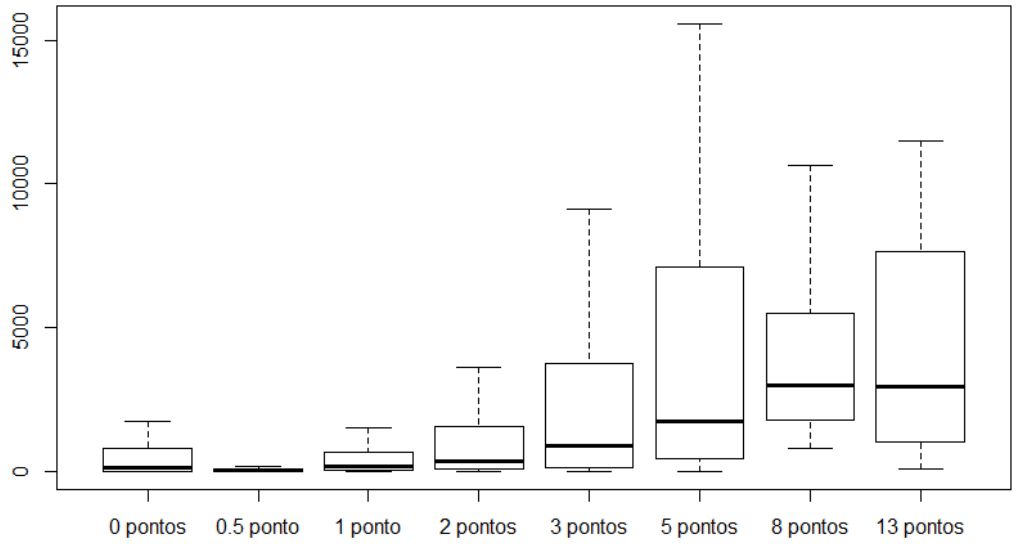

Figura 15. Análise da conversão de pontos para minutos

De posse desse gráfico, podemos normalizar os pontos estimados pela equipe em uma unidade comum e padronizada que é o tempo de desenvolvimento. A partir daí poderá ser utilizado esse valor para a comparação de capacidade de cada equipe mesmo com a diferença de critérios para a estimativa baseada em pontos.

5.Análise de sensibilidade

5.1.Considerações iniciais

Quando há a simulação de um sistema dinâmico uma observação importante é a análise de sensibilidade em relação aos fatores.

Para essa etapa, primeiramente foi considerado o resultado do modelo baseado nos indicadores retirados da base de dados. Dado esse *baseline*, alterações nos indicadores foram feitas para verificar seu impacto comparando com o inicial. Os indicadores que serão utilizados como base de comparação são o *Product Backlog*, representando a quantidade de trabalho que ainda necessita ser desenvolvido e Produto Entregável, representando o trabalho que está aprovado e pronto para ser entregue para o cliente. Ambos valores estão descritos em pontos de história.

O modelo utilizado para simular para os experimentos está representado na figura 16 abaixo. Infelizmente, pelo problema de serem de diferente granularidade, o modelo do desenvolvimento de 1 história não pode ser incorporado na simulação final do trabalho.

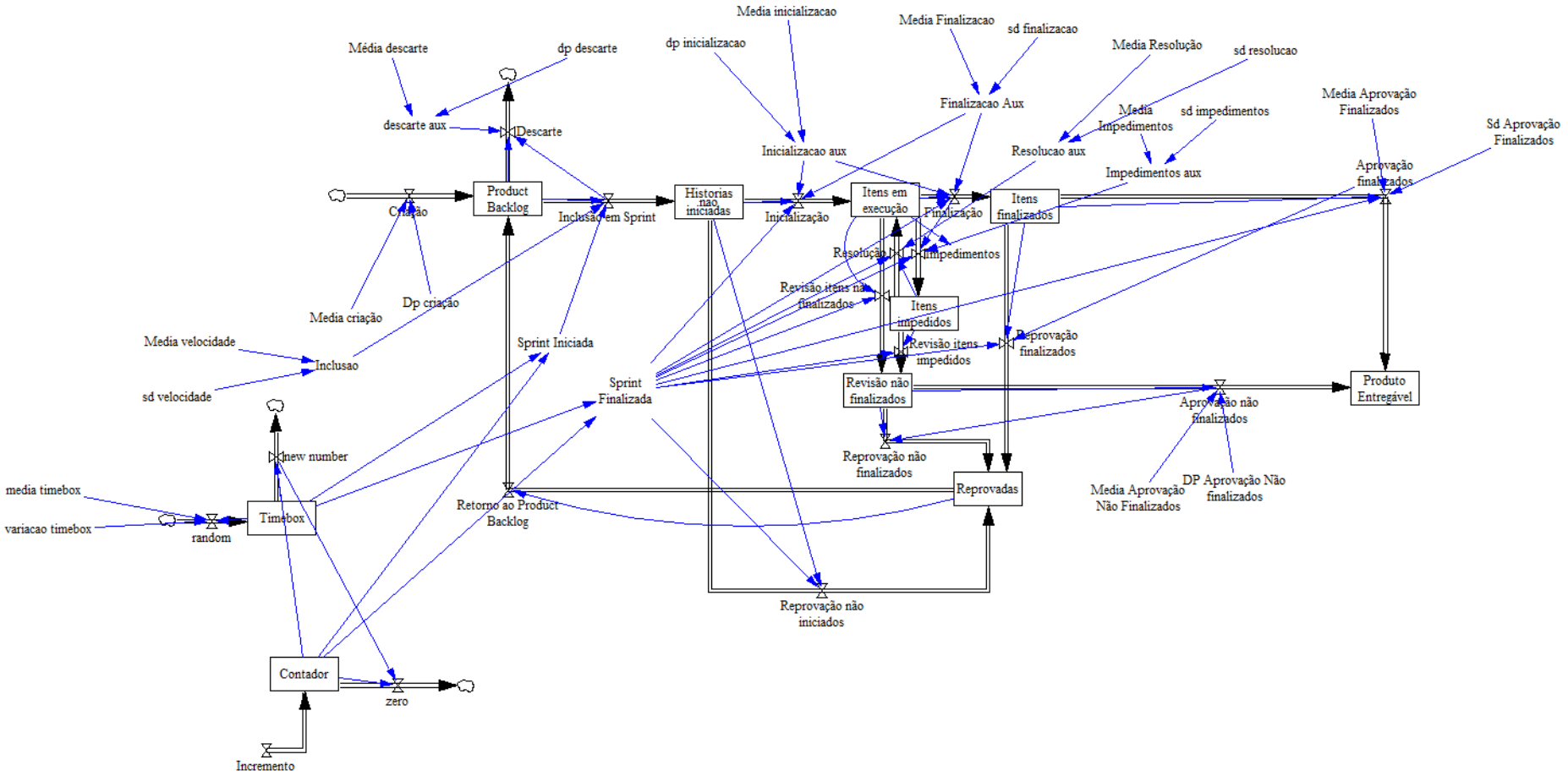

Figura 16. Modelo utilizado para a simulação

Nessa seção apresentaremos os casos e análises relacionadas aos dados do projeto *ScrumHalf*. Todos os casos aqui apresentados também foram realizados para os demais projetos e seus resultados se encontram no apêndice desse trabalho. Discrepâncias encontradas serão comentadas ao longo da seção, mas a utilização do apêndice se deu pela melhor organização e leitura desse trabalho.

5.2.Planejamento

Nossa primeira etapa nessa parte do trabalho foi alterar os indicadores relativos à etapa do planejamento. As duas variáveis que estão ligadas exclusivamente a essa etapa são a criação de histórias e o descarte de histórias. Nesse experimento, iremos aumentar o descarte buscando simular uma situação de *Change for Free* (“Agile Contracts: Money for Nothing and Your Change for Free”, 2008) onde é possível trocar funcionalidades de esforço equivalente sem custo. Essa possibilidade, no entanto, normalmente está associada à participação contínua do cliente durante todo o projeto.

O que foi observado em relação ao *Product Backlog* é que o mesmo não interfere na capacidade de entrega do time a não ser que o mesmo seja menor que a velocidade planejada no início da *Sprint*, o que ocasionaria numa redução forçada de pontos a serem desenvolvidos. Se existir ponto suficiente para que a iteração seja iniciada com a velocidade planejada, não haverá interferência nos pontos a serem desenvolvidos e, consequentemente, os pontos entregues também não sofrerão efeitos. Portanto, a partir dessa observação, podemos concluir que a quantidade de pontos no *Product Backlog* é apenas um fator limitante a capacidade de entrega do time.

5.2.1.Simulação de *Change for Free*

Nesse experimento, buscamos aumentar a criação de histórias, mas aumentando ao mesmo tempo o descarte. É um cenário que busca representar a alteração de escopo sem adicionar trabalho, trocando itens de mesmo esforço buscando aumentar o valor do produto final. Nesse exercício buscamos manter a diferença entre a esperança dos indicadores igual ao caso da *baseline* e para isso usamos a propriedade que abrange variáveis aleatórias E(X-Y) = E(X) - E(Y).

O resultado obtido com esse cenário se encontra descrito na tabela 2 em termos de pontos no *Product Backlog* e em termos de produto entregue.

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
Caso base	1942.87	79.05	595.01	229.54
Change for Free	1984.4	83.79	561.43	216.61

Tabela 2 – Previsão do modelo simulando um cenário de *Change for Free*

Como é possível observar, o resultado final é muito semelhante ao baseline, justificando, portanto, que o contrato do time *Change for Free* não traz atraso para o desenvolvimento do produto e não altera o estado final do *Product Backlog*. Esse resultado, no entanto, não entra no contexto de que o produto está dando um maior retorno de investimento ao cliente e também não é possível obter se o valor entregue foi maior. Essa análise é feita apenas em cima de produto entregue e não do valor que é agregado ao cliente com essa entrega.

5.3.Desenvolvimento da Sprint

Nessa etapa iremos focar o experimento na parte relativa ao desenvolvimento do produto propriamente dito. Iremos levar em consideração as variáveis que estão relacionadas a essa parte do modelo para experimentar a sensibilidade e verificar o aumento da produtividade que pode ser obtido através da otimização ou redução de cada indicador.

5.3.1.Resolução de impedimentos

Os impedimentos dentro do conceito do modelo são eventos que impedem a finalização de uma história. O desenvolvimento não está necessariamente parado, mas

dado que existe um obstáculo, a finalização desse item não é possível até que o mesmo tenha sido resolvido.

Nesse caso a média da resolução de impedimentos foi reduzida a zero, ou seja, uma vez impedida, a história ficaria impedida de ser finalizada nesse *Sprint.* A realização desse teste busca representar um cenário em que o *Scrum Master* esteja ausente, a equipe não esteja autônoma o suficiente para resolver seus obstáculos. Além disso, é importante entender o impacto que essa postura de não resolver os problemas trará para o final do projeto.

A taxa de criação de impedimento não foi alterada. Esses eventos continuam acontecendo com a mesma frequência do caso obtido a partir dos dados da execução do processo. É importante observar que o impacto dessa taxa está diretamente ligado a existência de itens impedidos e, portanto, também está relacionado a taxa de impedimento de histórias.

O resultado obtido a partir de 50 execuções do modelo pode ser observado na tabela abaixo.

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
Caso base	1942.87	79.05	595.01	229.54
Resolução zerada	1778.18	63.74	742.74	163.05

Tabela 3 - Previsão do modelo sem resolução de obstáculos

Podemos perceber, nesse caso, uma diminuição de cerca de 10% na capacidade de entrega desse time nesse projeto. Manter histórias impedidas gera um atraso ao projeto, dado que a mesma não poderá ser finalizada e será apresentada na revisão como item não finalizado que possui taxa de aprovação muito inferior aos itens que tiveram seu desenvolvimento concluído.

5.3.2.Impedimento de histórias

Pensando em outra variável, é preciso verificar a sensibilidade do modelo ao surgimento de impedimentos. Ao longo do projeto, eventos que impedem a finalização de histórias vão surgindo e a taxa de ocorrência do mesmo está representada no modelo. Diversos motivos podem gerar um aumento ou diminuição desse fator que pode representar erros aleatórios ou problemas sistemáticos.

Nesse cenário, buscou-se representar um cenário onde não houvessem esses eventos. O processo correria de maneira fluida sem nenhuma história estar impossibilitada de ser finalizada durante a duração da *Sprint*. A equipe só não finalizaria a história por falta de produtividade e não por estar lidando com um problema que a impeça.

A representação é importante e a demonstração dessa sensibilidade ajuda o time a entender o quanto esses eventos estão impactando no desenvolvimento.

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
Caso base	1942.87	79.05	595.01	229.54
Impedimentos zerados	1961	82.29	624.23	189.47

Tabela 4 - Previsão do modelo sem ocorrência de impedimentos

Como podemos perceber pelo resultado obtido, a ausência de impedimentos impactou muito pouco no resultado final. Analisando as outras variáveis, podemos perceber que a taxa de resolução de impedimentos é bem alta. Mesmo existindo os impedimentos, rapidamente a história retornava ao repositório de em execução e poucos pontos se encontravam no estado de impedido ao final do *timebox.*

5.4.Revisão

Na revisão existem duas variáveis que não estão relacionada a nenhuma outra etapa no modelo. A aprovação de histórias finalizadas e aprovação de histórias em execução são variáveis a serem alteradas e analisadas nesse momento. A proporção entre histórias finalizadas, não finalizadas e não iniciadas é dada pela produção da equipe, variáveis que se encontram estudadas na parte do desenvolvimento.

As histórias não iniciadas são consideradas automaticamente reprovadas e, por isso, seu percentual de aprovação é fixo em 0%. Conforme discutido nos capítulos anteriores por uma questão de semântica não faz sentido verificar a sensibilidade do modelo a esse fator dado que é incoerente que uma história que não foi sequer iniciada seja aprovada e possa ser considerada entregável.

5.4.1. Aprovação de histórias finalizadas

A primeira etapa desse estudo visa analisar o impacto do percentual de aprovação de histórias finalizadas. Esse indicador diz respeito a porcentagem de pontos finalizados que são aprovados, considerando as execuções de *Sprint* coletadas do banco de dados. Para essa análise, o valor foi iniciado em 0% e acrescido em 5% a cada execução do modelo até o valor máximo de 100% sem variância, representando assim uma variedade de 20 casos que variam da reprovação total desse tipo de história até a aprovação total.

O resultado dessa variação pode ser observado no gráfico abaixo.

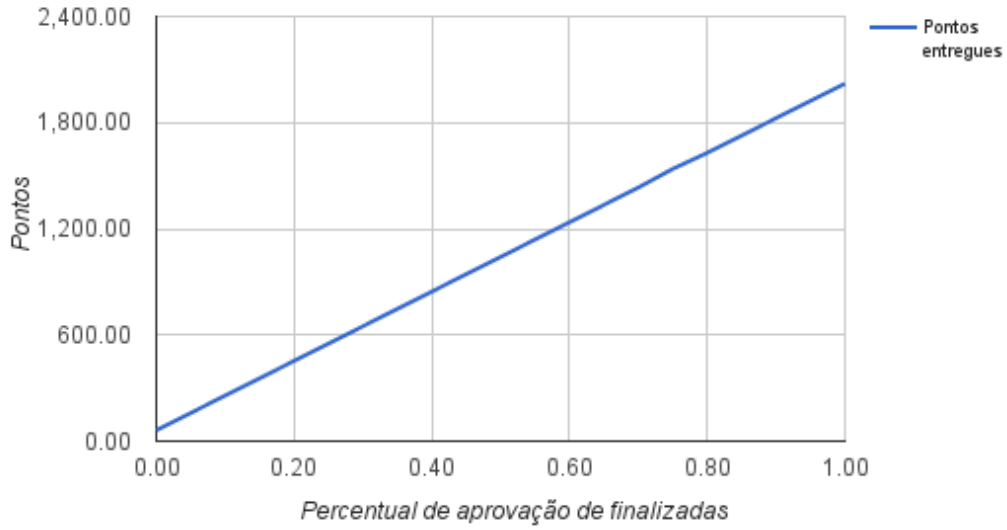

Figura 17. Análise de sensibilidade da aprovação de pontos finalizados para o projeto *ScrumHalf*

Nesse momento, duas observações se fazem importante. A primeira é que mesmo com a reprovação total há uma entrega de pontos. Com o percentual de aprovação sendo 0 a equipe não entregou 0 pontos ao final. Isso se deve ao fato de histórias não finalizadas também serem aprovadas. Entretanto em todos os projetos observados a aprovação de itens neste estado é muito menor do que os finalizados. Comparando a variação realizada com os dados reais obtidos nos projetos podemos perceber pouco espaço de melhoria nesse quesito já que a aprovação nesse caso já estava próxima ao máximo.

A sensibilidade da entrega nesse caso é notória e é compreensível dada que os itens finalizados representam a grande maioria dos pontos entregues em uma *Sprint*. O rigor, a clareza e o entendimento da definição de pronto podem ser essenciais para manter a taxa elevada. Divergências e sutilezas podem fazer com que a equipe considere o trabalho finalizado e apresentar na revisão como pronto e a mesma ser prontamente reprovada pelo *Product Owner*. O time como um todo deve buscar manter essa taxa alta, mantendo a comunicação eficiente e a transparência.

5.4.2. Aprovação de histórias não finalizadas

Como segundo ponto de análise, buscamos entender o impacto da aprovação de histórias não finalizadas no andamento do projeto. Conforme vimos anteriormente, todas as histórias que não tiveram o desenvolvimento dado como encerrado pela equipe e as histórias que continuaram com impedimentos ao final do *timebox* são consideradas para essa etapa.

Conforme vimos nos dados, já era esperado que essa taxa fosse baixa já que o time não entendeu que seu desenvolvimento estava encerrado e atendia todos os critérios de prontos determinados. No entanto, dada a negociação entre equipe e *Product Owner* é possível que esse item seja aprovado. Para determinar a influência final, o mesmo método que foi aplicado no item anterior foi aplicado para esse indicador. Iniciando em 0%, o caso de reprovação total desses itens e aumentando gradativamente em 5% até chegar aos 100% representando o aceite total de todas as histórias que se enquadrarem nesse caso quando a *Sprint* tiver seu desenvolvimento finalizado.

O gráfico resultante possui uma semelhança visual muito parecida com o item anterior sendo, no entanto, bem diferente em sua escala.

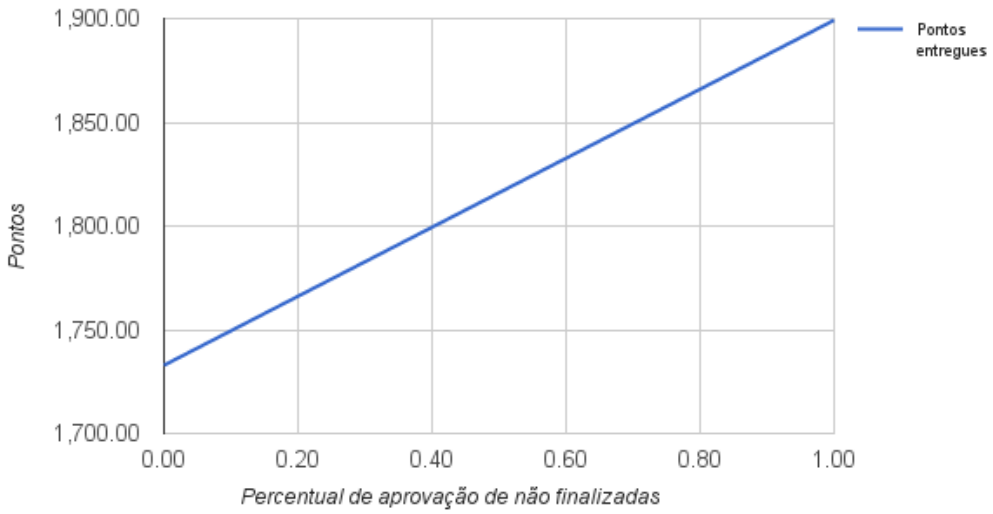

Figura 18. Análise de sensibilidade da aprovação de pontos não finalizados para o projeto *ScrumHalf*

Como era de ser esperar, conforme a taxa de aprovação de itens não finalizados aumenta, os pontos entregues no final de um determinado período também aumentam. No entanto, uma reprovação total de itens nesse estado representa uma entrega de 1733 pontos, muito diferente dos 62 com o indicador anterior. A variação nesse caso se mostrou tênue fazendo com que o modelo não seja muito sensível a alterações nesse indicador.

Tendo em vista que a maior quantidade de pontos estava finalizada ao final da *Sprint* nos casos que foram estudados, alterações na aprovação de itens não finalizados deveriam alterar pouco o resultado final. Além disso, altos valores desse indicador foram realizados para o fim de estudo, mas na prática o percentual de aprovação nesse caso normalmente não é um valor relevante. Problemas podem ser encontrados a partir de valores discrepantes como, por exemplo, a equipe considera que ainda havia trabalho para que o incremento atendesse todos os requisitos de uma história pronta quando, aos olhos do *Product Owner*, os critérios já tinham sido satisfeitos.

5.5.Sprint

No modelo existem variáveis que não afetam uma única área da *Sprint* como é o caso do *timebox*, por exemplo. Esses indicadores interligam e afetam mais de uma área ao mesmo tempo e por isso são tratados em uma seção separada.

5.5.1. Velocidade

No modelo, velocidade é o termo utilizado para determinar o número de pontos que serão incluídos no *Sprint backlog* para que a equipe desenvolva. Ao longo do projeto, é esperado que o time conheça sua capacidade de desenvolvimento e consiga estabelecer um determinado número de pontos que consegue desenvolver e entregar com qualidade e de forma sustentável.

Fatores como o número de desenvolvedores, a experiência dos mesmos e o número de dias úteis podem influenciar diretamente na velocidade de uma *Sprint* do projeto. A cada iteração, a média histórica é revista e ajustada para representar a nova capacidade da equipe, que está em constante evolução e melhoria.

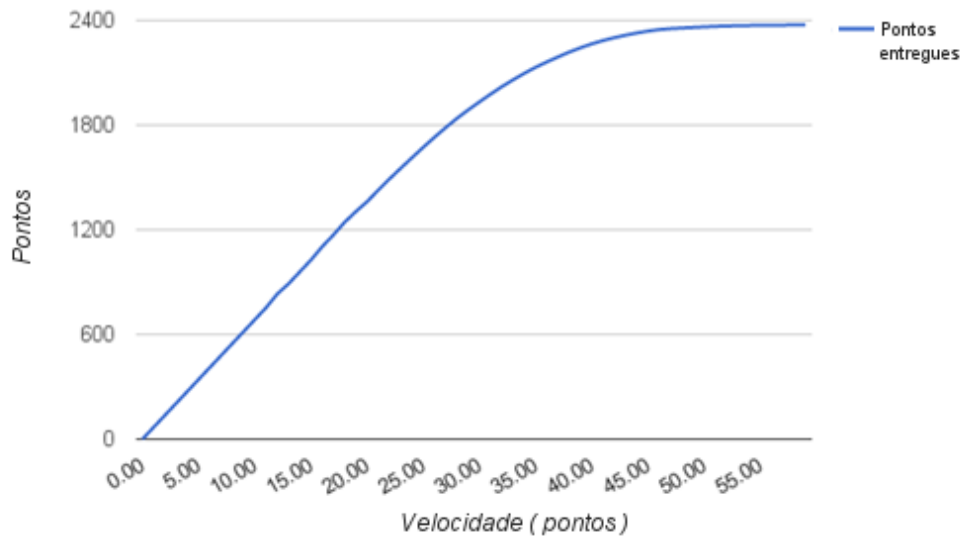

Figura 19. Análise de sensibilidade da velocidade para o projeto *ScrumHalf*

Como podemos perceber, há um grande aumento de produção inicialmente, mas que vai sendo atenuado a partir de determinados valores da velocidade. A equipe possui uma capacidade de inicialização e de finalização de pontos, conforme vimos no modelo. Ao determinar um valor muito baixo para a velocidade, a equipe fica ociosa e potencial

de desenvolvimento é desperdiçado. Até esse ponto, todo o potencial é aproveitado e transformado em itens finalizados e por isso a curva inicialmente tem uma forma linear que representa esse aumento.

A partir de um determinado valor, a finalização de histórias começa a não ser suficiente para a quantidade de pontos que foram inseridos. Nesse momento, a performance do desenvolvimento começa a ser atenuada porque muitos os itens começam a acumular e não são completamente desenvolvidos dentro do *timebox* combinado. O ganho a partir desse momento começa a ser marginal, sendo amparado apenas pela aprovação de itens não finalizadas, discutido no item 5.4.2. Nesse momento a equipe é capaz de iniciar, mas não está mais capaz de finalizar todo o trabalho. Por fim, após um determinado valor, a capacidade de desenvolvimento da equipe está totalmente saturada. Todos os itens adicionados a partir daí se acumulam como itens não iniciados e não aumentam a entrega.

5.5.2.Timebox

O tempo determinado para a duração da *Sprint* representa o número de dias que a iteração vai durar. Normalmente é associado a volatilidade e aos riscos do ambiente em que se está trabalhando.

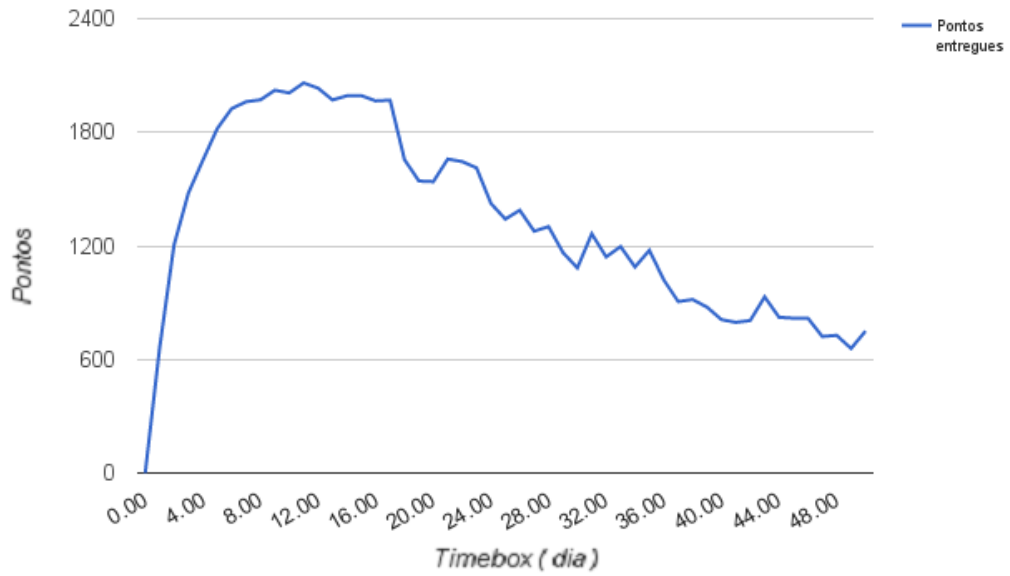

Figura 20. Análise de sensibilidade do timebox sem alteração da velocidade para o projeto *ScrumHalf*

Como podemos ver, apenas aumentar a duração da iteração gera desperdício. Apenas aumentar o tempo do desenvolvimento sem alterar a velocidade de planejamento faz com que a equipe fique ociosa. Dessa forma, conforme a duração aumenta e a produção se mantém igual, o resultado final do projeto sofre grandes perdas já que, apesar de mais duradora, a *Sprint* continua entregando a mesma quantidade de pontos.

Para fazer jus ao aumento do tempo para desenvolver os novos itens, uma nova análise foi realizada. Nessa abordagem, a velocidade no modelo foi maximizada fazendo com que a equipe ficasse saturada e limitada apenas a sua capacidade de inicialização e finalização. Alterações na velocidade foram tratadas na seção anterior e seu impacto no resultado final do projeto é conhecido.

Com essa alteração, obtemos um resultado diferente do anterior sem o desperdício ocorrido após um determinado valor de duração.

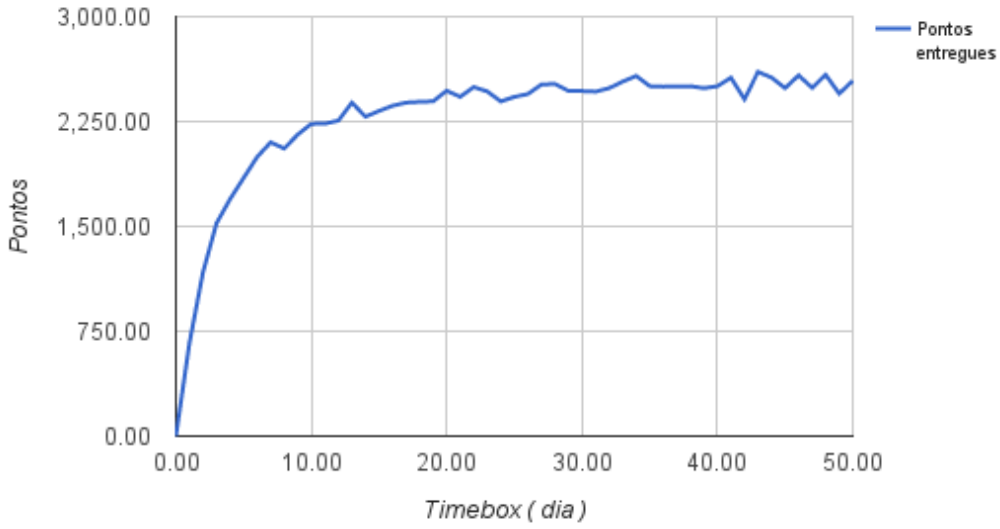

Figura 21. Análise de sensibilidade do timebox com alteração da velocidade para o projeto *ScrumHalf*

Podemos perceber que inicialmente há um aumento muito acentuado na entrega de pontos. Em valores muito pequenos, o *overhead* gerado pelo planejamento, revisão e a necessidade de tempo para iniciar e finalizar a história faz com que a entrega do time seja drasticamente reduzida.

A partir de 10 dias a entrega final começa a não sofrer diferenças significativas. A equipe consegue trabalhar em seu potencial completo de inicialização e finalização

e seu desempenho estabiliza. A diferença nesse momento é que cada iteração passa a entregar mais pontos, mas, como possui uma maior duração, menos iterações ocorrem dentro do mesmo período de tempo.

Podemos perceber uma certa instabilidade após a estabilização. Isso pode ser explicado pela data de término da Sprint vigente no período de tempo simulado. A capacidade do processo depois de um determinado valor do *timebox* está estabilizada e a diferença de pontos é resultado da Sprint em andamento ter terminado ou não. Por exemplo, numa medição de 220 dias, uma *timebox* de 25 dias terá uma Sprint em andamento ao final do período enquanto uma de 22 terá terminado. Isso causa a diferença observada no gráfico.

Com esse resultado podemos entender uma propriedade fundamental do desenvolvimento baseado em *Scrum*: a escolha da duração da *Sprint* está relacionada a fatores como risco, volatilidade e necessidade de responder a mudanças mais rapidamente e não altera a quantidade de pontos na entrega final, respeitando a capacidade produtiva da equipe.

5.6. Capacidade de previsão do modelo

Uma análise final em relação ao modelo dentro dos dados consiste em entender a capacidade e a viabilidade de previsão da capacidade do processo a partir dos dados coletados. O modelo já se mostrou capaz de apresentar resultados coerentes partindo dos indicadores que foram retirados da base de dados de execução dos projetos. Essa capacidade é de extrema importância para mostrar que o modelo é válido e invalidaria imediatamente o trabalho se isso não fosse provado.

Outra pergunta a ser respondida é quando e se o modelo conseguiria ser capaz de prever o andamento do projeto. O modelo deverá ser alimentado de indicadores que poderão ser retirados de uma base de dados do projeto já executado ou imaginados para fins de planejamento. Os experimentos a seguir se referem a coleta dos indicadores até um determinado tempo, percentual do tempo total de desenvolvimento do projeto.

Nesse trabalho, os indicadores foram calculados não em cima do projeto inteiro, mas sim em relação ao tempo imaginado para o experimento. Os resultados a partir do tempo determinado serão calculados a partir do modelo e comparados ao real para

verificar se há a capacidade de predição do resultado do projeto a partir do modelo e a partir de quando o mesmo é possível de ser obtido.

Como informado pela literatura, a estabilidade da velocidade pode ser obtida rapidamente com poucas *sprints*, tipicamente estabilizando entre 3 e 6 iterações (“Agile Scrum Velocity Calculation & FAQs”, [s.d.]), mantendo-se a integridade da equipe e sua independência. Sendo assim, é esperado que a produtividade e entrega da equipe rapidamente seja estabilizada e que rapidamente o modelo possa ser capaz de prever o andamento do projeto.

Por se tratar de um experimento sem outras referências, é difícil realizar uma escolha de tempo que representa o projeto e a maneira que os intervalos serão escolhidos. Inicialmente, dado a informação obtida em (“Agile Scrum Velocity Calculation & FAQs”, [s.d.]), serão simulados os dados de 3 e 6 iterações iniciais. Após realizado esses 2 casos baseados na literatura também serão estudados, apenas de forma ilustrativa, os valores de 25%, 50% e 75% do tempo de projeto decorrido para as demais simulações.

De maneira tradicional, a estimativa da duração de um projeto pode ser calculada dividindo a velocidade do time pela quantidade de pontos existentes no *product backlog*. Dessa forma, teremos o número de iterações necessárias para realizar o trabalho especificado. De maneira análoga, podemos calcular quantos pontos serão entregues em um determinado tempo, calculando o número de iterações que estarão contidas dentro do período e a velocidade do time. Essa medida será utilizada para fins de comparação da capacidade de previsão do modelo.

Baseado na duração de cada projeto, os indicadores foram coletados até a duração indicada. A partir disso, a simulação foi realizada na totalidade do tempo para obter a capacidade preditiva do modelo. A tabela abaixo apresenta o resultado para o projeto *ScrumHalf* e os demais resultados se encontram no apêndice.

	Produto Entregável		***Product Backlog***	
Duração	Média	Desvio padrão	Média	Desvio padrão
100% (1284 dias)	1942.97	79.05	595.01	229.54
75% (961 dias)	1869.02	67.75	1962.41	257.34
50% (662 dias)	1904.79	75.93	3963.33	256.16
25% (342 dias)	1997.39	70.17	1469.1	222.39
6 sprints (94 dias)	2286.12	44.79	2332.72	116.83
3 sprints (45 dias)	2173.14	47.16	2509.29	118.67

Tabela 5 - Valores previstos pelo modelo

O resultado real do projeto obtido foi de 1949.5 pontos entregues e 778.5 pontos restantes no *Product Backlog.* Podemos observar que o modelo é bem preciso em relação ao produto entregável em todos os percentuais recolhidos. No caso de sprints pontuais, o mesmo não está de acordo com o resultado final encontrado. Esse caso, no entanto, é uma exceção dos projetos de longa duração observados. Nos projetos em que o primeiro quarto da duração está aproximado do período de duração de 3 ou 6 sprints todos os casos são consideravelmente precisos no resultado obtido comparado ao real. Devido a longa duração desses projetos, a realidade das primeiras sprints pode ter sido consideravelmente alterada para que as mesmas representem a realidade em que o projeto foi desenvolvido.

O *Product Backlog* possui uma grande divergência quando realizado com indicadores diferentes da totalidade do projeto. A hipótese é de que o modelo ainda é válido para essa simulação e cálculo. O problema, no entanto, está relacionado ao comportamento da adição de pontos e ao descarte observado ao longo da análise de dados, que possuem comportamentos antagônicos: enquanto a adição de pontos é elevada no início do projeto e decai com o tempo, o inverso acontece com o descarte.

Por isso, a função normal pode não ser a mais adequada para a previsibilidade desse artefato o que, conforme visto, não invalida a capacidade de previsão do modelo.

A maneira usual de cálculo de pontos entregues no projeto, considerando a quantidade de sprints e a velocidade média calculada da equipe até o momento pode ser observado na tabela abaixo.

Duração	Produto Entregável
100% (1284 dias)	2833.36
75% (961 dias)	2608.23
50% (662 dias)	2763.16
25% (342 dias)	3043.08
6 sprints (94 dias)	3609.75
3 sprints (45 dias)	3623.45

Tabela 6 - Quantidade de pontos esperados calculado de maneira tradicional

Conforme pode ser observado, o modelo é muito mais capaz de prever o real cenário da quantidade de produtos entregues do que a maneira tradicional de estimativa. Os valores encontrados em cada cenário são muito mais próximos e factíveis a partir do modelo do que pela utilização tradicional. Dessa maneira, mesmo com o projeto no início, o modelo se mostrou perfeitamente capaz de estimar a quantidade pontos que serão entregues após um determinado tempo.

6.Conclusões e trabalhos futuros

6.1.Conclusão

Nesse trabalho tivemos diversas contribuições para a área do desenvolvimento de software baseado no *framework Scrum*. O objetivo dessas contribuições é melhorar a produção de software como um todo, visto que a agilidade foi apontada como um dos fatores que vem trazendo melhoria aos softwares desenvolvidos recentemente. Melhorar a área da agilidade é um caminho para avançar na capacidade e na qualidade de entrega de sistemas.

A partir da revisão da literatura foi observado que trabalhos que tratam de simulação de processo de desenvolvimento baseado em *Scrum* são escassos. Por se tratar da metodologia mais adotada na agilidade se faz necessário uma atenção maior a processos desse tipo. Até hoje, apenas modelos de agilidade genéricos que adotam variáveis de controles foram sugeridos e nenhum representa as etapas e particularidades do *Scrum.*

Um modelo foi apresentado como parte desse trabalho representando as diversas etapas do ciclo e suas particularidades. Desde a fase do planejamento até a revisão, todas as áreas foram cobertas em diferentes modelos que foram reunidos em um modelo completo. Esse modelo foi submetido a uma validação com os especialistas para apreciar sua estrutura, verificando que de fato o modelo atende a um processo genérico baseado em *Scrum*. Nos experimentos podemos ver que também é atendido a validação comportamental com os resultados encontrados coerentes com os resultados das execuções reais.

Os dados utilizados no projeto foram coletados ao longo de cerca de 4 anos ao longo de 6 projetos diferentes. Os indicadores foram retirados da base de dados de projetos executados pela mesma empresa e que foram caracterizados utilizando *Progress Poker* para fornecer um contexto mais completo do ambiente de desenvolvimento do projeto.

A partir do modelo e dos indicadores coletados foram feitos experimentos para compreender a sensibilidade as diferentes entradas que podem ser dadas para simular o desenvolvimento. Variando os indicadores avaliamos a alteração sofrida nos pontos

aprovados e nos pontos restantes do *Product Backlog*, buscando compreender quais os indicadores que mais afetam esses dois reservatórios.

Como etapa final do trabalho, foi testada a capacidade de predição do modelo a partir da execução parcial do projeto. Os indicadores foram coletados até um determinado tempo e o modelo simulou a execução do projeto na sua duração total para tentar prever a quantidade de pontos. Nesses experimentos, em todos os projetos o modelo foi capaz de prever com uma precisão muito melhor do que a maneira tradicional. Em projetos curtos, 3 sprints se mostraram suficiente para prever o resultado final. No caso de projetos de longa duração, poucas sprints não se mostraram um bom indicador, mas isso pode ser explicado por diversos fatores envolvendo o contexto do desenvolvimento dado que muito transcorreu desde as primeiras sprints até a atualidade.

6.2.Trabalhos futuros

Por se tratar de um trabalho com pouca referência na literatura muitas oportunidades de melhoria podem ser adotadas. A intenção desse trabalho é chamar a atenção e oferecer um ponto de partida para outros pesquisadores que se interessarem em seguir no caminho desse estudo. Conforme dito anteriormente, contribuir para a área de agilidade significa melhorar a área do desenvolvimento de software.

Inicialmente, um ponto de melhora encontrado é o modelo. Por se tratar de um modelo inicial, toda contribuição ao mesmo é bem-vinda. A alteração estrutural, inclusão de novos indicadores e novos modelos para que o desempenho seja comparado com o obtido são pontos de melhoria interessante. Esse primeiro resultado visa fornecer um *baseline* e poderá ser utilizado por novos modelos para comparar a eficácia.

Além disso, o modelo foca apenas em indicadores objetivos como a taxa de pontos iniciados ou o percentual de aprovação de pontos finalizados. Um grande espaço de melhoria se encontra na introdução de fatores subjetivos como comunicação, incerteza do negócio ou capacidade dos desenvolvedores. Uma hipótese imaginada é que o indicador objetivo é, na verdade, uma combinação dos fatores subjetivos. Apenas para ilustrar, o fator de aprovação de itens não finalizados poderia, por exemplo, ser descrito como uma combinação entre comunicação com o *Product Owner*, rigor da

aplicação da definição de *Pronto* e capacidade de negociação entre as partes. Os fatores citados anteriormente são apenas para ilustrar um possível trabalho futuro.

Sobre a questão dos indicadores, foi utilizada uma distribuição normal baseada na média e na variância obtida a partir da coleta dos mesmos. Analisando o resultado, essa abordagem de mostrou satisfatória a partir dos dados coletados da execução completa do projeto. Nos experimentos com indicadores de execução parcial, a previsão do *Product Backlog* se mostrou falha. Uma hipótese para isso é o comportamento das taxas de descarte e de criação que não são estáveis ao longo do projeto. Essa hipótese pode ser testada para melhorar a acurácia em relação a previsão desse artefato.

Por fim, o modelo não contempla um conceito da agilidade que é o valor. A análise se resume a quantidade de pontos entregues e aprovados e não o retorno de investimento que esse conjunto de história foi capaz de conceder. Um trabalho futuro que é visto é a inclusão desse conceito em uma extensão de modelo trabalhando não apenas a quantidade de pontos que foram entregues, mas sim o valor gerado no final de um determinado tempo.

Referências

10th Annual State of Agile™ Report. (2015). Recuperado 19 de maio de 2016, de http://stateofagile.com/new.html

Abdel-Hamid, T., & Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods - Review and analysis* (No. 478). VTT PUBLICATIONS.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In *25th International Conference on Software Engineering, 2003. Proceedings* (p. 244–254). http://doi.org/10.1109/ICSE.2003.1201204

Agile Contracts: Money for Nothing and Your Change for Free. (2008). Recuperado 4 de março de 2016, de https://www.scruminc.com/agile-contracts-money-for-nothing-and/

Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods. (2012). Recuperado 27 de novembro de 2015, de http://docplayer.net/2823321-Agile-project-dynamics-a-system-dynamics-investigation-of-agile-software-development-methods.html

Agile Project Management with Scrum. (2004) (1 edition). Redmond, Wash: Microsoft Press.

Agile Scrum Velocity Calculation & FAQs. ([s.d.]). Recuperado 11 de abril de 2016, de https://www.versionone.com/agile-101/agile-project-management-customer-management-best-practices/agile-scrum-velocity/

Agile Software Development Ecosystems. (2002) (1 edition). Boston: Addison-Wesley Professional.

Agile User Stories, Epics and Themes- Mike Cohn - Scrum Alliance. (2014). Recuperado 12 de maio de 2016, de https://www.scrumalliance.org/community/spotlight/mike-cohn/march-2014/agile-user-stories-epics-and-themes

Akerele, O., Ramachandran, M., & Dixon, M. (2013). System Dynamics Modeling of Agile Continuous Delivery Process. In *Agile Conference (AGILE), 2013* (p. 60–63). http://doi.org/10.1109/AGILE.2013.28

Augustine's Laws, Sixth Edition. (1997) (6th edition). Reston, Va: AIAA.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). *Manifesto para Desenvolvimento Ágil de Software*. Recuperado de http://www.agilemanifesto.org/iso/ptbr/

Chaos Manifesto. (2013). Recuperado 19 de maio de 2016, de https://pt.scribd.com/doc/198550543/Chaos-Manifesto-2013

Cocco, L., Mannaro, K., Concas, G., & Marchesi, M. (2011). Simulating Kanban and Scrum vs. Waterfall with System Dynamics. In A. Sillitti, O. Hazzan, E. Bache, & X. Albaladejo (Orgs.), *Agile Processes in Software Engineering and Extreme Programming* (p. 117–131). Springer Berlin Heidelberg. Recuperado de http://link.springer.com/chapter/10.1007/978-3-642-20677-1_9

Cockburn, A., & Williams, L. (2001). Extreme Programming Examined. In G. Succi & M. Marchesi (Orgs.) (p. 223–243). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. Recuperado de http://dl.acm.org/citation.cfm?id=377517.377531

Common Product Owner Traps - Scrum Alliance. (2010). Recuperado 13 de fevereiro de 2016, de

https://www.scrumalliance.org/community/articles/2010/april/common-product-owner-traps

Constantine on Peopleware. (1995) (1 edition). Englewood Cliffs, N.J.: Prentice Hall Ptr.

Defect Prevention: Reducing Costs and Enhancing Quality. (2010). Recuperado 3 de fevereiro de 2016, de http://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/

Essence. (2014). Recuperado 12 de fevereiro de 2016, de http://www.omg.org/spec/Essence/

Forrester, J. W. (1961). *Industrial dynamics*. [Cambridge, Mass.: M.I.T. Press. Recuperado de http://books.google.com/books?id=O1C3AAAAIAAJ

Forrester, J. W. (1993). System Dynamics and the Lessons of 35 Years. In K. B. D. Greene (Org.), *A Systems-Based Approach to Policymaking* (p. 199–240). Springer US. Recuperado de http://link.springer.com/chapter/10.1007/978-1-4615-3226-2_7

França, B. B. N. de, & Travassos, G. H. (2013). Are We Prepared for Simulation Based Studies in Software Engineering Yet? *CLEI Electronic Journal*, *16*(1), 9–9.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. *Computer*, *34*(9), 120–127. http://doi.org/10.1109/2.947100

impedimento: Dicionário Português Online: Moderno Dicionário da Língua Portuguesa. ([s.d.]). Recuperado 3 de fevereiro de 2016, de http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues-portugues&palavra=impedimento

Impediments are holding back the team. (2011). Recuperado 3 de fevereiro de 2016, de https://agilepainrelief.com/notesfromatooluser/2011/12/scrummaster-tales-impediments-are-holding-back-the-team.html#.VrI5VPkrLcc

Impediments vs. blockers: Common scenarios for Scrum Team. (2016, março 2). Recuperado 19 de maio de 2016, de https://lookforwardconsulting.com/2016/03/02/impediments-vs-blockers-why-make-the-distinction/

Incerteza nos Pontos de História. (2011). Recuperado de http://blog.myscrumhalf.com/2011/04/incerteza-nos-pontos-de-historia/

James W Grenning, J. G. (2002). Planning Poker Planning Poker or How to avoid analysis paralysis while release planning.

Karl E. Weick. (2015). Karl E. WEICK (1979), The Social Psychology of Organizing, Second Edition. *M@n@gement*, *18*(2), 189–193.

Lui, K. M., & Chan, K. C. C. (2003). When Does a Pair Outperform Two Individuals? In *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering* (p. 225–233). Berlin, Heidelberg: Springer-Verlag. Recuperado de http://dl.acm.org/citation.cfm?id=1763875.1763910

Manifesto for Agile Software Development. (2001). Recuperado 3 de fevereiro de 2016, de http://www.agilemanifesto.org/

Marcelo Nascimento Costa, M. K. (2008). Melhorando Processos de Software Através de Análise Causal de Defeitos. *Engenharia de Software Magazine*, *I*(3), 32–37.

Maria, A. (1997). Introduction to Modeling and Simulation. In *Proceedings of the 29th Conference on Winter Simulation* (p. 7–13). Washington, DC, USA: IEEE Computer Society. http://doi.org/10.1145/268437.268440

Maxwell, K. D., & Forselius, P. (2000). Benchmarking software development productivity. *IEEE Software*, *17*(1), 80–88. http://doi.org/10.1109/52.820015

Merrill, D., Collofello, C. J., Urban, C. J., & Faltz, C. L. (1996). *Training Software Development Project Managers with a Software Project Simulator*.

Murugaiyan, M. S., & Balaji, S. (2012). Succeeding with Agile software development. In *2012 International Conference on Advances in Engineering, Science and Management (ICAESM)* (p. 162–165).

Pair Programming. (1999). Recuperado 29 de março de 2016, de http://www.extremeprogramming.org/rules/pair.html

Principles behind the Agile Manifesto. (2001). Recuperado 3 de fevereiro de 2016, de http://www.agilemanifesto.org/principles.html

Progress Poker Flyer. (2014). Recuperado 3 de fevereiro de 2016, de https://www.ivarjacobson.com/publications/brochures/progress-poker-flyer

QSM. (2009, março 18). Function Point Languages Table [Text]. Recuperado 19 de maio de 2016, de http://www.qsm.com/resources/function-point-languages-table

Reel, J. S. (1999). Critical success factors in software projects. *IEEE Software*, *16*(3), 18–23. http://doi.org/10.1109/52.765782

Robinson, S. (1997). Simulation Model Verification and Validation: Increasing the Users' Confidence. In *Proceedings of the 29th Conference on Winter Simulation* (p. 53–59). Washington, DC, USA: IEEE Computer Society. http://doi.org/10.1145/268437.268448

Ross, S. M. (1990). *A Course in Simulation*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Scrum Guide. (2013). Recuperado 3 de fevereiro de 2016, de http://www.scrumguides.org/download.html

Scrum History. (2013). Recuperado 3 de fevereiro de 2016, de http://www.scrumguides.org/history.html

Scrum (software development). (2016, maio 9). In *Wikipedia, the free encyclopedia*. Recuperado de https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=719402490

ScrumHalf. (2010). Recuperado 4 de março de 2016, de http://myscrumhalf.com/?lang=pt

STERMAN J.D. (1988). *A Skeptic S Guide to Computer Models* (No. D-4010). MIT System Dynarnics Group, Cambridge, MA.

Tomaz, L. (2015). Validação do modelo.

What are impediments? (2013). Recuperado 3 de fevereiro de 2016, de http://www.leanagiletraining.com/impediments/what-are-impediments/

What is SEMAT? (2014). Recuperado 3 de fevereiro de 2016, de http://semat.org/what-is-semat-

Wilensky, U. (1999). NetLogo itself. Recuperado 30 de maio de 2016, de http://ccl.northwestern.edu/netlogo/

Wu, M., & Yan, H. (2009). Simulation in Software Engineering with System Dynamics: A Case Study. *Journal of Software*, *4*(10). http://doi.org/10.4304/jsw.4.10.1127-1135

Zhang, H., Kitchenham, B., & Pfahl, D. (2008). Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review. In Q. Wang, D. Pfahl, & D. M. Raffo (Orgs.), *Making Globally Distributed Software Development a*

Success Story (p. 345–356). Springer Berlin Heidelberg. Recuperado de http://link.springer.com/chapter/10.1007/978-3-540-79588-9_30

A. Apêndice

SCRUMHALF

Descrição: O *ScrumHalf* é uma ferramenta conhecida na agilidade, principalmente pelos praticantes do *Scrum*. A equipe era responsável pelo desenvolvimento, operação e suporte ao cliente, contemplando todas as etapas. O *Product Owner* era um membro interno da empresa e um dos membros da validação do modelo desse projeto. O projeto teve uma longa duração e foi desenvolvido utilizando J2EE, javascript e banco de dados MySQL Server.

Duração: 1284 dias

Pontos restantes no *Product Backlog:* 778.5

Pontos aprovados: 1949.5

INDICADOR	MÉDIA	DESVIO PADRÃO
CRIAÇÃO	2.16	8.11
DESCARTE	0.53	5.84
DURAÇÃO SPRINT	15.85	2.68
PONTOS POR SPRINT	33.1	13.42
INICIALIZAÇÃO	0.86	2.57
FINALIZAÇÃO	1.34	3.31
IMPEDIMENTOS	0.49	2.3
RESOLUÇÃO	0.5	2.41
APROVAÇÃO FINALIZADAS	95%	0.11
APROVAÇÃO NÃO FINALIZADAS	27%	0.4

Tabela 7. Indicadores coletados do projeto *ScrumHalf*

Projeto A

Descrição: Esse primeiro projeto possui como escopo o desenvolvimento de um portal de serviços para uma grande empresa de seguros. Além da consulta poder ser efetuada pelo portal WEB, haviam também a existência de *web services.* O projeto possuía cliente externo, escopo aberto e a equipe não era responsável pela infraestrutura do ambiente de produção. Foi desenvolvido utilizando J2EE e banco de dados SQL Server.
Duração: 1024 dias
Pontos restantes no *Product Backlog:* 1157
Pontos aprovados: 2391

INDICADOR	MÉDIA	DESVIO PADRÃO
CRIAÇÃO	3.07	10.79
DESCARTE	0.44	6.95
DURAÇÃO SPRINT	15.75	5.48
PONTOS POR SPRINT	46.54	15.12
INICIALIZAÇÃO	1.65	4.01
FINALIZAÇÃO	2.82	4.97
IMPEDIMENTOS	0.48	2.3
RESOLUÇÃO	0.5	2.07
APROVAÇÃO FINALIZADAS	96%	0.11
APROVAÇÃO NÃO FINALIZADAS	20%	0.36

Tabela 8. Indicadores coletados do projeto A

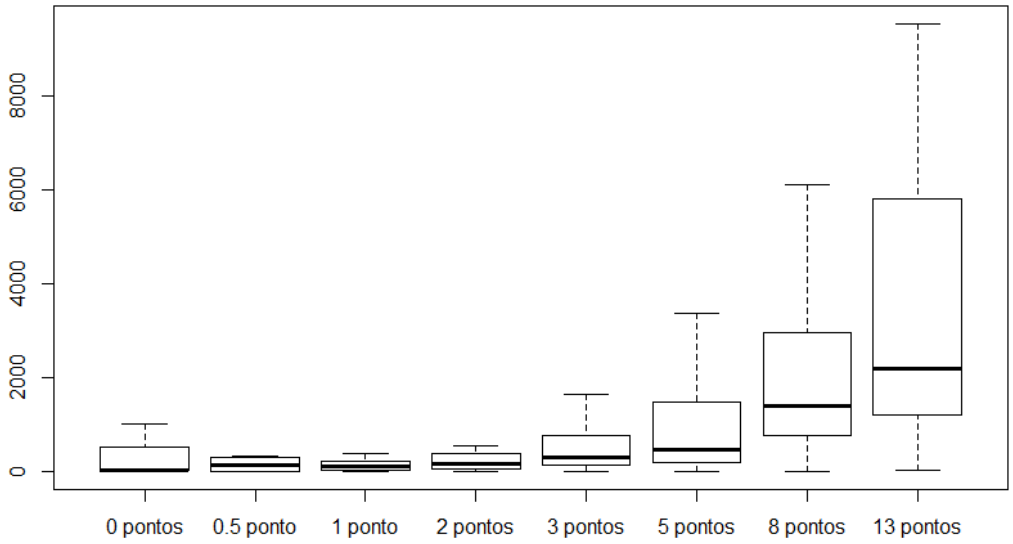

Figura 22. Análise da conversão de pontos para minutos para o projeto A

Resultado obtido pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Caso base	2162.31	84.85	1065.79	275.82
Impedimentos zerados	2229.58	89.27	931.15	286.71
Resolução zerada	1987.47	70.77	1195.78	243.79
Change for free	2174.98	87.07	1070.75	260.83

Tabela 9. Comparativo entre cenários para o projeto A

Gráficos obtidos pelos experimentos:

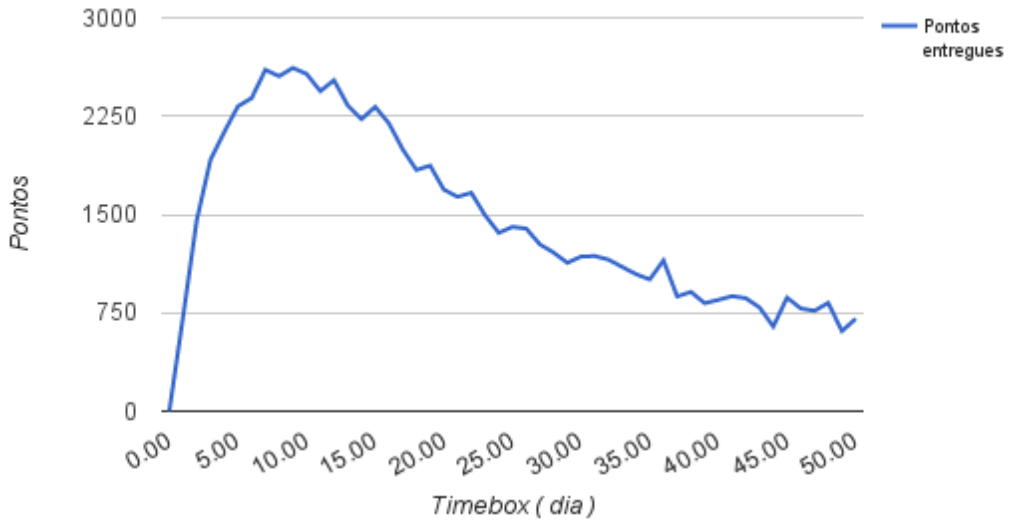

Figura 23. Análise de sensibilidade do *timebox* sem alteração da velocidade para o projeto A

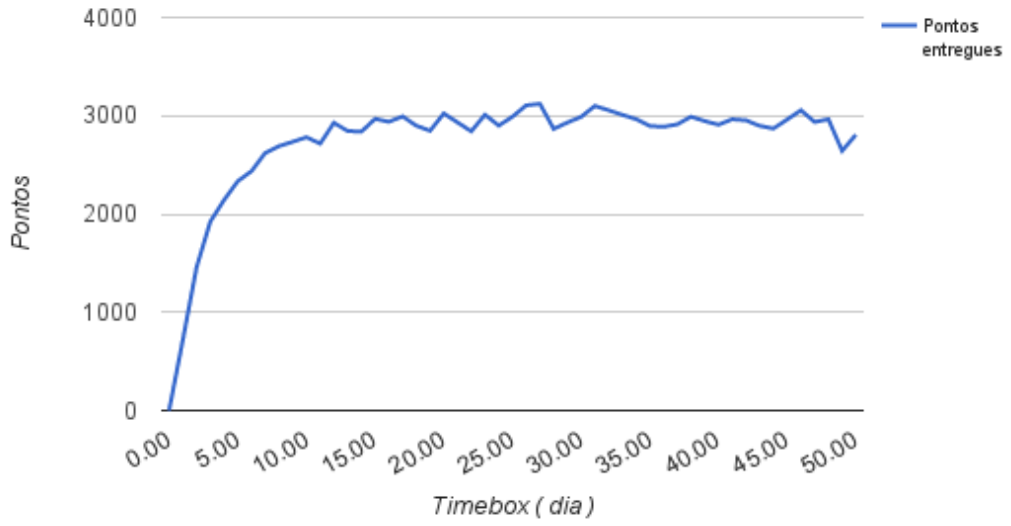

Figura 24. Análise de sensibilidade do *timebox* com alteração da velocidade para o projeto A

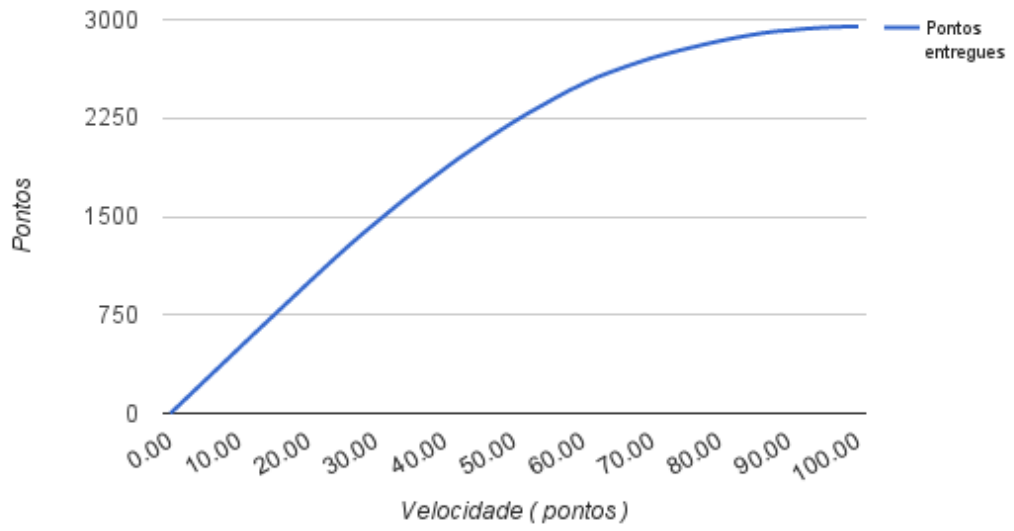

Figure 25. Análise de sensibilidade da velocidade para o projeto A

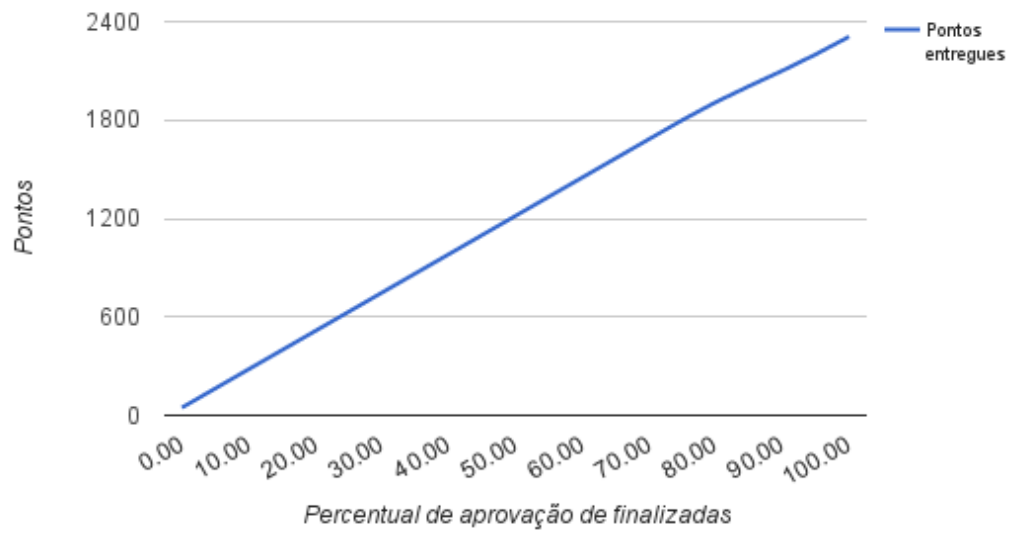

Figura 26. Análise de sensibilidade da aprovação de pontos finalizados para o projeto A

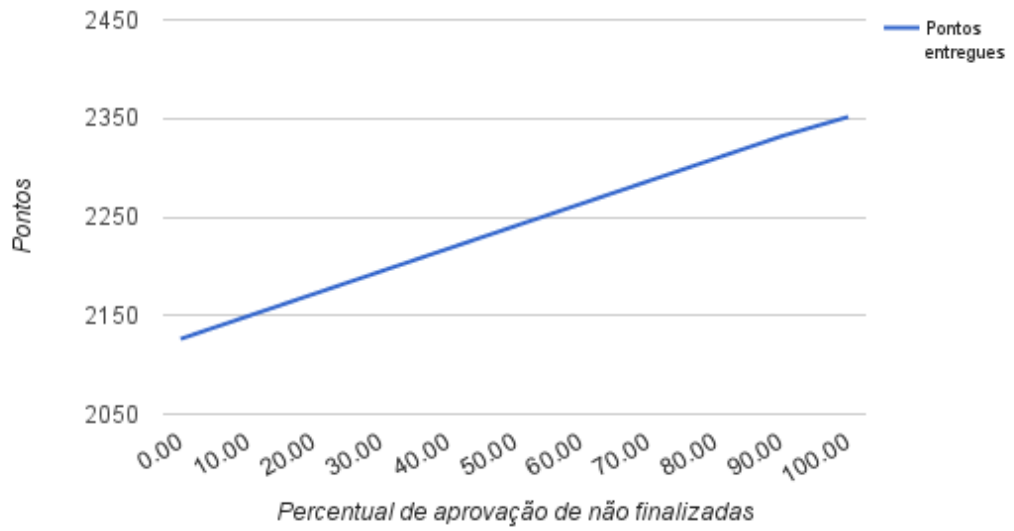

Figura 27. Análise de sensibilidade da aprovação de pontos não finalizados para o projeto A

Previsão obtida pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
100% (1681 dias)	2162.31	84.85	1065.79	275.82
75% (1261 dias)	2262.46	79.01	905.39	258.75
50% (840 dias)	2276.36	61.06	606.37	246.75
25% (420 dias)	2230.09	76.71	7109.82	283.98
3 sprints (54 dias)	1824.01	63.06	5837.65	201.19
6 sprints (104 dias)	1968.99	40.18	3800.57	196.54

Tabela 10. Valores previstos para o projeto A

Projeto B

Descrição: Como projeto encomendado por um cliente, foi solicitado um sistema de gestão acadêmica para cursos de extensão *lato-sensu* de uma escola de negócios. A equipe era responsável tanto pelo desenvolvimento do produto quanto pela operação do mesmo em produção. O suporte e auxílio ao usuário era inicialmente prestado pelo cliente e encaminhado a equipe quando relevante, mas a mesma interrompia o desenvolvimento para corrigir erros ou eventuais quedas em produção. Foi desenvolvido utilizando J2EE e banco de dados MySQL Server.

Duração: 324 dias

Pontos restantes no *Product Backlog:* 320

Pontos aprovados: 472

INDICADOR	MÉDIA	DESVIO PADRÃO
CRIAÇÃO	2.17	7.96
DESCARTE	0.69	3.4
DURAÇÃO SPRINT	18	4.86
PONTOS POR SPRINT	33.1	13.42
INICIALIZAÇÃO	1.06	2.98
FINALIZAÇÃO	1.57	3.75
IMPEDIMENTOS	0.33	1.42
RESOLUÇÃO	0.43	1.76
APROVAÇÃO FINALIZADAS	94%	0.15
APROVAÇÃO NÃO FINALIZADAS	5%	0.16

Tabela 11. Indicadores coletados do projeto B

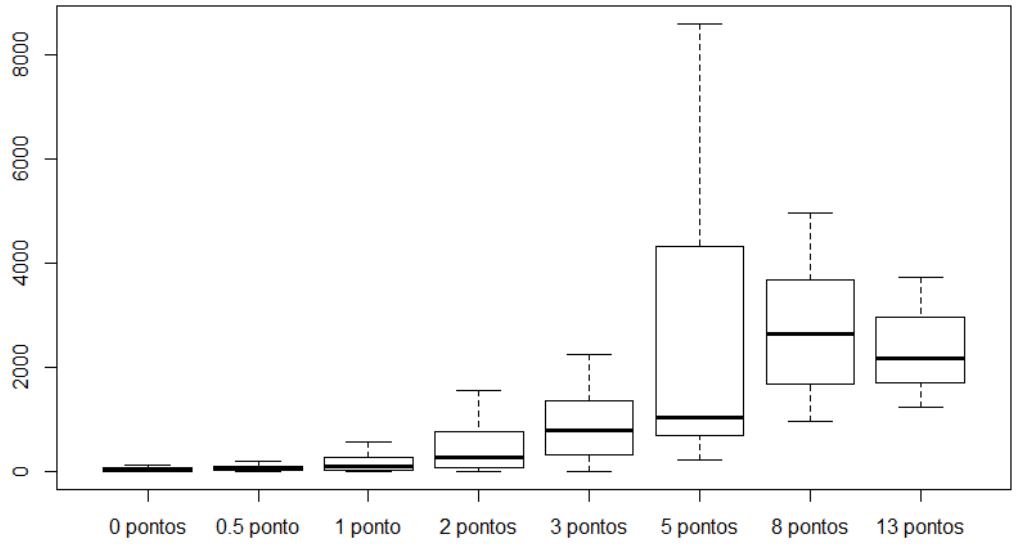

Figura 28. Análise da conversão de pontos para minutos para o projeto B

Resultado obtido pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Caso base	464.81	54.81	707.93	96.26
Impedimentos zerados	478.31	50.37	702.45	100.08
Resolução zerada	409.27	41.92	765.17	125.35
Change for free	459.7	50.33	728.18	128.61

Tabela 12. Comparativo entre cenários para o projeto B

Gráficos obtidos pelos experimentos:

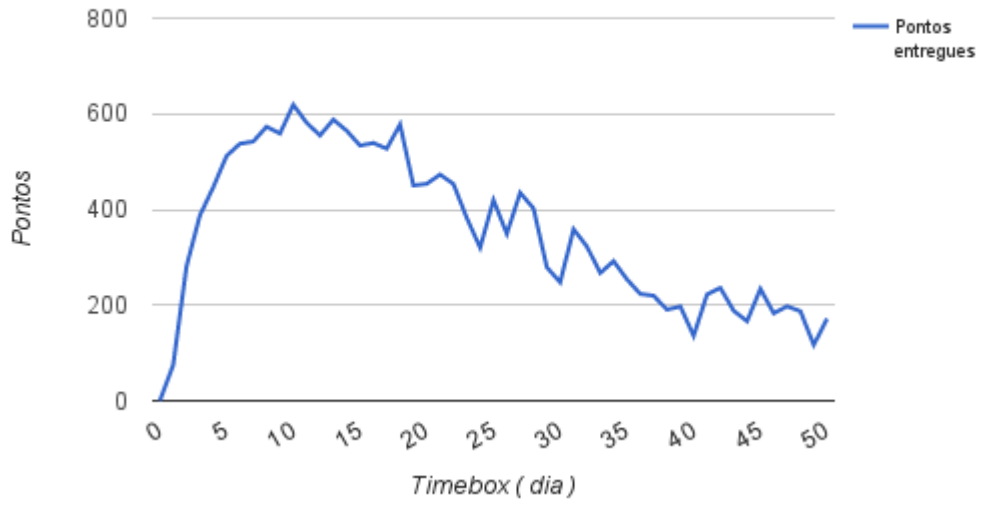

Figura 29. Análise de sensibilidade do *timebox* sem alteração da velocidade para o projeto B

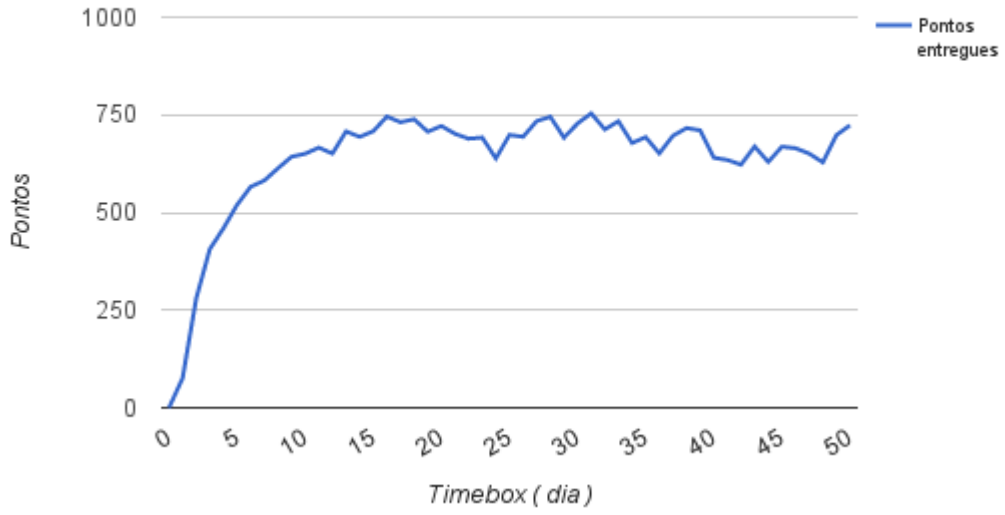

Figura 30. Análise de sensibilidade do *timebox* com alteração da velocidade para o projeto B

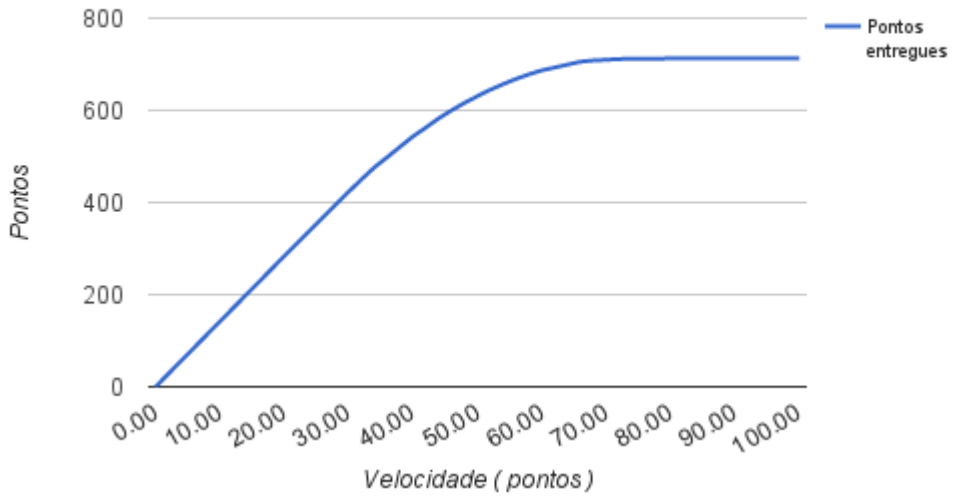

Figura 31. Análise de sensibilidade da velocidade para o projeto B

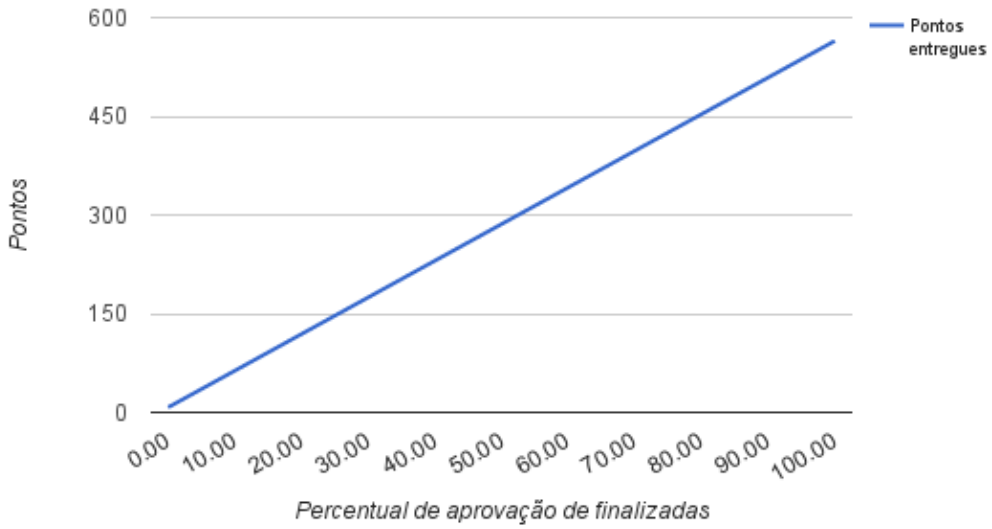

Figura 32. Análise de sensibilidade da aprovação de pontos finalizados para o projeto B

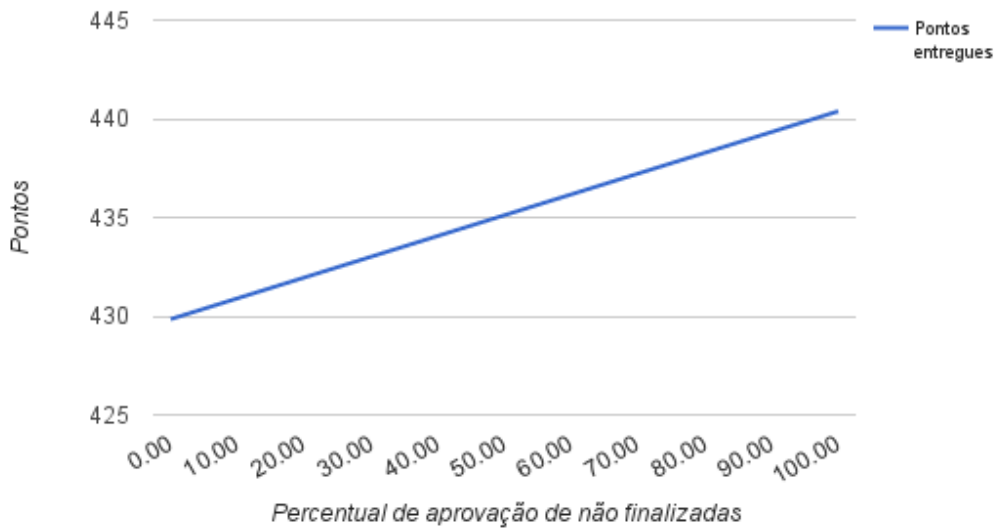

Figura 33. Análise de sensibilidade da aprovação de pontos não finalizados para o projeto B

Previsão obtida pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
100% (1681 dias)	464.81	45.45	707.93	96.26
75% (1261 dias)	499.14	41.99	896.9	125.79
50% (840 dias)	417.61	48.72	982.94	136.64
25% (420 dias)	390.03	32.89	1109.45	105.07
3 sprints (54 dias)	358.62	42.49	2010.07	129.78
6 sprints (104 dias)	426.12	40.19	966.87	117.04

Tabela 13. Valores previstos para o projeto B

Projeto C

Descrição: Com a *expertise* adquirida durante a realização do projeto B, a empresa decidiu levar adiante um produto de prateleira que atendesse a gestão acadêmica de cursos *lato sensu* e *stricto sensu* de maneira genérica e decidiu criar um produto que era o objetivo do projeto C. O cliente desse projeto era interno, ou seja, os requisitos eram dados por uma pessoa da empresa que entendia o mercado. A equipe era responsável, além do desenvolvimento, da operação e suporte do *software.* Foi desenvolvido utilizando J2EE e banco de dados MySQL Server.

Duração: 206 dias

Pontos restantes no *Product Backlog:* 182.5

Pontos aprovados: 126

INDICADOR	MÉDIA	DESVIO PADRÃO
CRIAÇÃO	1.05	4.95
DESCARTE	0.38	2.42
DURAÇÃO SPRINT	15.84	2.73
PONTOS POR SPRINT	15.27	5.15
INICIALIZAÇÃO	0.33	1.12
FINALIZAÇÃO	0.5	1.57
IMPEDIMENTOS	0.2	1.09
RESOLUÇÃO	0.23	1.13
APROVAÇÃO FINALIZADAS	85%	0.26
APROVAÇÃO NÃO FINALIZADAS	14%	0.31

Tabela 14. Indicadores coletados do projeto C

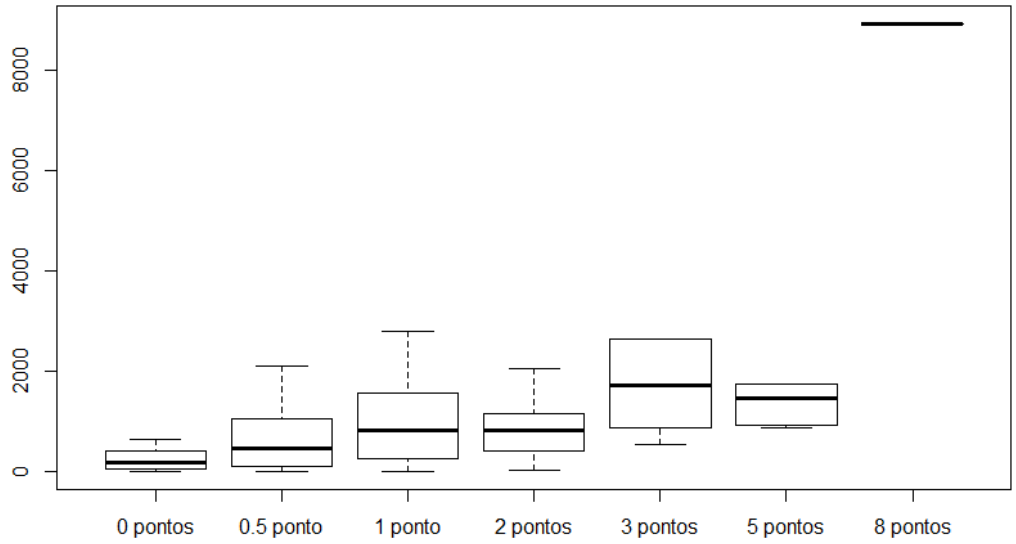

Figura 34. Análise da conversão de pontos para minutos para o projeto C

Resultado obtido pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Caso base	109.48	14.18	257.8	50.06
Impedimentos zerados	116.98	15.8	255.81	49.61
Resolução zerada	102.75	11.88	252.95	50.08
Change for free	113.3	13.93	255.88	55.5

Tabela 15. Comparativo entre cenários para o projeto C

Gráficos obtidos pelos experimentos:

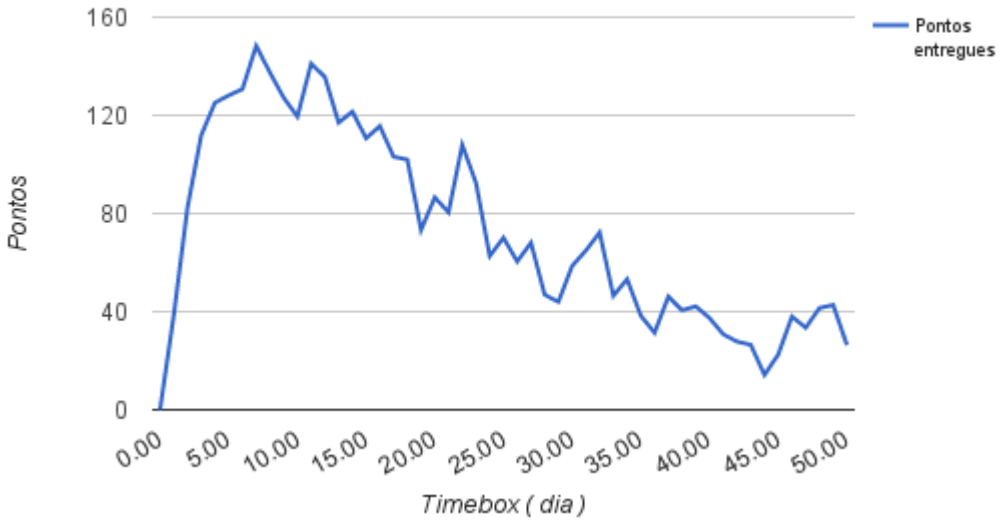

Figura 35. Análise de sensibilidade do *timebox* sem alteração da velocidade para o projeto C

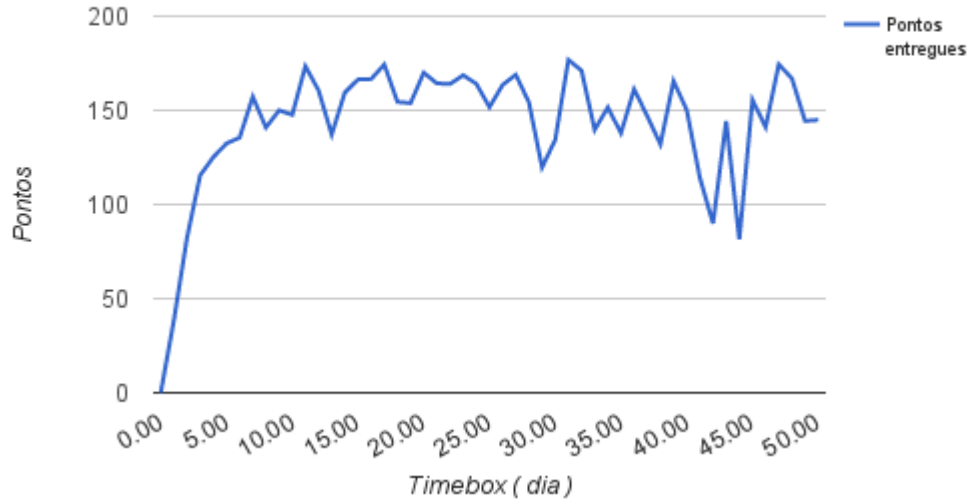

Figura 36. Análise de sensibilidade do timebox com alteração da velocidade para o projeto C

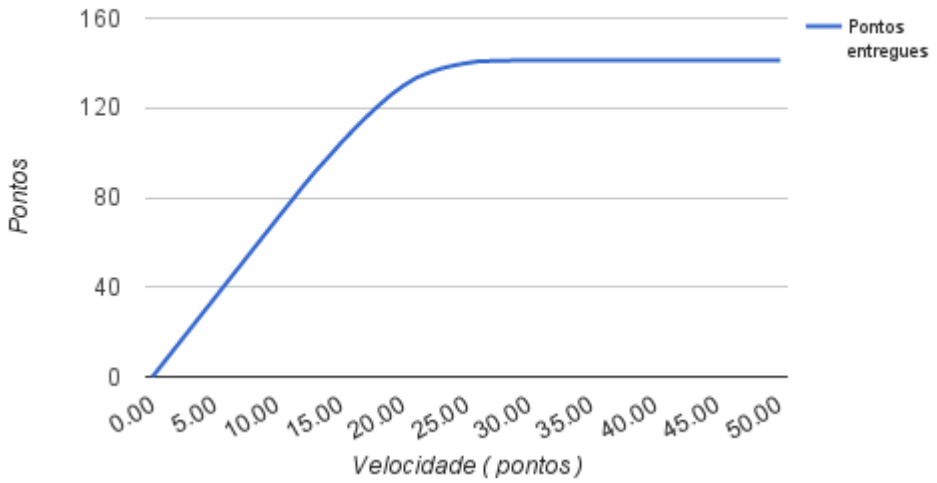

Figura 37. Análise de sensibilidade da velocidade para o projeto C

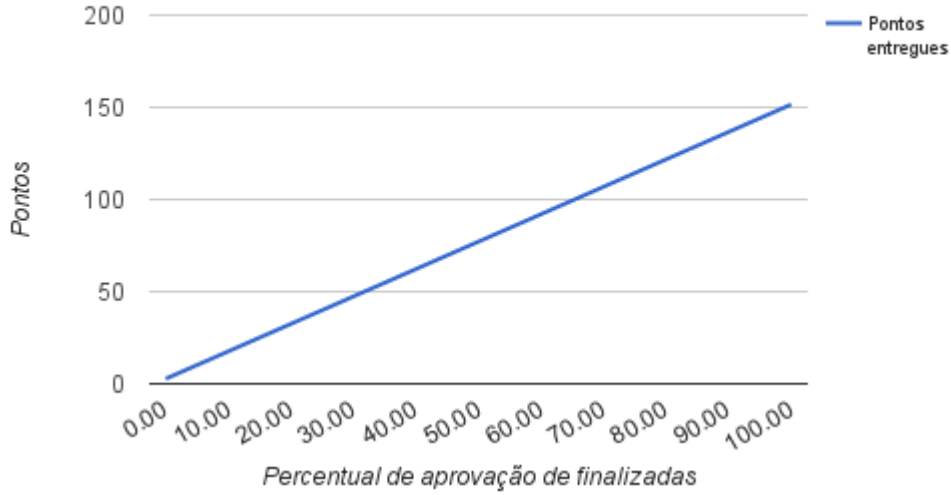

Figura 38. Análise de sensibilidade da aprovação de pontos finalizados para o projeto C

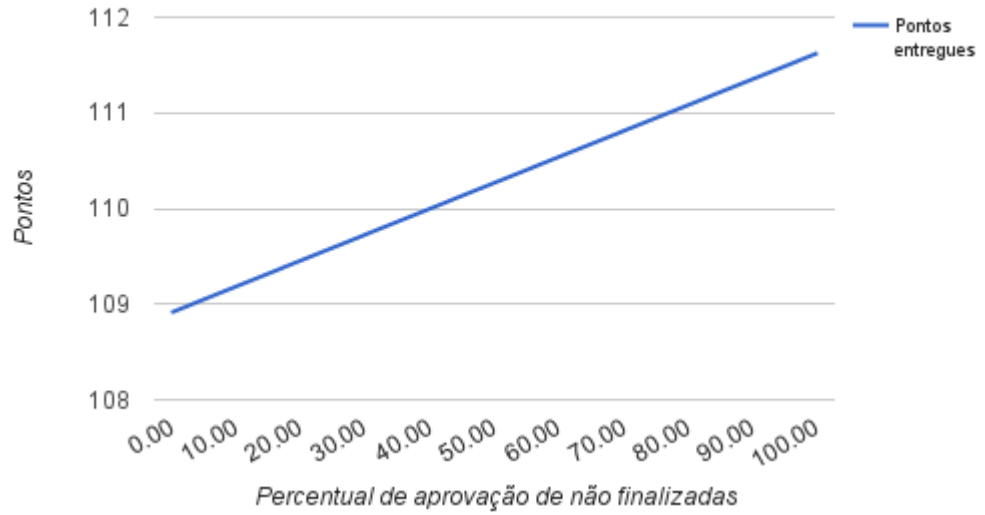

Figura 39. Análise de sensibilidade da aprovação de pontos não finalizados para o projeto C

Previsão obtida pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
100% (1681 dias)	109.48	14.18	257.8	50.06
75% (1261 dias)	145.96	13.1	401.36	49.91
50% (840 dias)	142.4	12.79	546.78	76.02
25% (420 dias)	143.92	12.07	776.78	77.52
3 sprints (54 dias)	134.55	12.29	1773.41	116.36
6 sprints (104 dias)	113.06	12.33	884.71	93.31

Tabela 16. Valores previstos para o projeto C

Projeto D

Descrição: Esse projeto é uma extensão do projeto B, incluindo adendos ao que foi entregue ao final. O escopo do projeto estava bem definido e foi estimado e desenvolvido por boa parte da equipe que integrou a outra etapa do projeto. As restrições e as tecnologias utilizadas foram as mesmas do projeto que antecedeu.

Duração: 167 dias

Pontos restantes no *Product Backlog:* 73

Pontos aprovados: 72

INDICADOR	MÉDIA	DESVIO PADRÃO
CRIAÇÃO	1.06	4.2
DESCARTE	0.41	2.45
DURAÇÃO SPRINT	15.18	2.64
PONTOS POR SPRINT	9.68	2.58
INICIALIZAÇÃO	0.14	0.59
FINALIZAÇÃO	0.42	1.21
IMPEDIMENTOS	0.13	0.92
RESOLUÇÃO	0.17	0.81
APROVAÇÃO FINALIZADAS	98%	0.05
APROVAÇÃO NÃO FINALIZADAS	21%	0.4

Tabela 17. Indicadores coletados do projeto D

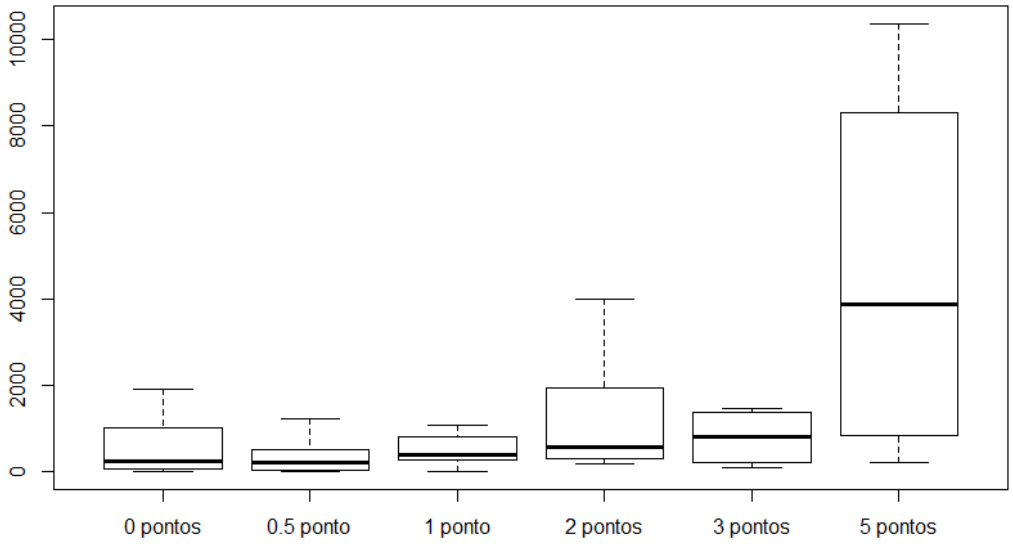

Figura 29. Análise da conversão de pontos para minutos para o projeto D

Resultado obtido pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Caso base	74.57	6.18	147.58	38.09
Impedimentos zerados	76.57	7.87	140.99	39.4
Resolução zerada	67.21	6.67	153.01	36.39
Change for free	74.45	6.69	141.31	46.51

Tabela 18. Comparativo entre cenários para o projeto D

Gráficos obtidos pelos experimentos:

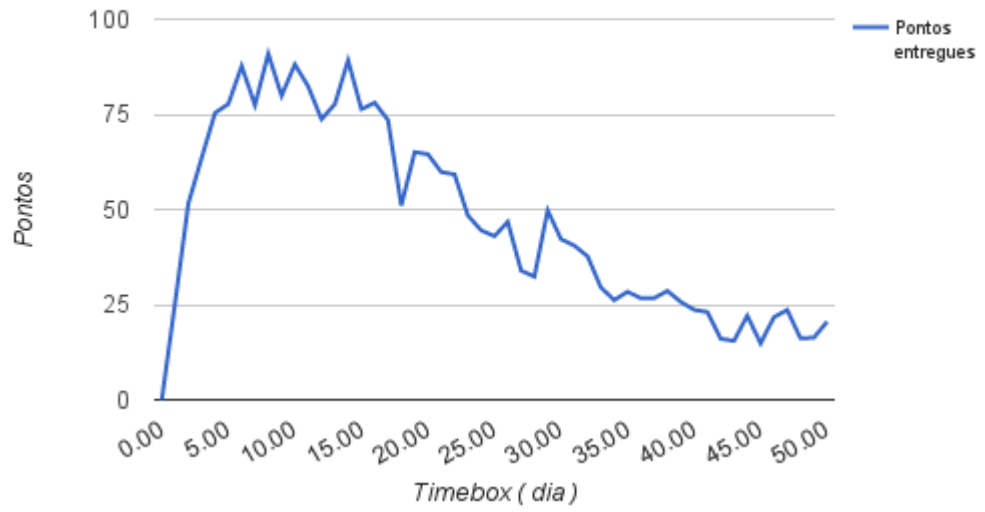

Figura 41. Análise de sensibilidade do *timebox* sem alteração da velocidade para o projeto D

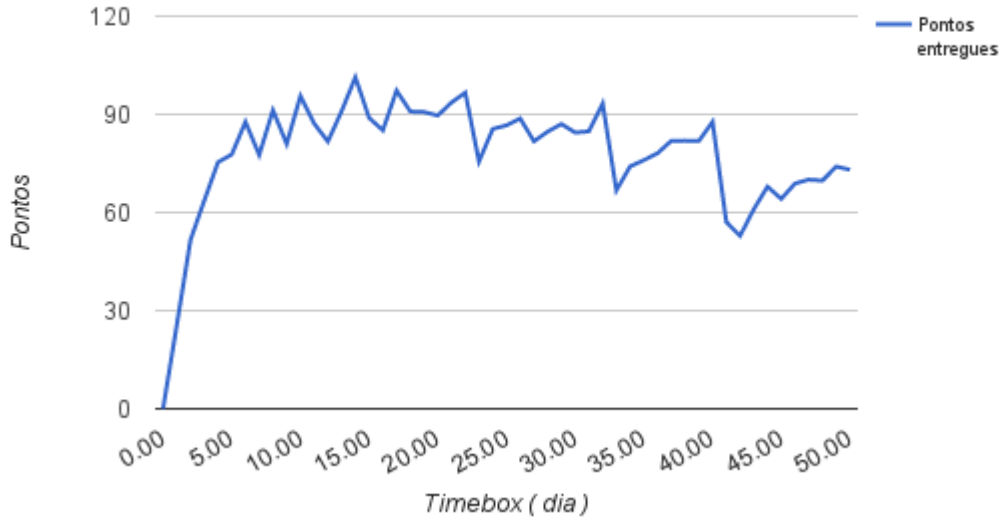

Figura 42. Análise de sensibilidade do *timebox* com alteração da velocidade para o projeto D

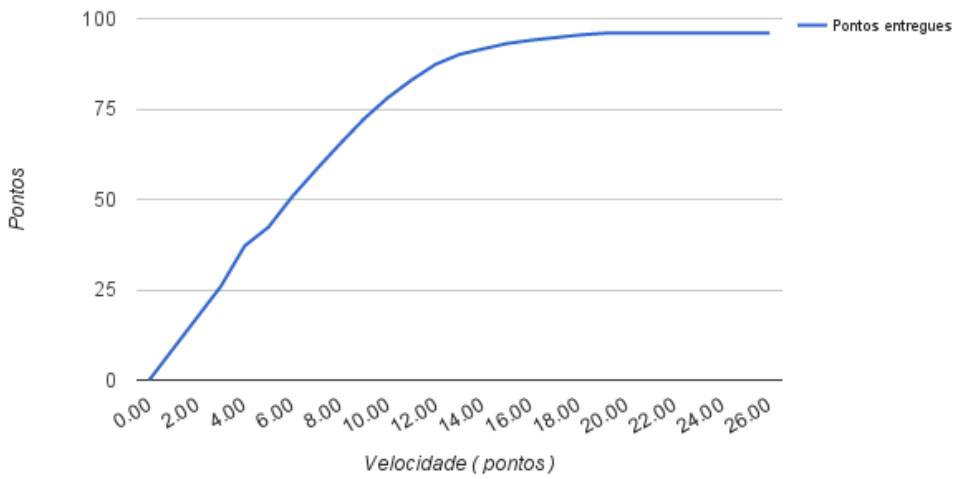

Figura 43. Análise de sensibilidade da velocidade para o projeto D

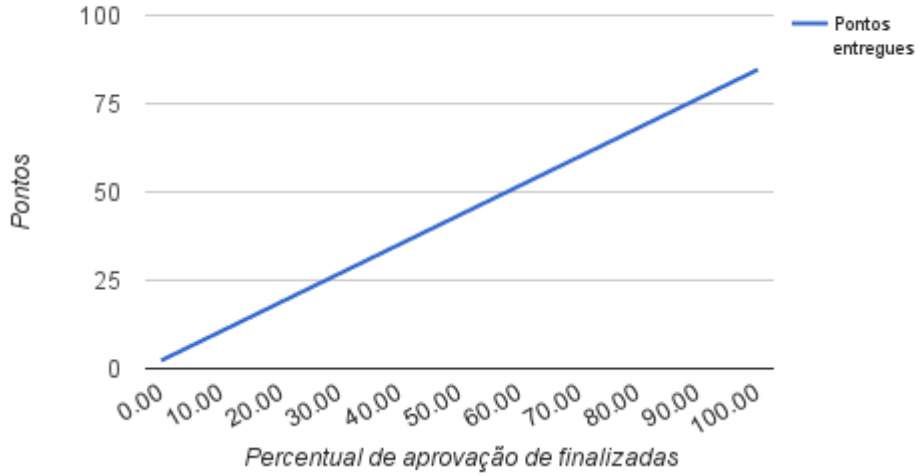

Figura 44. Análise de sensibilidade da aprovação de pontos finalizados para o projeto D

Figura 45. Análise de sensibilidade da aprovação de pontos não finalizados para o projeto D

Previsão obtida pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
100% (1681 dias)	74.57	6.18	147.58	38.09
75% (1261 dias)	65.28	5.17	33.08	20.36
50% (840 dias)	71.52	3.68	65.07	23.64
25% (420 dias)	65.34	3.36	137.46	26.14
3 sprints (54 dias)	64.41	3.73	127.56	27.31
6 sprints (104 dias)	71.58	3.56	73.43	24.76

Tabela 19. Valores previstos para o projeto D

Projeto E

Descrição: Como fruto do trabalho de doutorado de fundadores da empresa, o projeto E foi desenvolvido com a intenção de gerar um produto de suporte à gestão organizacional focado na estratégia. É constituído de uma suíte de aplicativos de forma integrada. O produto foi desenvolvido, mas não chegou a ser implantado em produção e, por isso, a equipe era responsável apenas pelo desenvolvimento. Foi desenvolvido utilizando J2EE e banco de dados MySQL Server.

Duração: 215 dias

Pontos restantes no *Product Backlog:* 447.5

Pontos aprovados: 293

INDICADOR	MÉDIA	DESVIO PADRÃO
CRIAÇÃO	2.92	9.23
DESCARTE	0.34	1.74
DURAÇÃO SPRINT	16.54	4.76
PONTOS POR SPRINT	29.27	8.14
INICIALIZAÇÃO	0.89	2.49
FINALIZAÇÃO	1.39	3.11
IMPEDIMENTOS	0.1	0.83
RESOLUÇÃO	0.1	0.83
APROVAÇÃO FINALIZADAS	90%	0.11
APROVAÇÃO NÃO FINALIZADAS	0%	0

Tabela 20. Indicadores coletados do projeto E

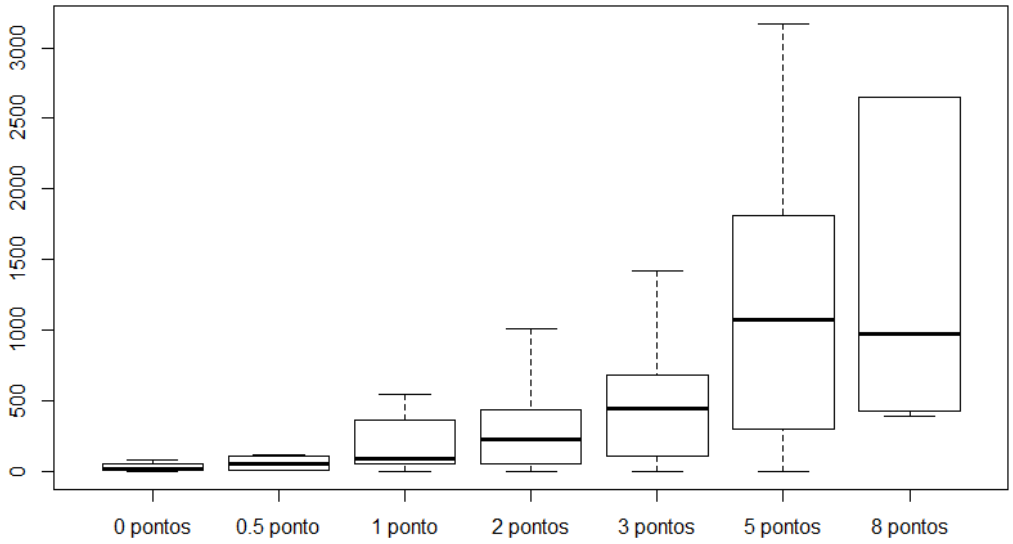

Figura 46. Análise da conversão de pontos para minutos para o projeto E

Resultado obtido pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Caso base	249.92	26.8	520.65	68.18
Impedimentos zerados	255.68	29.24	509.63	67.27
Resolução zerada	226.71	28.47	534.15	64.05
Change for free	255.09	26.07	510.94	67.57

Tabela 21. Comparativo entre cenários para o projeto E

Gráficos obtidos pelos experimentos:

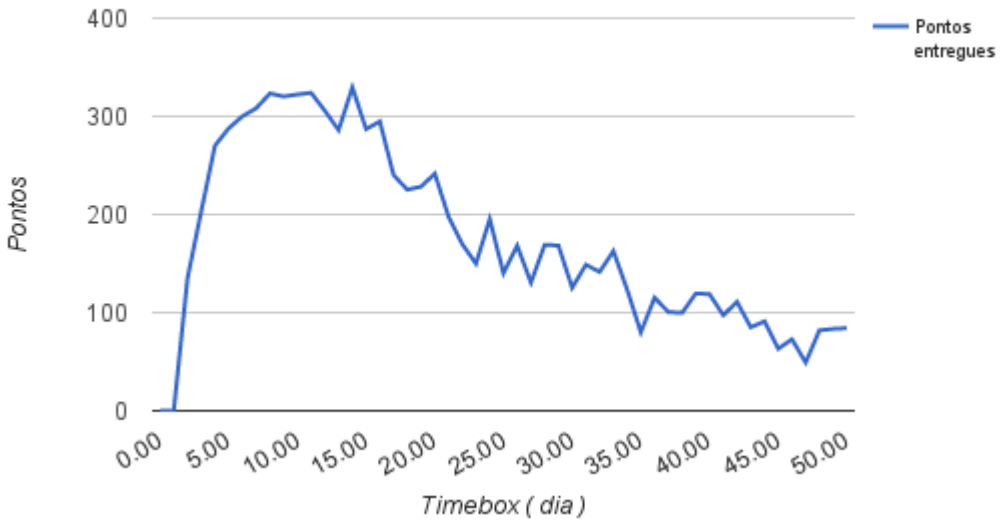

Figura 47. Análise de sensibilidade do *timebox* sem alteração da velocidade para o projeto E

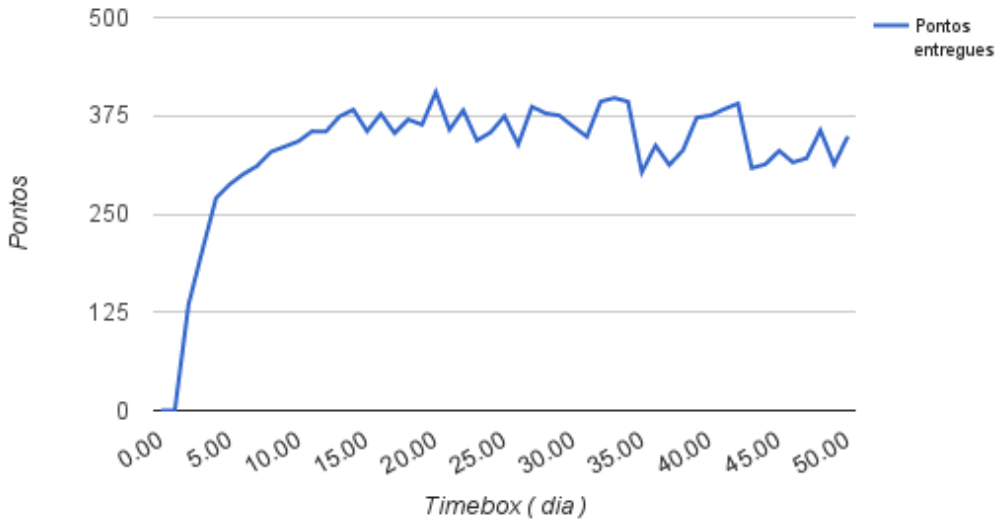

Figura 48. Análise de sensibilidade do *timebox* com alteração da velocidade para o projeto E

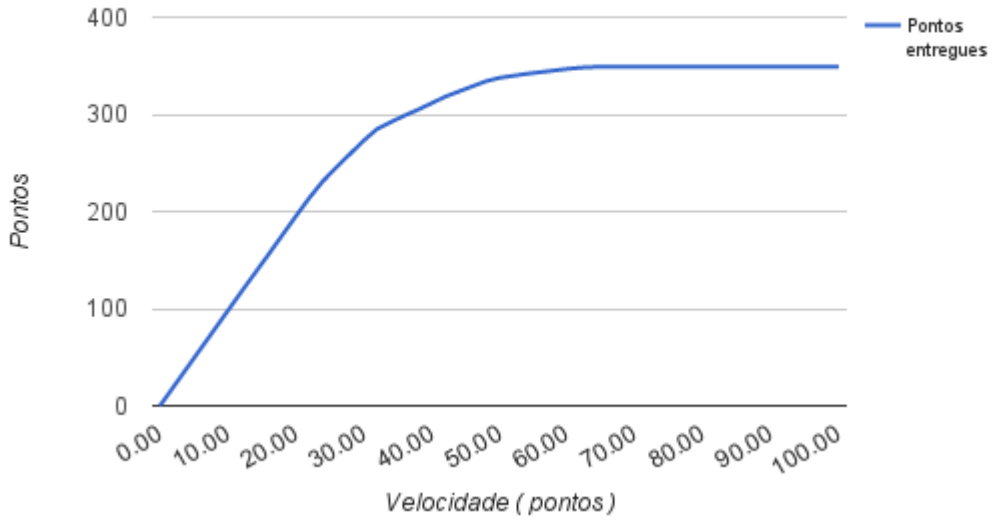

Figura 49. Análise de sensibilidade da velocidade para o projeto E

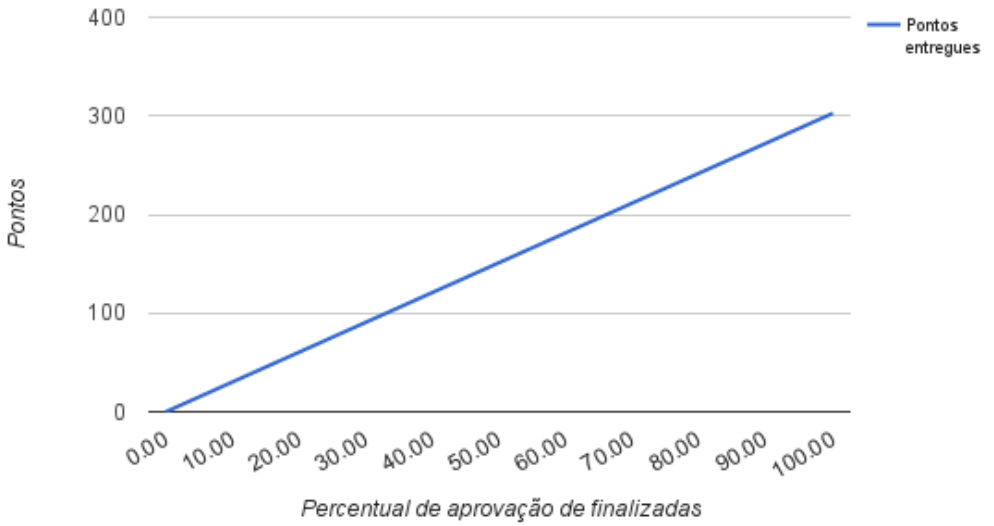

Figura 50. Análise de sensibilidade da aprovação de pontos finalizados para o projeto E

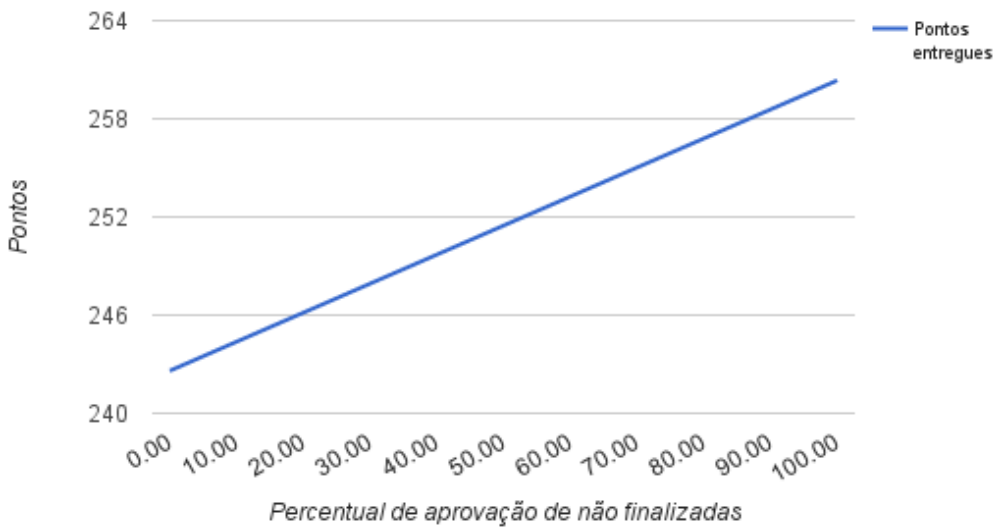

Figura 51. Análise de sensibilidade da aprovação de pontos não finalizados para o projeto E

Previsão obtida pela simulação:

	Produto Entregável		***Product Backlog***	
	Média	Desvio padrão	Média	Desvio padrão
100% (1681 dias)	249.92	26.8	520.65	60.85
75% (1261 dias)	246.54	28.53	426.18	54.88
50% (840 dias)	305.8	22.98	152.22	37.62
25% (420 dias)	319.61	17.99	337.26	43.7
3 sprints (54 dias)	331.11	17.95	1063.55	70.17
6 sprints (104 dias)	308.79	24.68	691.42	71.27

Tabela 22. Valores previstos para o projeto E