UNIVERSIDADE FEDERAL DE SANTA CATARINA

Análise Comparativa do Impacto dos Artefatos de Requisitos no Processo de Desenvolvimento de Software

Guilherme Mitsuo Tatibana

Florianópolis – SC

2017/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Análise Comparativa do Impacto dos Artefatos de Requisito no Processo de Desenvolvimento de Software

Guilherme Mitsuo Tatibana

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Florianópolis – SC
2017/1

Guilherme Mitsuo Tatibana

Análise Comparativa do Impacto dos Artefatos de Requisito no Processo de Desenvolvimento de Software

Trabalho de conclusão de curso apresentado com parte dos requisitos para obtenção do grau de Bacharel em Sistema de Informação

Orientadora: Fabiane Barreto Vavassori Benitti

Banca Examinadora

Rogério Tadeu de Oliveira Lacerda

Jean Carlo Rossa Hauck

Sumário

Lista de figuras

Lista de tabelas

Resumo

As constantes mudanças no ambiente em que a maioria das organizações opera é um desafio às abordagens tradicionais da engenharia de requisitos. Empresas de desenvolvimento de software lidam frequentemente com requisitos que tendem a evoluir rapidamente e muitas vezes se tornam obsoletos até mesmo antes da conclusão do projeto. O processo de desenvolvimento de software segundo a engenharia de software se compõe por diversas disciplinas, sendo a engenharia de requisitos uma delas. A engenharia de requisitos é o processo que tem por objetivo identificar, analisar, documentar e validar os requisitos do produto a ser desenvolvido. Dentro do contexto de metodologias ágeis, uma filosofia que surgiu pela necessidade da indústria de desenvolvimento de software de se adaptar às constantes mudanças e evoluções que ocorrem durante o ciclo de vida de desenvolvimento e com foco no produto a ser desenvolvido, a engenharia de requisitos não possui um único modelo estabelecido a ser seguido. Em contrapartida, os métodos tradicionais são considerados demasiadamente burocráticos porém, eficientes quando mudanças no escopo não ocorrem com frequência. Este trabalho de conclusão de curso tem como objetivo analisar e mensurar os efeitos causados dentro do processo de desenvolvimento de software abordando diferentes práticas da engenharia de requisitos utilizando metodologias ágeis, definindo, dentro do contexto da pesquisa, quais os benefícios, prejuízos e desafios que cada uma das abordagens apresenta. Para isto, durante a fase de planejamento, o processo de desenvolvimento do software foi mapeado e três formatos de apresentar a especificação de requisitos foram selecionados. Na fase de execução do estudo de caso, para cada formato de apresentação foi feita uma medição para coletar dados que e ao final foram confrontados para levantar os resultados. Os resultados foram então comparados e interpretados para concluir qual formato de apresentar a especificação de requisitos

à um time que segue práticas ágeis trouxeram melhores benefícios ao processo do desenvolvimento.

Palavras chave: Engenharia de Requisitos, Metodologias Ágeis, Especificação de Requisitos.

1. Introdução

A engenharia de requisitos e as metodologias ágeis são frequentemente vistas como incompatíveis: engenharia de requisitos é fortemente ligada a documentação para disseminar conhecimento enquanto métodos ágeis são focados em colaborações face a face entre clientes e desenvolvedores para alcançar um objetivo comum (PAETSCH, EBERLEIN e MAURER, 2003).

A engenharia de software considera o processo de criação e desenvolvimento do software, porém, as metodologias ágeis reforçam os aspectos relacionados às constantes mudanças durante o ciclo de desenvolvimento e ao *time-to-market, algo* que traz vantagem competitiva no mercado. Conciliar ambos exige melhoria contínua da qualidade e processo do produto.

Para que seja possível aferir a qualidade e resultados gerados pelo trabalho em conjunto da engenharia de requisitos e métodos ágeis com melhor precisão, realizar uma pesquisa de avaliação através de um estudo de caso e medição sobre uma melhoria de processos faz-se necessária. A melhoria de processos na prática visa entender os processos já existentes e planejar e implementar ações que o modifiquem para melhor atender as necessidades do negócio (FLORAC, PARK e CARLETON, 1997).

Medições são essenciais para a realização de melhorias em processos de software porque fornecem dados objetivos que permitem conhecer o seu desempenho. Os dados coletados para as medidas são a base para a detecção de problemas no desempenho e de inadequações nos processos, bem como para a identificação de oportunidades de melhoria e tomada de decisão (ROCHA, SOUZA e BARCELLOS, 2012).

Para apoiar o planejamento de medição, existem dentro da área de engenharia de software, processos de experimentação que definem métodos e práticas para se realizar desde a formulação de hipóteses e coleta de informações necessárias até a comparação e análise dos resultados, passando por todas as fases para que se possa seguir com o estudo.

Experimentação é o centro do processo científico. Experimentos são a melhor forma de verificar teorias. Somente experimentos podem explorar os fatores críticos e dar luz ao fenômeno novo para que as teorias possam ser reformuladas e corrigidas. Experimentação oferece o modo sistemático, disciplinado, computável e controlado para avaliação da atividade humana (TRAVASSOS, 2002).

Para a condução de experimentos na área de engenharia de software, existem quatro métodos relevantes: científico, de engenharia, experimental e analítico (TRAVASSOS, 2002). A abordagem considerada mais apropriada para área de engenharia de software é a experimental, a qual inicia propondo um novo modelo, baseado ou não em um já existente, e estuda seus efeitos sobre o processo através da medição, análise, avaliação e repetição.

Aplicando a engenharia de software empírica será possível medir os resultados obtidos pelo uso de um conjunto de artefatos de requisitos, analisar e avaliar os resultados, e repetir o processo para outros conjuntos. Pela interpretação dos resultados de observações e análises será possível avaliar esse impacto e seguir utilizando aquele que propicia mais valor de negócio ao processo do desenvolvimento.

1.1 Problema de pesquisa

Processos devem ser tecnicamente corretos e devem ser capazes de atender às necessidades do negócio. Entretanto, processos podem estar corretos do ponto de vista da engenharia de software e não serem competitivos. Podem consumir demasiado tempo e esforço ou não produzirem produtos com a qualidade necessária para satisfazer as necessidades de seus usuários. Processos podem apresentar problemas e devem ser objeto, continuamente, de melhorias. É importante, nesse contexto, dispor-se de mecanismos capazes de evidenciar problemas nos processos e apoiar na identificação de objetivos de melhoria (ROCHA, SOUZA e BARCELLOS, 2012).

Dentro da organização e nos processos que se enquadram no contexto da pesquisa, os fornecedores e consumidores das especificações de requisitos muitas vezes fazem críticas à forma como ela é apresentada e elaborada. Para que se adeque ao processo, aos métodos ágeis, e agrade os responsáveis por elaborá-las e os desenvolvedores que irão consumi-la, desde o momento da concepção até o seu desenvolvimento, a especificação é vista e revisada por diversas vezes, os *feedbacks* nem sempre são positivos e o resultado final pode não ser satisfatório.

1.2 Solução proposta

Desde 1986 Basili já afirmava que as variáveis presentes nos processos de desenvolvimento de software das organizações são diversas, portanto, não existe um modelo de processo padrão que irá atender de forma geral à todas elas. Cada organização tem seus projetos, seus ambientes e times de desenvolvimento. Há também uma variação de fatores em termos de custo e qualidade, experiências,

domínio de problema, metodologias e limitações. Muitas vezes é difícil demonstrar como os programas de melhoria geram valor comercial. Geralmente projetos de software não são capazes de demonstrar explicitamente suas contribuições para metas de nível superior e sucesso dos negócios (BASILI, TRENDOWICZ, *et al.*, 2014).

Pensando nisso, a solução proposta é, através de um estudo de caso, comparar a utilização de diversos conjuntos de artefatos de requisitos, já estabelecidos pela engenharia de requisitos, de forma ágil para guiar o time de desenvolvimento nas direções que prometem atingir melhores resultados. De forma mais específica, estudar e levantar diferentes alternativas de especificar requisitos e através dos processos de experimentos na área de engenharia de software avaliar o que melhor se encaixa nesta pesquisa, a qual foca seu estudo em apenas uma equipe de desenvolvimento dentro de uma organização específica com processos já definidos. Através deste estudo, aplicar estas alternativas de especificação, observar e analisar os impactos sofridos pelo processo para cada tipo diferente aplicado dentro do ciclo de desenvolvimento do software e apresentar os benefícios, prejuízos e desafios encontrados.

1.3 Objetivos

Este trabalho objetiva avaliar o impacto de diferentes formas de especificação de requisitos em um time de desenvolvimento que utiliza práticas ágeis em uma organização específica de desenvolvimento de software por meio de um estudo de caso.

1.3.1 Objetivos específicos

1. Planejar o estudo empírico envolvendo a aplicação de dois formatos distintos para especificação de requisitos;
2. Conduzir o estudo empírico para coleta de dados e sua interpretação;
3. Fornecer uma recomendação à empresa referente ao formato para especificação de requisitos.

1.4 Escopo

O estudo será aplicado em apenas uma organização, no contexto de um processo já implantado e que utiliza práticas ágeis. Assim, não é objetivo definir o processo e sim apenas propor e avaliar alternativas para a especificação de requisitos.

Não é objetivo deste trabalho ser aderente à modelos de qualidade e melhoria de processos como CMMI (CMMI) e MPS.BR (SOFTEX).

A análise do processo atual da equipe selecionada para o estudo realizada durante este projeto, apontou que atualmente nenhum artefato que modele o comportamento do sistema é utilizado. Assim, a partir de entrevistas com os desenvolvedores, restringiu-se a duas alternativas para especificação/medição, sendo elas: caso de uso e diagrama de atividades. Dentre os artefatos de requisitos apresentados por este trabalho, estes dois foram selecionados por representar o comportamento do sistema. Os casos de uso já são comumente utilizados na indústria (COCKBURN, 2001) e que, neste trabalho, foram elaborados utilizando linguagem natural, descartando necessidade de ferramentas específicas porém, podendo

apresentar defeitos como descrições ambíguas (ANDA e SJøBERG, 2002) ou informações incompletas com ausência de regras ou fluxos necessários para entendimento completo do caso de uso (TIWARI e GUPTA, 2015). O diagrama de atividades traz uma abordagem predominantemente gráfica e pode ser útil para mostrar de forma mais direta o comportamento do sistema (GUTIÉRREZ, NEBUT, *et al.*, 2008).

1.5 Justificativa

Diferente dos métodos tradicionais de desenvolvimento de software, os métodos ágeis são marcados por ampla colaboração, como por exemplo, o incentivo à comunicação face a face. Segundo Inayat et al. (2014), embora alega-se ser benéfico, a comunidade de desenvolvimento de software como um todo ainda não está familiarizada com o papel da engenharia de requisitos em métodos ágeis. O termo “engenharia de requisitos ágeis” é usado para definir o “modo ágil” de planejar, executar e pensar sobre as atividades da engenharia de requisitos (INAYAT, SALIM, *et al.*, 2014).

Considerando a falta de familiaridade quanto ao modo ágil de praticar a engenharia de requisitos, propõem-se um estudo de caso para a melhoria do processo de desenvolvimento envolvendo esta área. A qualidade do desenvolvimento de software e produtividade se beneficiam das consequências geradas através das melhorias de processo. Para atingir melhores resultados, o processo deve então sofrer mudanças, e a experimentação oferece o modo sistemático, disciplinado, computável e controlado para se atingir os melhores benefícios e obter os melhores resultados.

Com a adoção das metodologias ágeis, a tendência foi a redução no número de artefatos gerados em detrimento de uma melhor comunicação entre os envolvidos no processo. Entretanto, ainda é desconhecido pelos profissionais de engenharia de requisitos e da área de desenvolvimento de software como um todo como lidar com essas novas práticas e processos de uma forma dinâmica e flexível, como elas solucionam problemas recorrentes gerados pela especificação de requisitos e qual traz melhores benefícios e em quais circunstâncias (INAYAT, SALIM, *et al.*, 2014).

Desta forma, o desenvolvimento deste trabalho, motiva-se pela necessidade das empresas de adequar seus processos de engenharia de requisitos, no que tange as atividades de especificação, em que contexto se encaixam e qual aproximação permite que os princípios ágeis permaneçam claros e em ação e que melhorem a qualidade funcional e estrutural do produto e a qualidade do processo de desenvolvimento.

1.6 Metodologia

O passos para realização deste trabalho são, levantar diferentes alternativas de apresentar a especificação de requisitos e aplicá-las na prática, observar e analisar os impactos sofridos pelos processos para cada tipo diferente aplicado dentro do ciclo de desenvolvimento do software e discutir sobre os resultados obtidos através da análise.

Para tanto, um estudo de caso é desenvolvido dentro de um ambiente real de uma empresa de desenvolvimento de software de forma a demonstrar como a apresentação dos artefatos de especificação de requisitos influenciam o

desenvolvimento de software do ponto de vista de qualidade funcional e estrutural do produto e qualidade do processo de desenvolvimento.

Por ser um estudo dentro de uma empresa real com um time que exerce as atividades de desenvolvimento de software na prática e ser uma avaliação empírica, de acordo com a classificação de métodos de pesquisa (Figura 1), o estudo pode ser considerado uma avaliação de pesquisa, mais especificamente, um estudo de caso industrial.

	R1	R2	R3	R4	R5	R6
Conditions						
Used in practice	T	•	T	F	F	F
Novel solution	•	T	F	•	F	F
Empirical evaluation	T	F	F	T	F	F
Conceptual framework	•	•	•	•	T	F
Opinion about something	F	F	F	F	F	T
Authors' experience	•	•	T	•	F	F
Decisions						
Evaluation research	√	•	•	•	•	•
Solution proposal	•	√	•	•	•	•
Validation research	•	•	•	√	•	•
Philosophical papers	•	•	•	•	√	•
Opinion papers	•	•	•	•	•	√
Experience papers	•	•	√	•	•	•

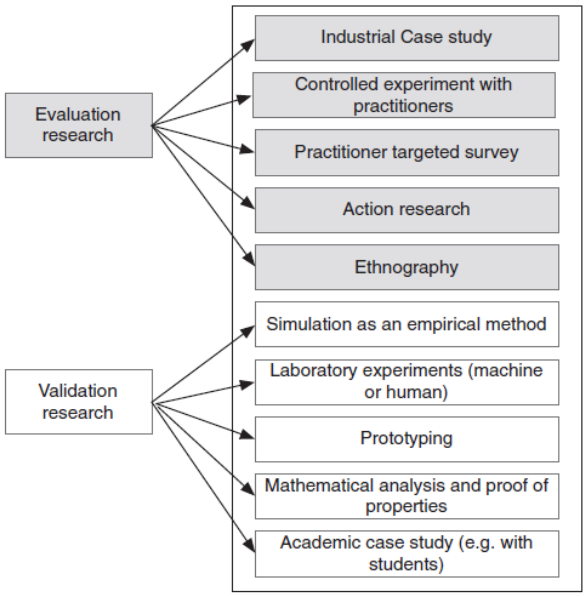


Figura 1 - Classificação de métodos de pesquisa (PETERSEN, VAKKALANKA e KUZNIARZ, 2015)

Este trabalho de conclusão de curso apresenta um estudo da especificação de requisitos, que foca em elicitar, analisar, especificar e avaliar os requisitos de um sistema de informação e o contexto no qual este será usado. Posteriormente, são apresentados os processos de desenvolvimento já utilizados pela organização evidenciando onde se encaixa a engenharia de requisitos e quais processos serão monitorados e analisados, via experimentação, para averiguar os impactos sofridos nestes para cada diferente forma de aplicar a engenharia de requisitos. Ao final, por meio de um estudo de caso, são levantadas as métricas de qualidade de processo de desenvolvimento, e estrutural e funcional do produto, e então é analisada a influência da engenharia de requisitos sob a perspectiva de processos que seguem práticas ágeis. Os passos para a elaboração do trabalho estão detalhados a seguir e o fluxo representado na (Figura 2).

Etapa 1. Fundamentação Teória

a) Estudo através de pesquisa bibliográfica sobre medição e melhoria de processo, como aplicar a experimentação, e modelos e processos para medição;
b) Estudo através de pesquisa bibliográfica sobre a engenharia de requisitos e artefatos utilizados para elaboração de especificações visando identificar formas alternativas para especificar requisitos em um contexto ágil.

Etapa 2. Estudo de caso

a) Modelar o atual processo de desenvolvimento praticado pelo time escolhido para o estudo utilizando a notação BPMN;

b) Por meio de reuniões, comunicar e apresentar ao time de desenvolvimento os artefatos identificados (etapa 1b) para a definição de quais serão aplicados na prática;
c) Levantar os objetivos, elaborar um planejamento e modelo de operação e análise dos resultados para a aplicação do estudo de caso e medição do processo baseando-se no *framework* da experimentação e medição proposto por Basili em *"Experimentation in Software engineering"* (1986) e em *"The Goal Question Metric Approach"* (1994);
d) Iniciar processo de medição utilizando os artefatos de requisitos já utilizados pelo time de desenvolvimento antes do estudo de caso utilizando como guia o modelo de experimentação e medição elaborado na etapa 2c;
e) Repetir o processo de medição para mais 2 diferentes conjuntos de artefatos conforme modelo de experimentação elaborado na etapa 2c;
f) Realizar a análise e interpretação dos resultados do estudo de caso conforme modelo de experimentação e medição elaborado na etapa 2c, e apresentar as conclusões.

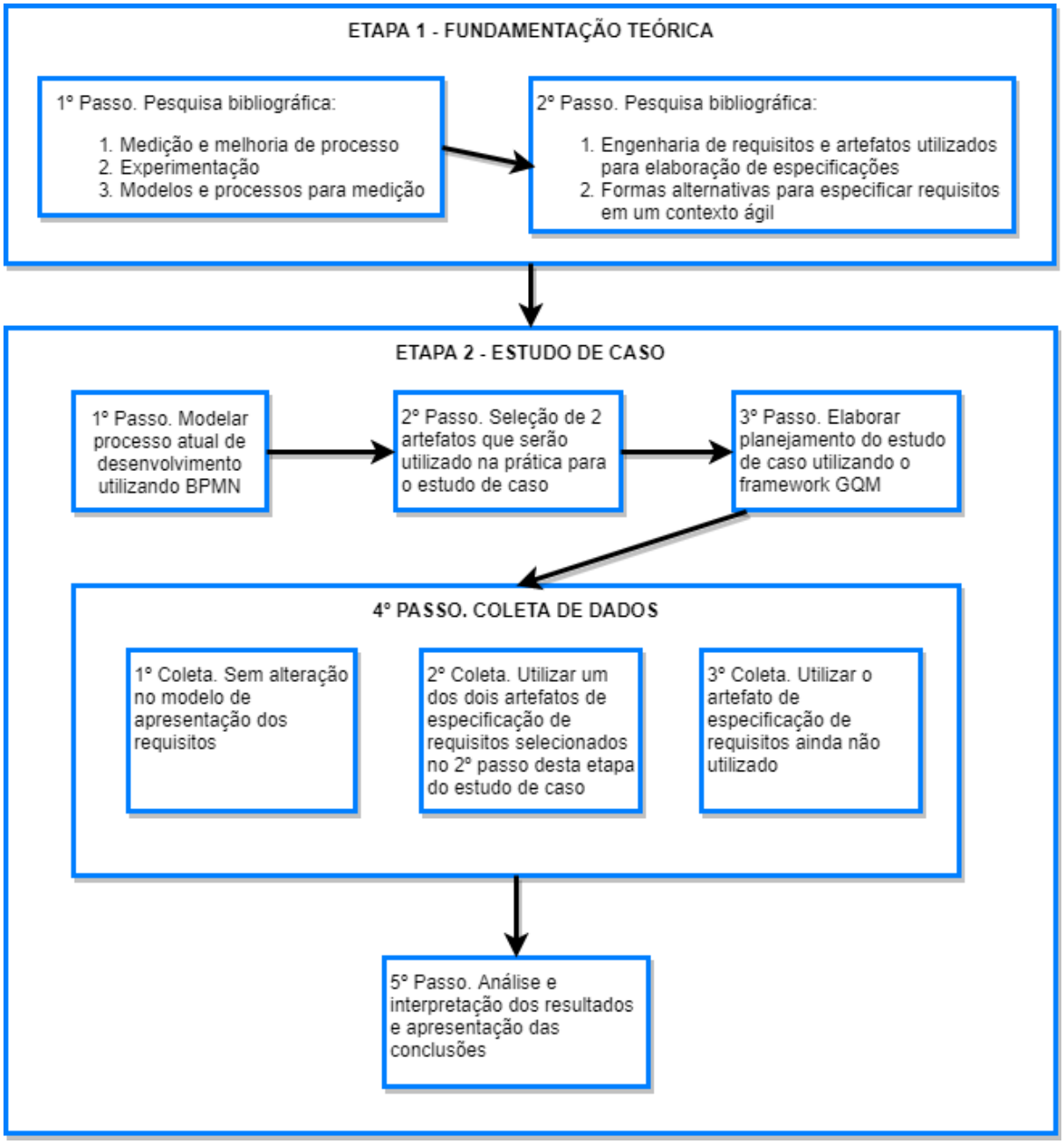


Figura 2 - Metodologia

2. Fundamentação Teórica

Neste capítulo são apresentados alguns conceitos sobre o *framework* de experimentação baseado no aprendizado iterativo e motivações para utilização deste dentro da área de engenharia de requisitos. Também serão apresentadas quais suas etapas e tipos de escopo de projeto.

Em conjunto com o *framework* de experimentação também é abordado o tema “Medição e mehoria de processos” e é apresentado como a medição auxilia a melhorar o nível de maturidade da organização, porque isso se faz necessário, quais métodos já existem formalizados para o processo de medição e o que motivou a utilização do GQM para a execução do estudo de caso.

Após a introdução dos assuntos de experimentação e medição e melhoria de processos, são apresentados alguns conceitos e etapas do método GQM e como é feita a definição do objetivo da medição.

Ao final é introduzido o assunto de engenharia de requisitos onde é apresentado um pouco sobre as atividades exercidas por esta área, como a evolução das práticas ágeis afetou estas atividades e como alguns dos artefatos são gerados e utilizados seguindo estas práticas ágeis.

2.1 Experimentação

A experimentação provê uma base para a necessidade de conhecer e entender os componentes e relacionamentos do desenvolvimento de software e é realizada para melhor avaliar, predizer, entender, controlar e melhorar o processo deste (BASILI, SELBY e HUTCHENS, 1986). O processo de experimentação dentro da área de engenharia de requisitos permite o avanço do conhecimento através de uma

metodologia de aprendizado iterativo. Envolve uma iteração de hipóteses e processos de teste onde modelos de processo ou produto são construídos e hipóteses à seu respeito são testadas e então, a partir das informações geradas com o resultado, o aprendizado é usado para refinar outras hipóteses ou criar novas.

Segundo Basili, Selby e Hutchens (1986), o *framework* de experimentação constitui de 4 fases principais. São elas: definição, planejamento, operação e interpretação (Figura 3).

A primeira fase do *framework* é sub dividido em 5 partes: motivação, objeto, propósito, perspectiva, domínio e escopo. É possível que o estudo possa ter mais de 1 item em cada parte. Por exemplo, a motivação indica o porquê se está fazendo o estudo. Pode ser entender, aplicar ou melhorar um processo do desenvolvimento do produto ou entender as consequências do uso de uma tecnologia diferente. O objeto é a entidade principal do estudo. O propósito é o objetivo que se pretende realizar com o estudo, pode ser, por exemplo, caracterizar a melhoria e redução de custo de desenvolvimento do produto a partir da eficácia do processo de testes. A perspectiva deve indicar quais os pontos de vista devem ser considerados pelo estudo, se de um desenvolvedor, um usuário, ambos ou outras. O domínio se divide em dois grupos importantes, um desenvolvedor (ou um time de desenvolvimento) ou o programa/projeto que é desenvolvido no contexto de aplicação da experimentação. A classificação pode ser obtida examinando o tamanho dos domínios. O escopo pode contextualizar 4 diferentes grupos de projetos separados em 2 tipos de domínios (Figura 4). *Blocked subject-project* aplica um estudo sobre um ou mais objetos entre um conjunto de times e um conjunto de projetos. *Replicated project* aplicam o estudo sobre um ou mais objetos entre times do mesmo projeto, enquanto *multi-project*

variations faz o mesmo para apenas um time que trabalha em diferentes projetos. Uma experimentação com escopo definido como *Single project* faz um estudo sobre um ou mais objetos em um único time de apenas um projeto. Quanto maior o escopo e maior a representatividade das amostras coletadas pela experimentação, melhores são as chances de a conclusão representar uma maior parcela de projetos e times de organizações existentes no mundo.

A segunda fase do *framework* de experimentação tem por objetivo o planejamento do estudo. Nessa fase é feita uma conexão do escopo do estudo com métodos analíticos e indicação das amostras do domínio que serão examinadas, focando o experimento somente naquilo que deve de fato ser estudado para guiar o processo na direção correta. Os critérios do estudo serão moldados por aquilo que foi definido na primeira fase da experimentação e a manifestação destes critérios serão capturados através do processo de medição desta fase, que planeja os dados que serão medidos e coletados pelo estudo.

A terceira fase do processo de experimentação é a fase de operação do estudo. Para calibrar e preparar o experimento, um estudo piloto pode ser aplicado, ajudando a validar o cenário e fatores do estudo. Os experimentadores coletam e validam os dados definidos na fase de planejamento, durante a fase de operação.

I. Definition					
Motivation	Object	Purpose	Perspective	Domain	Scope
Understand Assess Manage Engineer Learn Improve Validate Assure	Product Process Model Metric Theory	Characterize Evaluate Predict Motivate	Developer Modifier Maintainer Project manager Corporate manager Customer User Researcher	Programmer Program/project	Single project Multi-project Replicated project Blocked subject-project

II. Planning		
Design	Criteria	Measurement
Experimental designs Incomplete block Completely randomized Randomized block Fractional factorial Multivariate analysis Correlation Factor analysis Regression Statistical models Non-parametric Sampling	Direct reflections of cost/quality Cost Errors Changes Reliability Correctness Indirect reflections of cost/quality Data coupling Information visibility Programmer comprehension Execution coverage Size Complexity	Metric definition Goal-question-metric Factor-criteria-metric Metric validation Data collection Automatability Form design and test Objective vs. subjective Level of measurement Nominal/classificatory Ordinal/ranking Interval Ratio

III. Operation		
Preparation	Execution	Analysis
Pilot study	Data collection Data validation	Quantitative vs. qualitative Preliminary data analysis Plots and histograms Model assumptions Primary data analysis Model application

IV. Interpretation		
Interpretation context	Extrapolation	Impact
Statistical framework Study purpose Field of research	Sample representativeness	Visibility Replication Application

Figura 3 - Resumo do framework de experimentação (BASILI, SELBY e HUTCHENS, 1986).

#Teams per project	#Projects	
	one	more than one
one	Single project	Multi-project variation
more than one	Replicated project	Blocked subject-project

Fig. 3. Experimental scopes.

Figura 4 - Tipos de escopo na fase de definição da experimentação (BASILI, SELBY e HUTCHENS, 1986).

A quarta e última fase do *framework* de experimentação é a fase de interpretação que consiste em interpretar o contexto, extrapolar e avaliar o impacto. O contexto de interpretação define a base estatística na qual os resultados derivam, o propósito particular do estudo e a área de pesquisa, ou seja, o cenário em que está sendo aplicado o estudo e como deve-se interpretar os dados. As amostras coletadas e a representatividade dos cenários dentro de um contexto generalizado definem como os resultados podem ser extrapolados para outros ambientes. Outras atividades que podem contribuir para o estudo é publicar os resultados para receber *feedbacks*, replicar o experimento e aplicar de fato as modificações estudadas baseando-se nas conclusões da experimentação.

Um estudo aplicado por Basili sobre múltiplas publicações à respeito dos quatro tipos genéricos de escopo de experimentação concluiu que aparentemente não existe *"modelo universal"* ou *"bala de prata"* na engenharia de software. Existe uma grande variação de fatores que diferem entre diversos ambientes, em termos objetivo de custo/qualidade desejados, metodologias, experiências, domínio de problemas, limitações, etc (BASILI, SELBY e HUTCHENS, 1986). Há diversas categorias de experimentos, feitos para diversos propósitos que tipicamente buscam a relação entre duas variáveis como a relação entre características do processo e características do produto, desde experimentos que lidam com um grande número de variáveis à problemas de baixo nível e pequeno número de variáveis (BASILI, 2011).

2.2 Medição e Melhoria de Processos

Para atingir melhores níveis de maturidade, os processos da organização sofrem constantes melhorias. Nem sempre os processos bem definidos e enraizados

na cultura da organização são competitivos. Pode ser devido ao excesso burocracia, planejamentos extensos ou até mesmo o inverso, a falta dele. Para que sejam competitivos, agreguem valor ao negócio da organização e melhorem a qualidade do produto, os processos devem obter uma melhoria no seu desempenho, melhor se adequar às necessidades do negócio e da organização ou almejar níveis mais altos de maturidade. De acordo com Souza, Barcellos e Rocha (2012), estas melhorias devem ser bem planejadas e com objetivos bem definidos, específicos e o mais importante, devem ser mensuráveis.

Segundo Florac e Carleton (1999), essencialmente, os objetivos devem ser mensuráveis, pois é através da medição que se obtém dados do desempenho da melhoria. Com coleta, medição, análise e interpretação dos dados é possível predizer custos e desempenhos, construir *benchmarks* para avaliar tendências, estabilidade e capacidade dos processos, detectar problemas, inadequações, oportunidades de melhoria e auxílio à tomada de decisões.

O processo de medição pode, portanto, ser definido como um conjunto de passos que orienta a realização da medição em uma organização. Um processo de medição eficiente é fator crítico de sucesso do programa de medição, pois é ele que direciona as atividades a serem realizadas para que com os resultados da análise dos dados coletados seja possível a identificação de tendências e antecipação aos problemas (WANG, 2005)

Para execução do processo de medição, é exigido o apoio de um processo capaz de garantir o cumprimento disciplinado das tarefas, sendo que de acordo com Travassos (2002), a experimentação oferece o modo sistemático, disciplinado, computável e controlado para a avaliação da atividade humana.

Para se obter sucesso no processo de medição, este precisa ser bem planejado e corretamente alinhado com o objetivos estratégicos da empresa. Uma falha na fase de planejamento da medição pode acarretar em medidas superficiais e que não atendem corretamente à necessidade de informação para atingir os objetivos. Para apoiar o planejamento do processo de medição, existem dois métodos que se destacam: GQM (*Goal-Question-Metric*) e PSM (*Practical Software Measurement*).

Segundo Souza, Barcellos e Rocha (2012), o método GQM é bastante utilizado para apoio à definição de medidas devido à sua simplicidade de uso e à estrutura *top down* que possibilita a identificação das medidas a partir de questões associadas aos objetivos de medição. Já o PSM é um método com um modelo de informação para medição padrão que associa os objetivos almejados com as informações necessárias para medi-los. Possui também um modelo de processo de medição.

Por ser um estudo de pequeno porte, onde apenas uma equipe de desenvolvimento de apenas um projeto de uma organização serão analisados, optou-se por utilizar o método GQM por ser o mais simples de utilizar.

2.2.1 GQM

Segundo Basili, Caldiera e Rombach (2001), de acordo com muitos estudos sobre aplicação de métricas e modelos em ambientes industriais, para que a medição seja efetiva é necessário que esta seja: focada em objetivos específicos, aplicada em todo o ciclo de vida do produto, processo e recursos e interpretada baseado na caracterização e entendimento do contexto da organização, ambiente e objetivos. Isso significa que a medição precisa ser planejada com uma perspectiva *top-down*

para que esteja de acordo com modelos e objetivos definidos pelo contexto da organização, visto que se feito da maneira inversa, *bottom-up*, a quantidade de variáveis e métricas a se observar é grande, e a forma de interpretá-las não ficará clara e visível sem um modelo com objetivos definidos. Já a sua execução, no processo de análise e interpretação dos dados coletados, a perspectiva deve ser *bottom-up* pois as métricas foram definidas focadas em um objetivo e as informações fornecidas através das métricas deve ser interpretadas e analisadas respeitando-os (Figura 5).

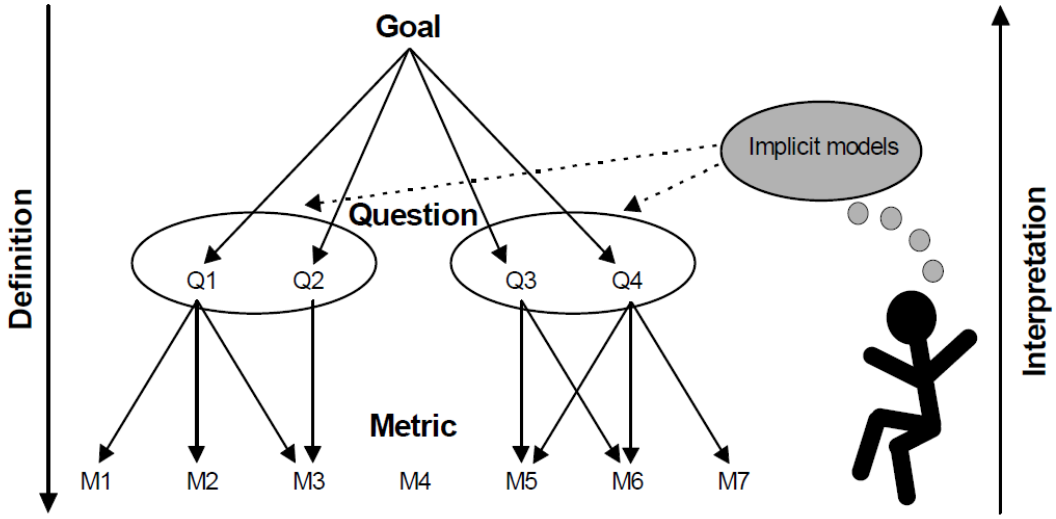


Figura 5 - The GQM Paradigm (BASILI, CALDIERA e ROMBACH, 2001).

O método GQM é composo por quatro fases (Figura 6): planejamento, definição, coleta de dados e interpretação (SOLINGEN e BERGHOUT, 1999). Na fase de planejamento contempla-se todo tipo de medida que deve ser tomada para que seja possível realizar a medição, definindo as pessoas envolvidas, treinamento necessários e o planejamento do projeto onde será feito o estudo. A fase de definição é reservada à elaboração do modelo com objetivos, questões e métricas definidas. Durante a fase de coleta de dados são definidos os formulários para a coleta dos

dados que são preenchidos com os valores medidos e armazenados para posterior interpretação. Por fim, a fase de interpretação é utilizada para que os valores da medição sejam interpretados para responder às questões levantadas e afirmar se os objetivos foram de fato alcançados.

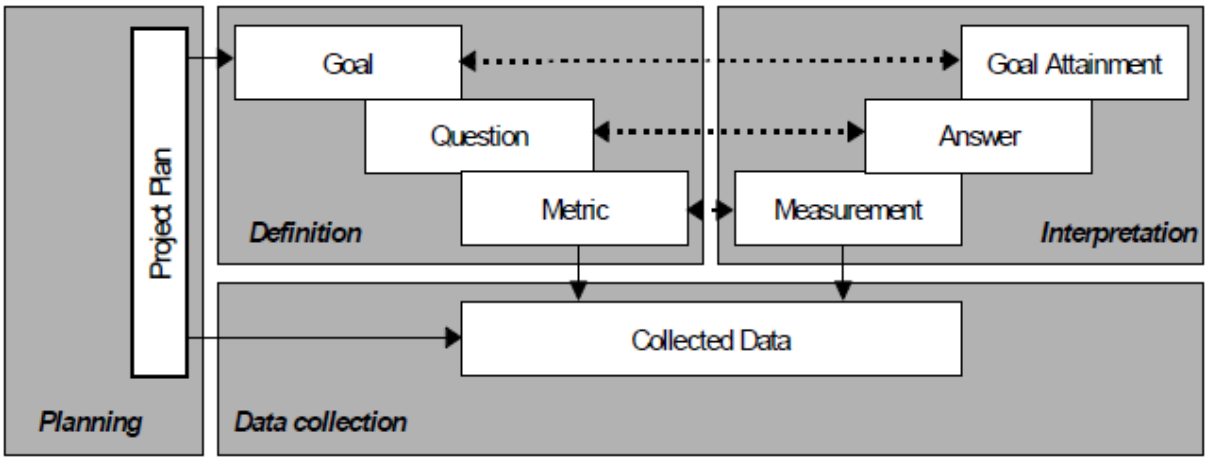


Figura 6 - As quatro fases do GQM (SOLINGEN e BERGHOUT, 1999).

A saída do processo de aplicação do GQM é uma especificação de um sistema de medições voltado para assuntos particulares e um conjunto de regras de interpretação dos dados obtidos. O resultado do modelo de medição deve conter 3 níveis: conceitual (*goal*), operacional (*question*), quantitativo (*metric*).

O nível conceitual define objetivos, os quais são definidos para um objeto de estudo relativo a um ecossistema em particular. De acordo com Basili, Caldiera e Rombach (2001), objetos de medição são:

- Produtos: artefatos, entregáveis e documentos que são produzidos durante o ciclo de vida do sistema; por exemplo, a especificação de requisitos, designs, programas, suíte de testes;
- Processos: atividades relativas ao software normalmente associadas à tempo; por exemplo, especificar, desenhar, testar, entrevistar;

- Recursos: itens utilizados por processos para que seja gerada um saída; por exemplo, pessoas, hardware, software, espaço físico.

O nível operacional define um conjunto de questões que será utilizado para caracterizar como a avaliação ou realização dos objetivos será alcançada. As questões tentam caracterizar o objeto de medição relativo à uma demanda de qualidade específica e determinar a sua qualidade por esse ponto de vista.

O nível quantitativo define um conjunto de dados que serão coletados e associados à cada questão correspondente para que possa respondê-la de forma quantitativa. Segundo Basili, Caldiera e Rombach (1994) os dados podem ser:

- Objetivos: se dependem apenas do objeto de estudo e não dependem do ponto de vista em que estão sendo coletados; por exemplo, o número de versões de um determinado tipo de documento, horas utilizadas para implementar uma tarefa específica.
- Subjetivos: se dependem tanto do objeto do estudo quanto do ponto de vista de onde são coletados; por exemplo, legibilidade de um texto, nível de satisfação do usuário.

Um modelo GQM é uma estrutura hierárquica (Figura 7) que se inicia com um objetivo de medição, o qual especifica propósitos, objeto de medição, assuntos a serem medidos e pontos de vista dos quais a medição será feita. O refinamento deste objetivo deve então gerar as questões, que especificam os assuntos em perguntas, as quais serão respondidas por mais uma etapa de refinamento que geram as métricas, que podem ser objetivas ou subjetivas. Algumas métricas podem ser utilizadas para responder as mesmas perguntas sobre o mesmo objetivo, inclusive,

modelos de GQM podem conter questões e métricas em comum, desde que seja levado em consideração o ponto de vista de onde são coletados.

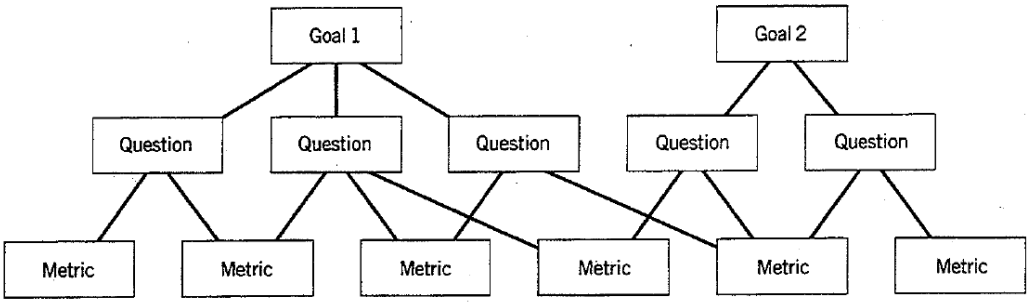


Figura 7 - Estrutura de modelo hierárquico do GQM (BASILI, CALDIERA e ROMBACH, 2001).

A definição do objetivo é uma tarefa essencial para o processo de medição pela aplicação do GQM e ela é apoiada por uma metodologia passo a passo. Em conformidade com Basili, Caldiera e Rombach (2001) um objetivo possui três coordenadas: o assunto, o objeto e um ponto de vista, e um propósito (Figura 8).

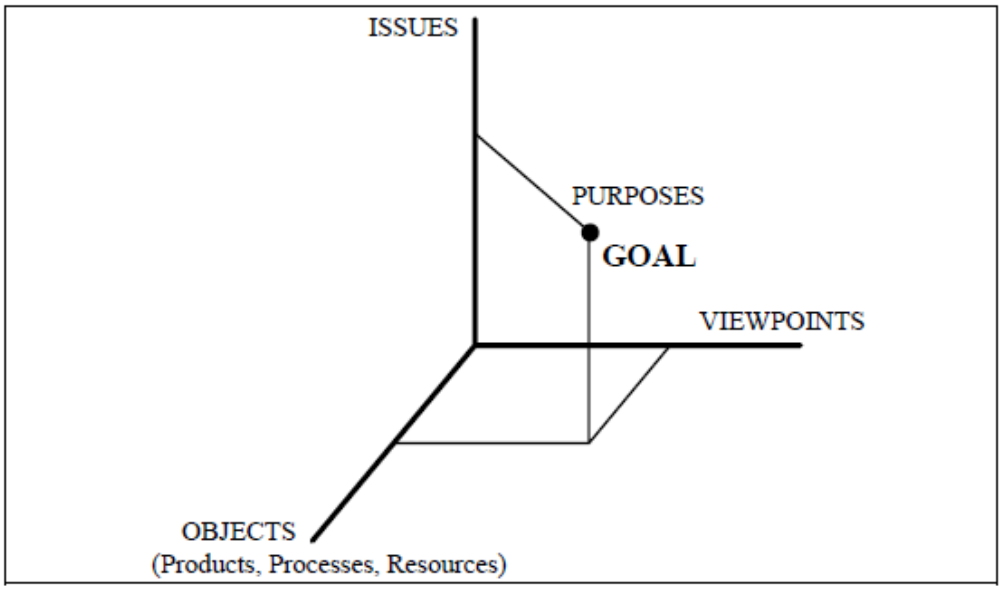


Figura 8 - Coordenadas do objetivo (GQM).

O processo de definição de objetivos é crítico para a efetividade da aplicação do processo GQM. Para essa fase de definição, existe uma metodologia passo-a-passo, seguindo 3 as coordenadas mostradas na Figura 8.

1. Assunto: Tempo de duração

2. Objeto: Processo de solicitação de alteração
3. Ponto de vista: Gerente de projeto

E o propósito:

- Propósito: Melhorar

O assunto e propósito do objetivo são derivados das políticas e estratégias da organização, que podem ser obtidos através da análise de declarações das políticas corporativas, planos estratégicos ou entrevistas com os indivíduos envolvidos com o objeto de estudo. Para definir a coordenada “Objeto” do objetivo, uma fonte de informação são as descrições dos processos que se enquadram no escopo da medição, ou seja, se a avaliação será feita sobre um processo, utiliza-se o modelo do processo e seus sub processos. A partir desta tarefa, com os objetivos formalizados, é possível refiná-los em questões para que sejam quantificados posteriormente através das métricas.

De acordo com Basili (2001), são levantados 3 grupos de questionamentos (tabela 1):

Tabela 1 - Grupos de questões (GQM).

Grupo 1.	Como caracterizar o objeto de acordo com o objetivo?
Exemplos:	Qual a velocidade atual do processo?
	O processo de solicitação de alteração é de fato executado?
Grupo 2.	Como caracterizar os atributos relevantes do objeto de acordo com o assunto?
Exemplos:	Qual a diferença entre o tempo real e o tempo estimado da execução do processo de solicitação de alteração?
	A performance do processo está melhorado?
Grupo 3.	Como avaliar as características relevantes do objeto de acordo com o assunto?

Exemplos:	A atual performance do processo é satisfatória do ponto de vista do gerente do projeto?
	A visibilidade da performance do processo está melhorando?

Fonte tabela 1 - adaptado de (BASILI, CALDIERA e ROMBACH, 2001)

Após o levantamento das questões, inicia-se o procedimento de associá-las à métricas, e para isso alguns fatores devem ser considerados. Um deles é a quantidade e qualidade dos dados já existentes, pois deve ser levado em consideração todas as informações já coletadas que estão disponíveis e são confiáveis. A maturidade do objeto de medição é outro fator, para objetos com nível de maturidade mais alto, pode-se aplicar medidas objetivas, e quanto mais baixo a maturidade, menos formais ou mais instáveis são os objetos de medição, portanto mais subjetivas devem ser as medidas de avaliação associadas à ele. Assim que o modelo de GQM estiver desenvolvido, seleciona-se as técnicas, ferramentas e procedimentos apropriados para a coleta e interpretação dos dados.

2.3 Engenharia de Requisitos

A engenharia de requisitos é uma abordagem sistemática e disciplinada para a especificação e gerenciamento de requisitos (POHL e RUPP, 2012). Durante o processo de desenvolvimento, a engenharia de requisitos deve elicitar os requisitos dos *stakeholders*, documentar os requisitos de forma adequada, validar e verificar os requisitos, e gerenciar os requisitos ao longo de todo o ciclo de vida do sistema (POHL, 1996).

De acordo com Pohl e Rupp (2012), os objetivos que norteiam a engenharia de requisitos são: conhecer os requisitos relevantes, estabelecer um consenso entre os *stakeholders* a respeito de tais requisitos, documentar os requisitos de acordo com

determinados padrões e gerenciar os requisitos de forma sistemática e; compreender e documentar as expectativas e necessidades dos *stakeholders*, especificar e gerenciar os requisitos para minimizar o risco de entregar um sistema que não atenda às suas expectativas e necessidades. Existem quatro atividades principais nos processos da engenharia de requisitos (Figura 9): estudo de viabilidade, elicitação e análise de requisitos, especificação de requisitos e validação de requisitos (SOMMERVILLE, 2010).

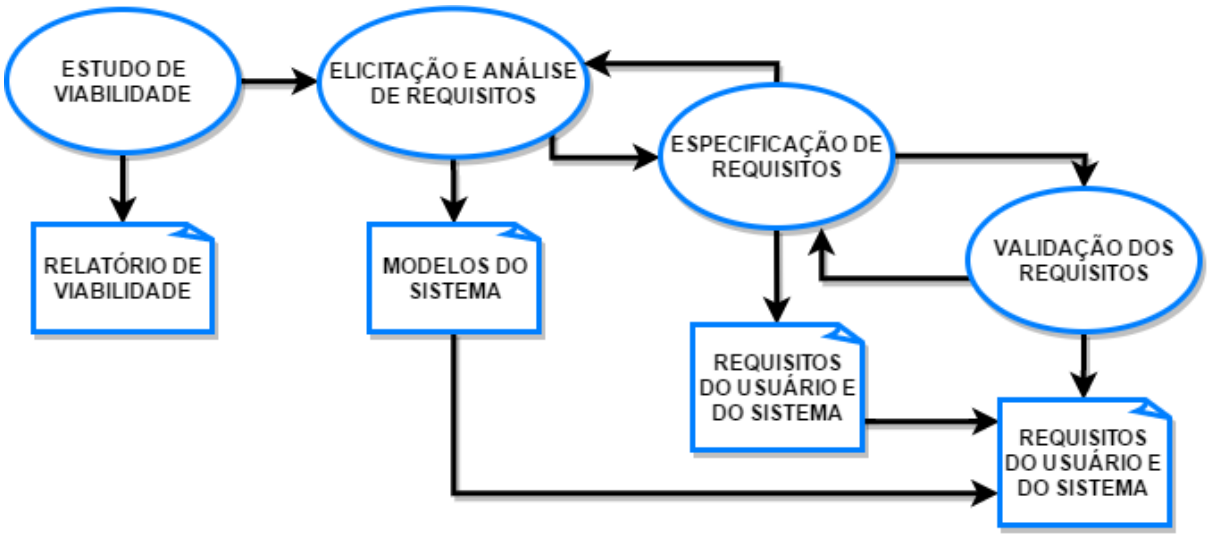


Figura 9 - Os processos da Engenharia de Requisitos (SOMMERVILLE, 2010)

O estudo de viabilidade é feito para gerar uma estimativa de que as necessidades do usuário podem ser satisfeitas utilizando as tecnologias de software e hardware disponíveis. O estudo também considera se o produto final será rentável do ponto de vista comercial e se o orçamento disponível será suficiente. O estudo deve ser relativamente barato e rápido e deve concluir se o projeto deve ou não seguir em frente.

A atividade de elicitação envolve obter os requisitos para o sistema a ser desenvolvido através de técnicas e fontes de requisitos. Este é o processo de derivação dos requisitos do sistema através da observação de sistemas existentes,

discussões com potenciais usuários e compradores, análise de perguntas e assim por diante. Isso pode envolver o desenvolvimento de um ou mais modelos de sistema e protótipos. Isso ajuda você a entender o sistema a ser especificado. Durante a atividade de especificação os requisitos elicitados são descritos de forma mais adequada utilizando linguagens naturais ou modelos conceituais.

A especificação dos requisitos é a atividade de traduzir as informações coletadas durante a atividade de análise em um documento que define um conjunto de requisitos. Para garantir que os critérios de qualidade previamente definidos sejam atingidos, os requisitos documentados devem ser validados e negociados desde o princípio.

A Figura 9 mostra as atividades da engenharia de requisitos como atividades sequenciais no entanto, na prática, a engenharia de requisitos é um processo iterativo no qual as atividades são intercaladas (Figura 10). As atividades são organizadas como um processo iterativo em torno de uma espiral, sendo a saída um documento de requisitos do sistema. A quantidade de tempo e esforço dedicado a cada atividade em cada iteração depende do estágio do processo geral e do tipo de sistema que está sendo desenvolvido. No início do processo, o maior esforço será gasto na compreensão dos requisitos comerciais e não funcionais de alto nível e dos requisitos do usuário para o sistema. Mais tarde no processo, nos anéis externos da espiral, mais esforço será dedicado a induzir e entender os requisitos detalhados do sistema. O desenvolvimento ágil pode ser usado em vez de prototipagem para que os requisitos e a implementação do sistema sejam desenvolvidos em conjunto.

Em praticamente todos os sistemas, os requisitos mudam. As pessoas envolvidas desenvolvem uma melhor compreensão do que eles querem que o

software faça; A organização que compra o sistema muda; Modificações são feitas no hardware, software e ambiente organizacional do sistema. O processo de gerenciamento desses requisitos em mudança é chamado de gerenciamento de requisitos (SOMMERVILLE, 2010).

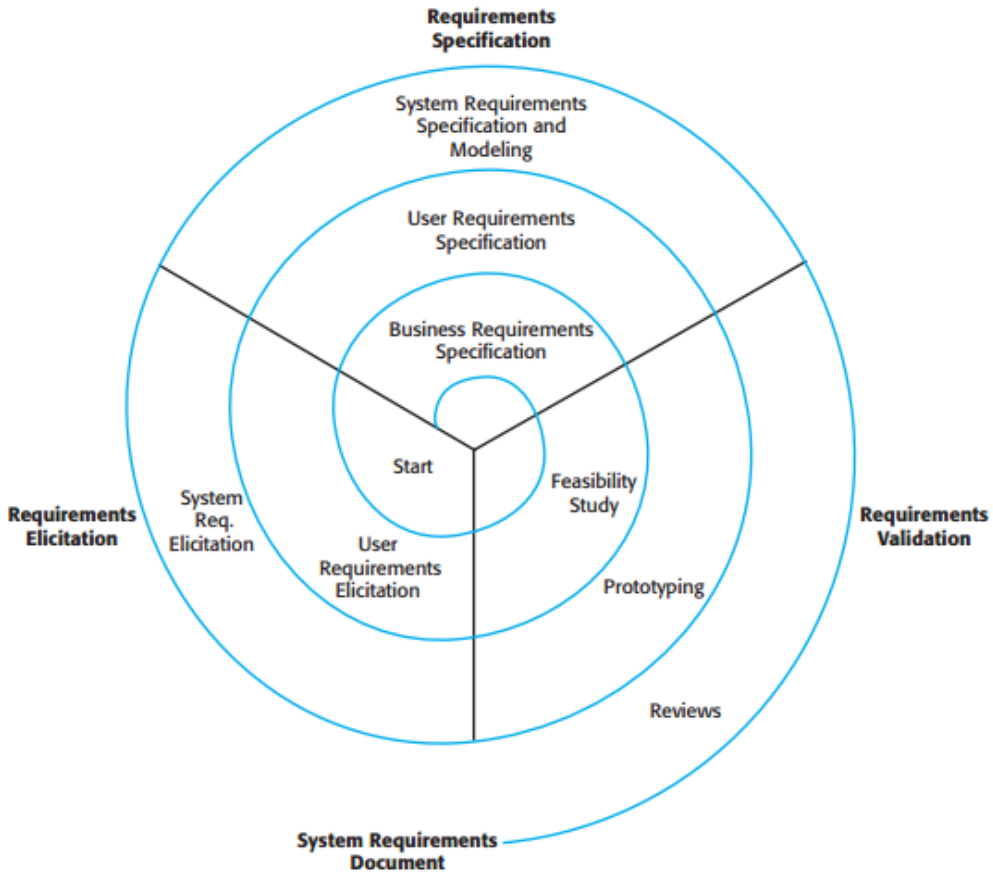


Figura 10 - Uma visão espiral dos processos da engenharia de requisitos (SOMMERVILLE, 2010)

Para o estudo de caso deste trabalho, será alterado o produto da atividade de especificação de requisitos, produzindo artefatos diferentes para cada iteração do time e realizando uma comparação de execução do processo de desenvolvimento em cada uma delas.

2.3.1 Artefatos de Requisitos sob Práticas Ágeis

Por mais de 40 anos softwares tem sido implementado utilizando diversas metodologias, processos, estrutura, gestão e controle do trabalho (Figura 11). Com o tempo, o desenvolvimento de software começou a alcançar muitas áreas, e rapidamente já estavam sendo utilizados para desenvolver desde simples ferramentas à controle de vôos comerciais e com isso, os riscos de custos econômicos e humanos cresceram exponencialmente. Chegou a um ponto onde as aplicações sendo desenvolvidas estavam cada vez maiores, e as metodologias estavam ficando cada vez mais carregadas com processos e ferramentas de controle, e com isso a habilidade de entregar valor e software de qualidade foi decaindo. Desde o final da década de 90 as metodologias de desenvolvimento de software “agile” e “lean”, que incluem tratamentos dos requisitos de software mais leves, porém seguros, tem sido um dos fatores mais importantes afetando essa indústria (LEFFINGWELL, 2011).

Um dos primeiros modelos utilizados foi o modelo cascata, criado na década de 1970, onde o desenvolvimento seguia uma série sequencial de estágios e se trabalhava com orçamento, cronograma e escopo fechado. Um estudo feito por Thomas (2001) chegou a conclusão que a grande causa deste modelo ter falhado foi o escopo de requisitos fixado. Era preciso um modelo que tivesse mais interação com stakehoders e flexível à mudanças.

Nas décadas seguintes surgiram os modelos de processos iterativos (Spiral, RAD, RUP, etc) que intencionalmente se distanciaram do modelo tradicional cascata onde existia uma fase inicial para especificar todo os escopo do projeto, e assumiram

um processo baseado na “descoberta”, com documentação inicial menos carregada, com mais definições de “o que” deve ser desenvolvido, e menos o “como”.

No final da década de 90 até os dias de hoje houve uma grande quantia de modelos adaptativos sendo apresentados à indústria de desenvolvimento de software. Esses modelos assumem que com as ferramentas e práticas de desenvolvimento certas, é mais financeiramente eficiente escrever código, tê-lo avaliado pelo usuário, e se incorreto, refatorá-lo rapidamente, do que antecipar toda a documentação de requisitos previamente.

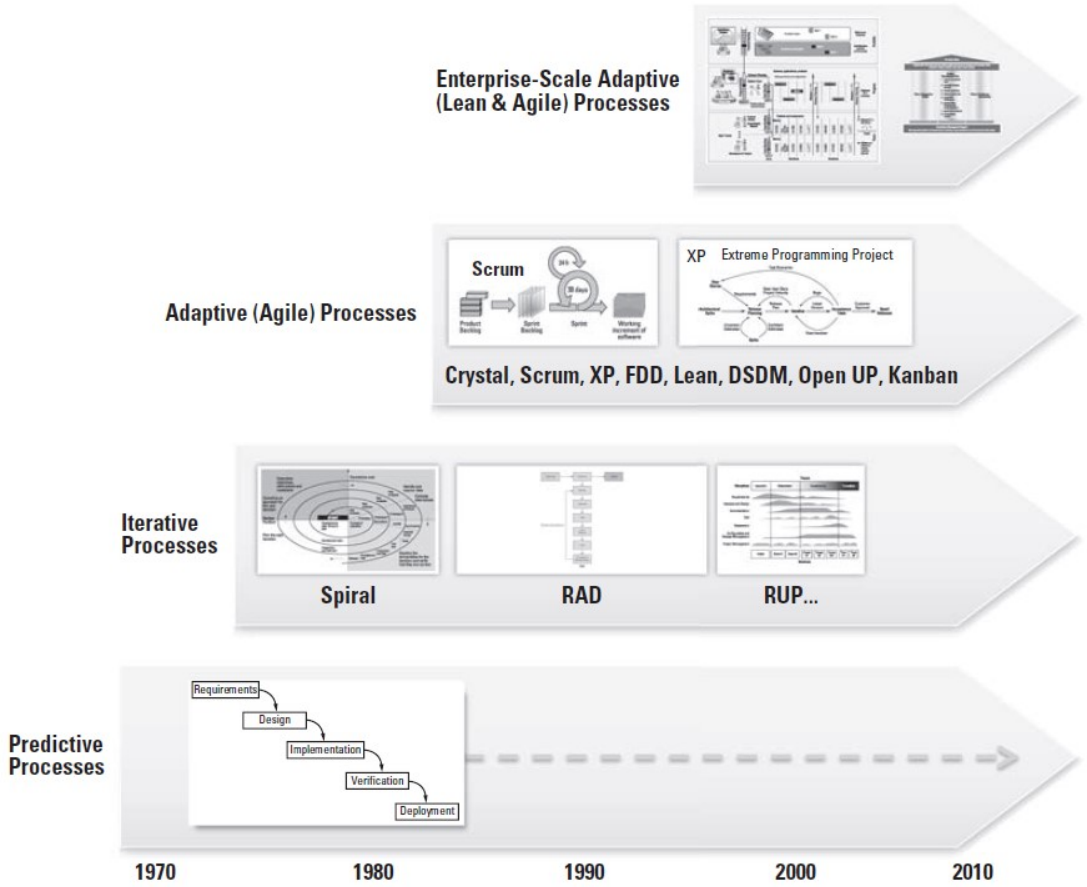


Figura 11 - Movimento de processo de software das últimas décadas (LEFFINGWELL, 2011).

Então, com o ágil, a gestão de requisitos de software tem uma aproximação muito mais flexível, iterativa, e feita no momento certo (LEFFINGWELL, 2011). Com o ágil, é deixado de lado os métodos tradicionais de especificar requisitos de software, e junto, o compromisso de entregar tudo ao mesmo tempo, com orçamentos e cronogramas fixos.

Um time que pratica os princípios ágeis deve aplicar um modelo de requisitos o mais simples possível porém, que dê suporte à todos os stakeholders e necessidades dos membros do time e que construa e compartilhe uma visão única do produto. Times ágeis tem como artefato de requisito central de desenvolvimento a história de usuário, pois é ela que contém e carrega todos os dados que devem entregar valor de negócio ao usuário (LEFFINGWELL, 2011) entretanto, ela também introduz algumas dificuldades ao time. Cockburn (2001) afirma que somente histórias de usuário não fornece ao desenvolvedor um contexto para se trabalhar. Um time de desenvolvimento estima um projeto em pontos de história mas, assim que iniciam o trabalho, essa pontuação só tende a aumentar (COCKBURN, 2001). Histórias de usuário não fornecem uma visão do trabalho que está por vir.

Para a seleção dos possíveis artefatos para se trabalhar sob os conceitos de práticas ágeis, que visa evitar ao máximo o desperdício de trabalho e recurso, uma lista pré-selecionada por Ambler (2001) foi coletada em agilemodeling.com e apresentada ao time de desenvolvimento para definição e escolha de quais seriam utilizados para o estudo de caso. Os seguintes artefatos foram apresentados à equipe:

- 2.3.1.1 Testes de aceitação;
- 2.3.1.2 Regras de Negócio;

- 2.3.1.3 Change Case;
- 2.3.1.4 CRC Model;
- 2.3.1.5 Constraint Definition;
- 2.3.1.6 Diagrama de Fluxo de Dados;
- 2.3.1.7 Use Case;
- 2.3.1.8 Protótipos de Interface (alta de baixa fidelidade);
- 2.3.1.9 User Interface Flow Diagram;
- 2.3.1.10 Feature;
- 2.3.1.11 Requisitos Técnicos;
- 2.3.1.12 Diagrama de Caso de Uso;
- 2.3.1.13 Diagrama de Atividades;
- 2.3.1.14 User Story.

2.3.1.1 Testes de aceitação

Descrevem requisitos de *caixa preta que* são identificados pelos stakeholders, e que o sistema deve estar de acordo (AMBLER, 2001). Tipicamente são identificados como artefatos de testes porém, é através dos critérios descritos nestes que se torna possível determinar se o sistema está de acordo com a necessidade descritas pelos stakeholders. Regras de negócio, funcionalidades e requisitos técnicos não funcionais podem ser encapsulados em testes de aceitação.

A tabela 2 mostra um modelo de um teste de aceitação que normalmente inclui uma descrição do que está sendo testado, instruções de como executar o teste e quais os resultas esperados. Opcionalmente pode também incluir um identificador (ID) e uma configuração inicial (*setup*).

Tabela 2 – Exemplo de especificação de teste de aceitação.

ID	T001
Descrição	Contas correntes tem um limite de cheque especial de R$500. Enquanto o saldo após o saque for suficiente (maior que -R$500), estes devem ser permitidos.
Configuração inicial (setup)	1. Criar uma conta #12345 com saldo inicial de R$50; 2. Criar uma conta #67890 com saldo inicial de R$0
Instruções	1. Sacar R$200 da conta #12345 2. Sacar R$350 da conta #67890 3. Depositar R$100 na conta #12345 4. Sacar R$200 da conta #67890 5. Sacar R$150 da conta #67890 6. Sacar R$200 da conta #12345 7. Depositar R$50 na conta #67890 8. Sacar R$100 da conta #67890
Resultados esperados	Conta #12345: • Balanço final = -R$250 • Um log de saque de R$200 • Um log depósito de R$100 • Outro log saque de R$200 Conta #67890: • Balanço final = -R$500 • Um log de saque de R$350 • Outro log saque de R$150 • Um log de depósito de R$50 Log de erros: • Saldo insuficiente na conta #67890 (saldo de -R$350) para o saque de R$200 • Saldo insuficiente para conta #67890 (saldo de -R$450) para saque de R$100

Fonte tabela 2 - (AMBLER, 2001).

2.3.1.2 Regras de Negócio

De maneira geral, regras de negócio são limitações: elas definem as condições que devem ser verdadeiras em situações específicas (MORGAN, 2002). Regras de negócio não são descrições de processos, são as condições sob quais os processos são realizados ou são condições que devem existir após o processo ser completado.

Uma regra de negócio define ou limita um aspecto de negócio com a intenção de estruturar ou influenciar o comportamento do sistema. As regras de negócio costumam especificar aspectos relacionados à controle de acessos, cálculos, ou políticas do assunto tratado pelo sistema.

Quando a descrição é concisa e foca em apenas um conceito de negócio, a regra pode ser considerada bem descrita, e dessa forma ela pode ser reutilizada, sendo referenciada por outro artefato da especificação.

Uma regra de negócio pode ser descrita de uma forma muito simples e dependendo da ocasião pode ser apenas uma frase. Uma regra de negócio deve ter um identificador, um nome e uma descrição. Opcionalmente é possível colocar exemplos, fontes de informação e o identificador de outras regras relacionadas para facilitar o entendimento. A tabela 3 demonstra um exemplo de regra de negócio para um sistema de gestão de uma universidade.

Segundo Ross (2003), existem alguns princípios que uma regra de negócio deve seguir:

- Ser escrita e explícita;
- Ser expressa em linguagem simples;
- Existir independente de procedimento ou fluxo de trabalho;
- Construída em fatos;
- Guiar ou influenciar o comportamento da maneira desejada;
- Ser motivada por fatores de negócio identificáveis e importantes;
- Ser acessível por pessoas autorizadas;
- Ser a única fonte;
- Ser especificada por pessoas que detém o conhecimento;

- Ser gerenciada

Tabela 3 - Exemplo de especificação de regra de negócio.

Nome	Professores titulares podem administrar notas de alunos.
Identificador	RN123.
Descrição	Somente professores titulares tem permissão para incluir, modificar e alterar notas de alunos referentes à provas e seminários que eles ministram.
Exemplos:	Dr. João, professor titular da matéria “Matemática A” pode gerenciar a nota de todos os estudantes matriculados nesta matéria enquanto estiver ativa ,porém não pode fazer o mesmo para os alunos matriculados em “Matemática B”, matéria ministrada pelo Dr. José.
Fonte	(inserir fonte de referência com explicações sobre a regra)
Regras relacionadas	RN65 – Período ativo de matérias. RN200 – Modificando nota de alunos.

Fonte tabela 3 - (AMBLER, 2001).

2.3.1.3 Change Case

Change cases são utilizados para descrever potenciais requisitos ou para modificar algum já existente. Deve ser descrito de uma maneira simples, relatando apenas a alteração em potencial para um requisito, indicando as chances de que essa mudança ocorra e qual o impacto dela. São artefatos simples e diretos, tornando-os fácil de compreender (AMBLER, 2001).

Change cases possibilitam o time considerar casos à longo prazo, tornando mais fácil a decisão de questões de arquitetura do sistema, evitando implementações muito robustas para funcionalidades que não as exigem, e em casos onde a chance do change case ocorrer for muito grande, ela pode ser tratada como um requisito normal do sistema sendo então priorizada pelos stakeholders e então implementada.

A tabela 4 a seguir mostra um exemplo de utilização deste artefato.

Tabela 4 – Exemplo de especificação de change case

Change case: Cadastramento vai ocorrer inteiramente pela internet.
Probabilidade: Média dentro de 3 anos. Alta para até 10 anos.
Impacto: Desconhecido. Embora o cadastramento estará disponível para ser feito online em setembro, espera-se que menos de um quarto deles será feito pela internet. Tempo de resposta do servidor pode se tornar um problema durante períodos de pico com muitos acessos, que são as primeiras duas semanas antes do início do semestre, assim como a semana posterior.

Fonte tabela 4 - (AMBLER, 2001).

2.3.1.4 CRC Model

A criação do CRC Model veio da necessidade de documentar decisões de design de forma colaborativa (BECK e CUNNINGHAM, 1989).

Um modelo de Classe, Responsabilidade e Colaboradores (CRC) é uma coleção de *index cards* divididos em três seções (Figura 12).

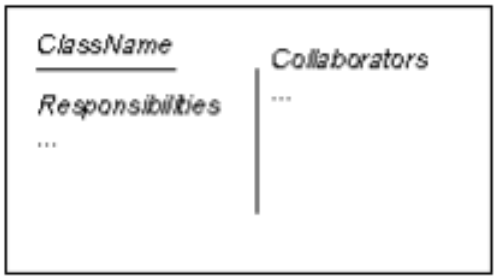


Figura 12 - CRC Card (BECK e CUNNINGHAM, 1989).

Uma classe representa uma coleção de objetos similares, podendo ser uma pessoa, lugar, coisa, evento ou conceito relevante para o sistema. Por exemplo, para um sistema de uma universidade, a classe poderia ser um estudante, professor, uma prova ou matéria.

Uma responsabilidade é algo que a classe conhece ou faz. Por exemplo, um estudante conhece seu um nome, endereço e telefones, assim como também

matricula-se em matérias e realiza provas. Uma classe pode alterar os valores que conhece, mas não pode fazer o mesmo para outras classes.

Algumas vezes uma classe possui responsabilidades a cumprir, porém não possui todas as informações para isso. Por exemplo, um estudante matricula-se em matérias, mas para isso precisa saber se ainda há vagas disponíveis, portanto, a classe estudante colabora ou interage com a classe Matéria.

Tabela 5 - Exemplo de especificação de CRC Model (classe professor).

Professor	
Nome Endereço Telefone E-mail Salário Leciona matérias	Matérias

Fonte tabela 5 - (AMBLER, 2001)

Tabela 6 - Exemplo de especificação de CRC Model (classe matérias).

Matérias	
Nome Número Taxa de matrícula Lista de espera Alunos matriculados Instrutor Adiciona estudante Remove estudante	Estudante Professor

Fonte tabela 6 - (AMBLER, 2001)

Um CRC Model pode ser elaborado conforme exemplos da tabela 5 e tabela 6, que mostram duas classes, suas responsabilidades, e como elas se relacionam através do elemento “colaboradores”.

2.3.1.5 Constraint Definition

Uma contraint definition é uma restrição à nível global do sistema para se limitar o uso de uma determinada solução proposta. São documentadas de forma parecida com regras de negócio e requisitos técnicos (AMBLER, 2001).

Exemplos de constraint definition:

- O sistema deve funcionar na infraestrutura já existente;
- O sistema só poderá utilizar dados dos bancos disponíveis dentro da rede corporativa;
- O sistema deve estar disponível 99.99% dado qualquer intervalo de 24 horas;

2.3.1.6 Diagrama de Fluxo de Dados

Diagramas de fluxo de dados (Figura 14) mostram o fluxo de dados provenientes de entidades externas para o sistema, o fluxo de dados entre os processos e os bancos de dados onde são armazenados. Existem apenas 4 símbolos para construção de um diagrama de fluxo de dados (Figura 13):

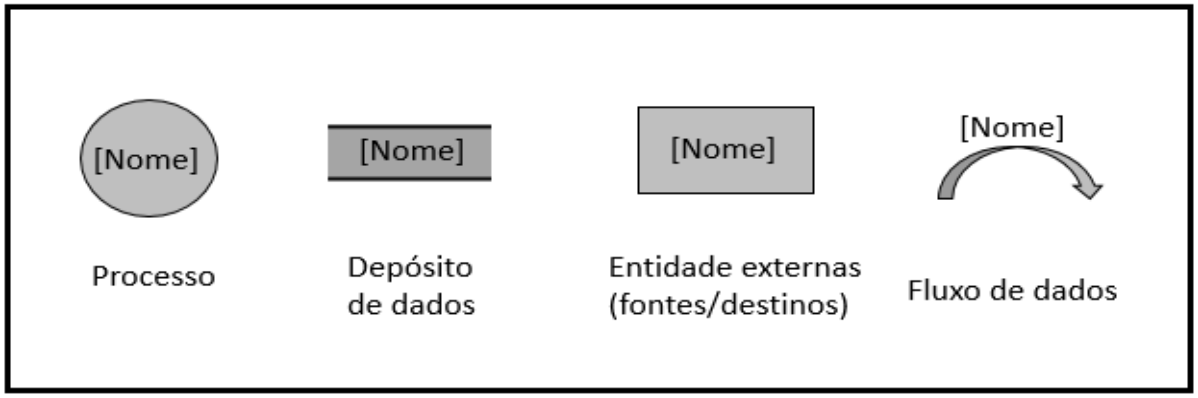


Figura 13 - Elementos de modelagem DFD (POHL e RUPP, 2012).

1. Fontes/destinos: Quadrados que descrevem objetos no ambiente do sistema e que trocam dados com o mesmo. São aspectos do ambiente e não pode ser alterados pelo desenvolvimento do sistema. Fontes fornecem dados ao passo que destinos recebem dados do sistema;
2. Processos: retângulos arredondados ou círculos que representam as funções necessárias em um determinado sistema para transformar os dados que fluem

para o sistema. Um processo recebe e processa os dados e gera uma saída. Como os dados são transformados não é representado pelo diagrama;

3. Fluxo de dados: setas que descrevem dados que são transportados entre processos, repositórios de dados e fontes/destinos (YOURDON, 1988). Geralmente apenas fluxos de dados;
4. Repositório de dados: retângulos abertos que representam conceitos abstratos projetados para representar dados persistentes. Processos podem acessar os dados como forma de dado de entrada ou escrever dados em repositórios como forma de dados de saída.

Para utilizar o DFD de uma maneira ágil, os diagramas devem ser pequenos e deve-se utilizar ferramentas simples. Muitas vezes, apenas um quadro branco é suficiente. E só devem ser criados se forem adicionar valor, não apenas porque o processo o requere.

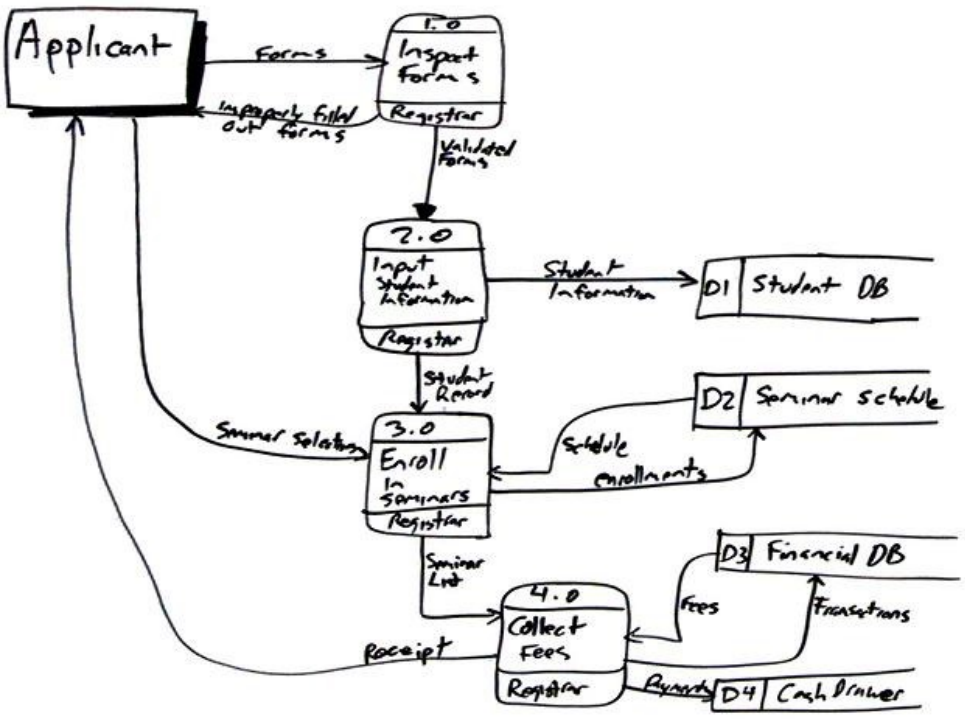


Figura 14 - Exemplo de diagrama de fluxo de dados (AMBLER, 2001).

2.3.1.7 Use Case

Segundo Cockburn (2001), use cases capturam um contrato entre os *stakeholders* do sistema, sobre o seu comportamento. Os use cases descrevem o comportamento do sistema sob várias condições à respostas de requisições realizadas por um usuário ou outros sistemas, chamado de ator.

Um caso de uso é uma peça de funcionalidade no sistema que dá ao usuário um resultado de valor. Casos de uso capturam requisitos funcionais do sistema. Todos os casos de uso juntos compõem o modelo de caso de uso que descreve a funcionalidade completa do sistema (JACOBSON, BOOCH e RUMBAUGH, 1998)

Casos de uso generalizam a intenção do uso da implementação de forma independente. Um caso de uso bem descrito possui uma narrativa estruturada, usa uma linguagem do domínio da aplicação e dos usuários, e compõe de forma simples, abstrata, e independente de tecnologia, uma descrição de tarefa ou interação.

Casos de uso normalmente são escritos utilizando duas colunas, sendo a coluna da esquerda para indicar a intenção do usuário/ator, e a da direita a responsabilidade do sistema para esta intenção (tabela 7). O ator deve tentar executar uma ação e espera receber uma ou mais respostas para esta ação do sistema.

De acordo com Cockburn (2001), a maior diferença entre o formato de documentar os use cases são o quanto elaborado eles são, então, para projetos onde times grandes trabalham em um software complexo e crítico, o custo de se ter mais cerimônias para elaborar use cases bem descritos e bem detalhados para se evitar ambiguidades e desentendimentos é valido, ao passo que times pequenos trabalhando em software de menor impacto em caso de falhas, utilizar um *template* mais simples para a elaboração de use cases é tolerável e vai poupar tempo.

O caso de uso pode fazer referência à regras de negócio, entretanto não às descrevem por completo, por isso a importância de um identificador. No exemplo dado, três itens são opcionais, porém muito úteis. O identificador, que pode ser utilizado caso o artefato precise ser referenciado em outro local, e as pré-condições e pós-condições, que definem valores que devem ser verdadeiros antes e após a execução do caso de uso.

Tabela 7 – Exemplo de caso de uso

ID: UC001 Caso de uso: matricular-se em matéria. Pré-condições: - A situação da matrícula do estudante está regular. Pós-condições: - Nenhuma.	
Intenção do usuário/ator	Responsabilidade do sistema
Estudante se identifica	Verifica a elegibilidade para matricular-se através da *RN001 determinar elegibilidade para matricular-se em matérias*; Indicar matérias disponíveis.
Escolher matéria	Valida escolha da matéria pela *RN002 Determinar elegibilidade de matricular-se na matéria*; Validar se encaixa na agenda pela *RN003 Validar agenda do estudante*; Calcular taxas através da *RN004 Cálculo de taxa do estudante* e *RN005 Cálculo de taxa da matéria*.
Confirmar matrícula	Sumarizar valores de taxas; Solicitar confirmação; Matricular estudante na matéria; Adicionar taxas à conta do estudante; Providenciar confirmação da matrícula.

Fonte tabela 7 - (AMBLER, 2001).

Os casos de uso podem fazer referência à regras de negócio, entretanto não às descrevem por completo, por isso a importância de um identificador. No exemplo dado, três itens são opcionais, porém muito úteis. O identificador, que pode ser utilizado caso o artefato precise ser referenciado em outro local, e as pré-condições e pós-condições, que definem valores que devem ser verdadeiros antes e após a execução do caso de uso.

2.3.1.8 Protótipos de Interface (alta de baixa fidelidade)

Um protótipo de interface do sistema é uma técnica que permite pré-visualizar a interface do usuário no sistema e pode ser utilizada com vários propósitos (AMBLER, 2001). Permite explorar e validar o desenvolvimento do sistema em conjunto com os stakeholders e é descartável, ou seja, não será necessário codificá-la e descartar o código implementado em caso de inconformidade, evitando desperdício de recursos. Para construir um protótipo da interface, levanta-se as necessidades do usuário com relação àquela funcionalidade em questão, a partir disso o protótipo então é construído e avaliado. Se não for validado, o processo se repete, caso contrário, está pronto para ser desenvolvido. Os protótipo de baixa fidelidade normalmente são construídos somente com rascunhos, utilizando papel e caneta (Figura 15), mas também pode-se utilizar software específicos.

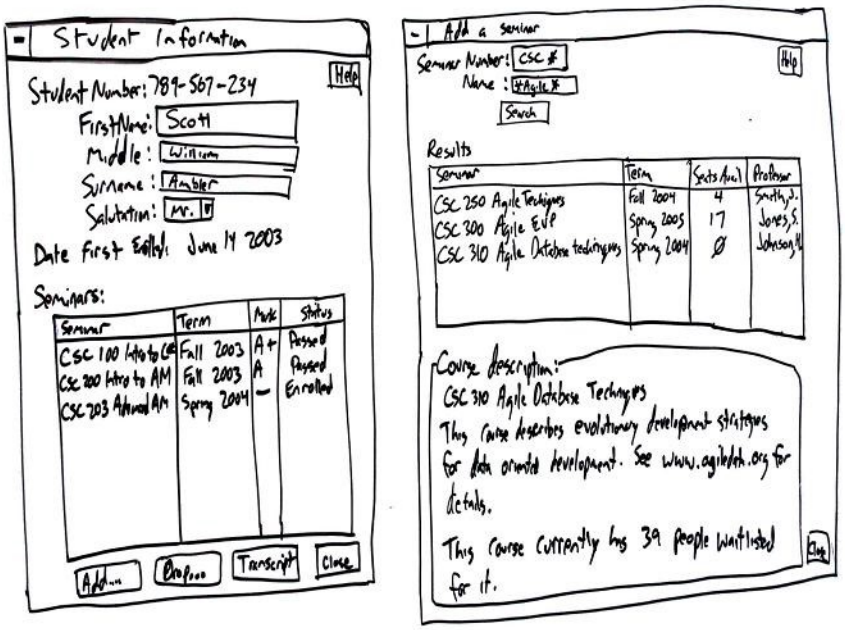


Figura 15 - Exemplo de protótipo de interface (AMBLER, 2001).

O objetivo é focar no usuário e usabilidade do sistema. Com esse tipo de artefato é possível modelar desde elementos grandes do sistema, como uma página

completa ou relatórios que podem ser gerados pelo software até pequenos elementos da tela, como *widgets* ou *inputs* que estarão disponíveis para interagir com o usuário.

Quando as necessidades da interface do usuário estão entendidas pode-se dar início à outra etapa de construção de um protótipo de maior fidelidade através de ferramentas específicas ou linguagem de alto nível, como HTML para sistemas web, convertendo os elementos levantados na construção dos protótipos de baixa fidelidade em elementos mais fiéis à construção final.

2.3.1.9 User Interface Flow Diagram

Protótipos de interface são bons para explorar a utilização de uma determinada interface do software, porém existe a possiblidade de perder-se nos detalhes deixando de lado o conjunto de requisitos como um todo, esquecendo as interações e relacionamento entre as interfaces do sistema. User interface flow diagrams, também conhecidos como storyboards, interface-flow diagrams, windows navigation diagrams, e context-navigation maps permitem modelar a relação entre diferentes interfaces facilitando levantar questões de usabilidade que vão além de uma única tela.

User interface flow diagrams são comumente utilizados de duas formas (AMBLER, 2001). Uma delas é modelar, dentro do contexto de um caso de uso, a interação do usuário com o software, ou seja, o caso de uso faz referência à determinadas interfaces do software, fornecendo informação de como elas são utilizadas, e então baseado nessas informações, é possível modelar um user interface flow diagram que reflete o comportamento individual deste caso de uso. A segunda forma, é modelar uma visão geral do uso das interfaces de todo o sistema (Figura 16),

contemplando uma combinação do comportamento de todas as *views* derivadas dos casos de uso.

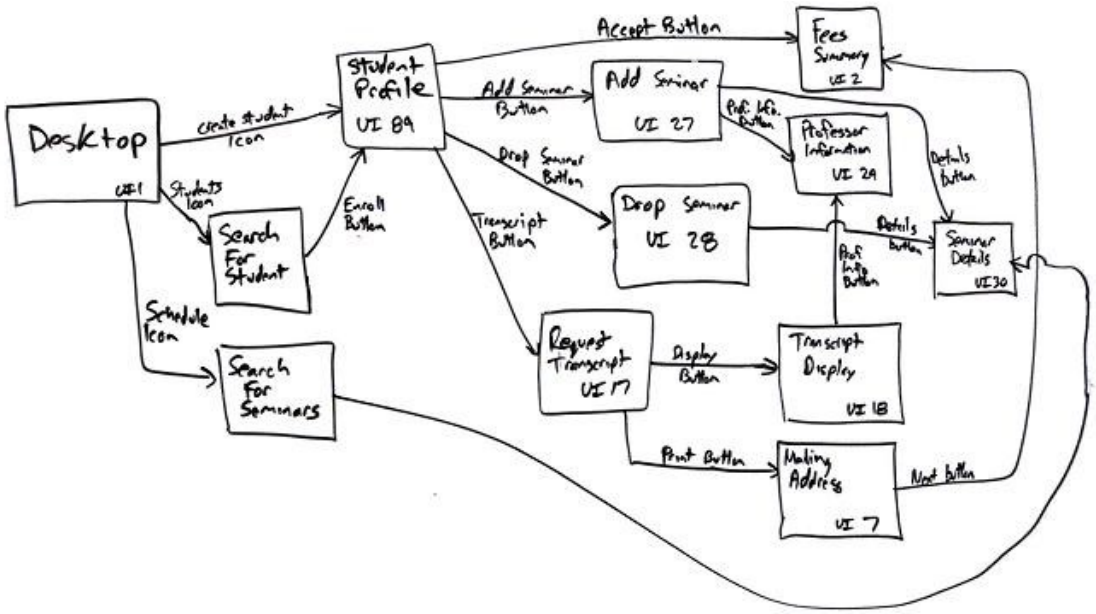


Figura 16 - Exemplo de user interface flow diagram (AMBLER, 2001).

2.3.1.10 Feature

Feature é um recurso, uma função pequena que entrega valor para o cliente. Features são muito pequenas e normalmente são implementadas em poucas horas (AMBLER, 2001).

Uma feature é um elemento distinto ou funcionalidade do sistema que provê capacidade ao negócio, por exemplo, em um sistema que controla um fórum, uma feature poderia ser “permitir usuários adicionar comentários”.

2.3.1.11 Requisitos Técnicos

Um requisito técnico diz respeito aos aspectos técnicos que o sistema deve cumprir, como questões relacionadas à performance, segurança, disponibilidade e

confiabilidade (AMBLER, 2001). São também chamados de requisitos não funcionais. São constituídos por um identificador e uma descrição, e podem ser documentados da mesma forma que regras de negócio com exemplos, fontes de informação e referências à outros requisitos técnicos (tabela 8).

Tabela 8 - Exemplos de requisitos técnicos

ID	Descrição
RT34	O sistema deve estar disponível 99.99% do tempo dado qualquer período de 24 horas.
RT78	A consulta de matérias deve ocorrer em menos de 3 segundos em 95% das vezes.
RT79	A consulta de seminários deve ocorrer em menos de 10 segundos 99% das vezes.

Fonte tabela 8 - (AMBLER, 2001).

2.3.1.12 Diagrama de Caso de Uso

Diagramas de caso de uso são modelos simples para documentar de forma esquemática as funções de um sistema a partir do ponto de vista do usuário, bem como as inter-relações das funções de um sistema e as relações entre essas funções e seu ambiente (POHL e RUPP, 2012).

O diagrama de caso de uso (Figura 17) é uma representação da interação do usuário com o sistema, mostrando o relacionamento entre o ator e os diversos casos de uso em que está envolvido. Os diagramas são mais utilizados para apresentar aos stakeholders, enquanto os casos de uso são melhores aproveitados pelo desenvolvimento pois contém mais detalhes.

O diagrama é composto por casos de uso, atores e associações, e também são opcionais a utilização de blocos limitadores e pacotes. Os casos de uso descrevem uma tarefa ou interação do usuário com o sistema. O ator pode ser uma pessoa, organização ou sistema externo que interage com o sistema em especificação. As associações são representadas por linhas retas conectando atores

à casos de uso, elas são utilizadas sempre que existir uma interação entre um ator e um caso de uso, sendo opcional, o uso de uma seta em uma das extremidades para indicar a direção de quem inicia a ação. Os blocos limitadores são representados por retângulos envolvendo casos de uso para indicar o escopo do sistema, ou seja, o que não estiver envolvido pelo bloco, não será desenvolvido, sendo um recurso opcional e pouco utilizado. Por fim, os pacotes são recursos utilizados para agrupar os elementos de um modelo para fins de organização.

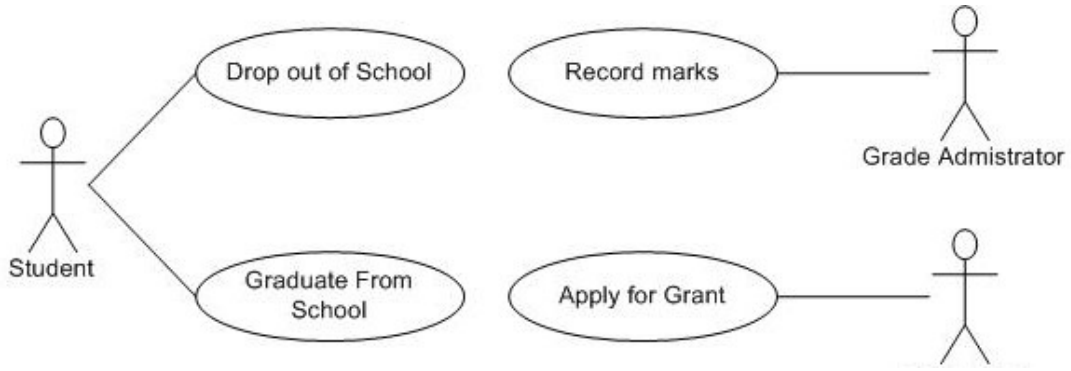


Figura 17 - Exemplo de diagrama de caso de uso (AMBLER, 2001).

As associações podem existir também entre elementos do mesmo tipo. Os casos de uso podem relacionar-se através de uma associação de ampliação, inclusão ou herança, já os atores só podem relacionar-se pela associação de herança (Figura 18).

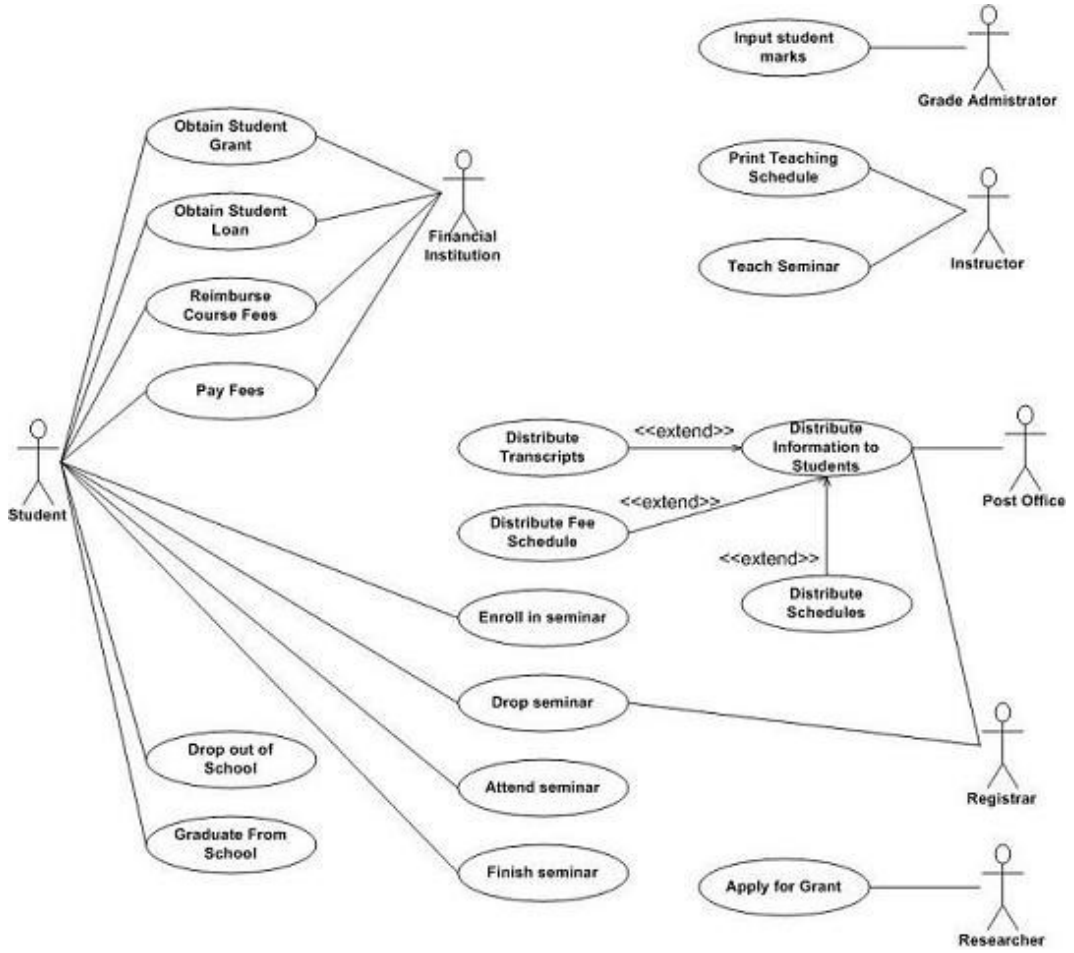


Figura 18 - Exemplo de diagrama de caso de uso e associação entre elementos do mesmo tipo (AMBLER, 2001).

2.3.1.13 Diagrama de Atividades

Diagramas de atividades representam o fluxo de controle entre atividades ou ações. No caso de uma progressão sequencial de ações, uma ação subsequente é executada depois que cada ação precedente termina (POHL e RUPP, 2012).

Diagrama de atividades (Figura 19) são mais utilizados para modelagem de processos, de lógicas de caso de uso ou de lógicas de regras de negócio. Uma utilização muito comum do diagrama de atividades é para modelagem de casos de uso pois podem descrever o caminho básico de ações, assim como caminhos alternativos. Podem ser utilizados sem o envolvimento de casos de uso, por exemplo,

em conjunto com *stakeholders* para analisar user stories ou processos de negócio muito extensos.

Uma metodologia descrita por Ambler (2001) define passos para modelar um digrama de atividades, definindo tarefas que devem ser executadas de forma iterativa.

1. Identicar o escopo do diagrama de atividades. Definir se a modelagem será referente à um único caso de uso, um porção de caso de uso, um processo de négocio envolvendo diversos casos de uso, etc. Após identificá-lo, dar um título e um identificador único;
2. Adicionar um ponto de início e um ponto final. Todo diagrama de atividades tem um início e um fim. Alguns autores descrevem o ponto final como opcional;
3. Adicionar atividades. Ao modelar casos de uso, adiciona-se uma atividade para cada passo iniciado por um ator. Ao modelar processos de negócio, utiliza-se uma atividade para cada processo macro. Para diagramas de atividade de código, é utilizada uma atividade para cada método;
4. Adicionar transições entre as atividades. Quando existe mais de uma saída para a mesma atividade, a mesma deve ser rotulada.
5. Adicionar pontos de decisão. Muitas vezes, a lógica exige um ponto de decisão, quando algo é avaliado ou comparado, sendo o destino dependente do resultado desta ação. Pontos de decisão são opcionais;
6. Identificar potenciais atividades paralelas. Duas atividades podem correr em paralelo quando não há dependências ou relações diretas entre elas, porém ambas precisam acontecer antes que uma terceira atividade seja iniciada.

Diagramas de atividades são documentados com uma descrição breve das atividades e das ações que são executadas no processo. Também podem ser

colocados referências à casos de uso ou método codificados. A Figura 20 apresenta importantes elementos de modelagem dos diagramas de atividades.

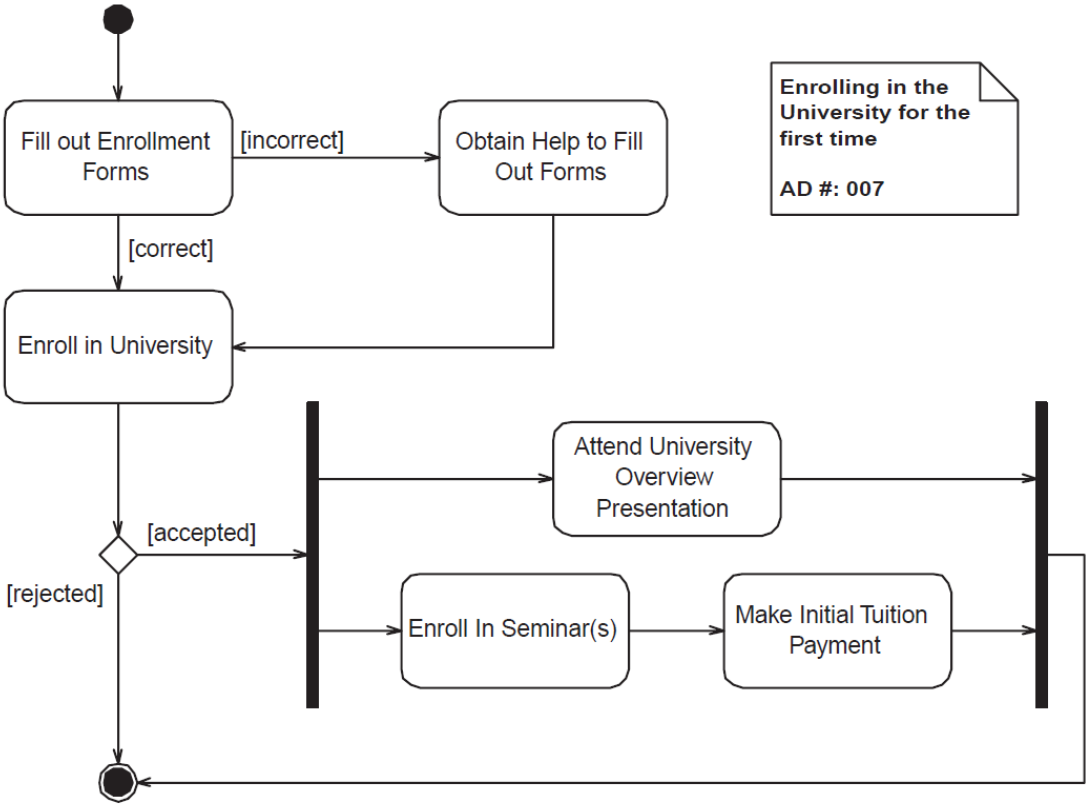


Figura 19 - Exemplo de diagrama de atividades (AMBLER, 2001)

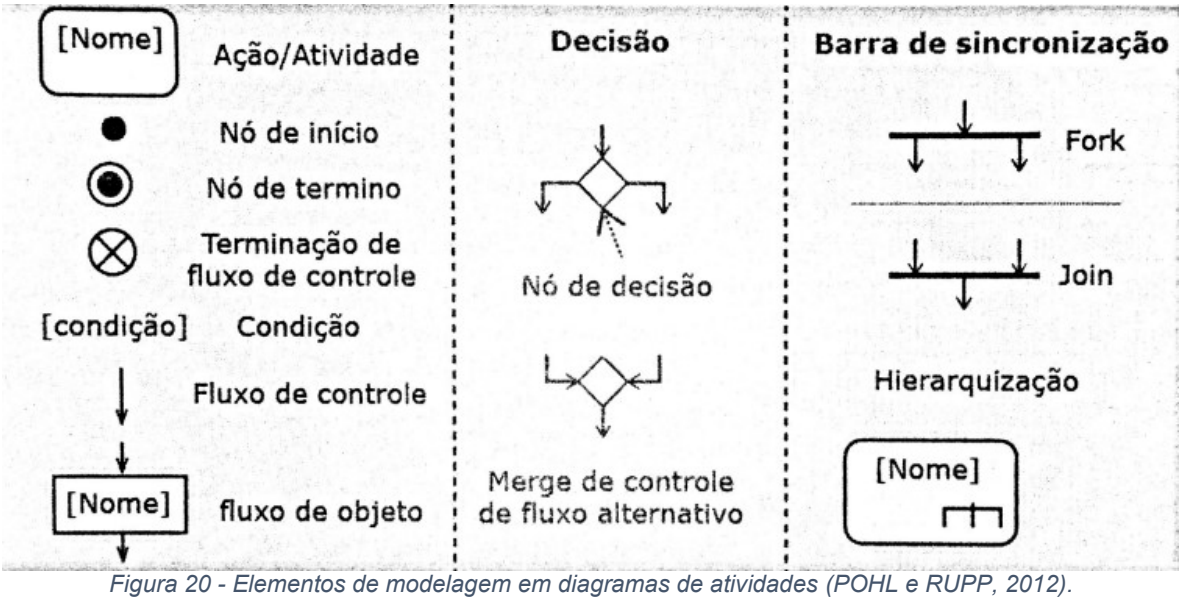


Figura 20 - Elementos de modelagem em diagramas de atividades (POHL e RUPP, 2012).

2.3.1.14 User Story

Uma user story é um artefato de requisito de alto nível e faz uma breve descrição, suficiente apenas para que o time de desenvolvimento possa estimar o esforço necessário para concluí-la. Não necessita de nenhuma ferramenta ou linguagem padrão para defini-la. Podem ser feitas utilizando apenas *post its* e caneta (Figura 21) e descrevem requisitos funcionais ou não funcionais.

Para construir uma user story, Cohn (2004) sugere um formato padrão:

"Como um (papel/função) eu gostaria (alguma funcionalidade) para que (benefício)"

Este formato permite deixar claro o que deve ser feito, para quem será feito e qual o objetivo este usuário pretende alcançar utilizando esta funcionalidade.

Por serem requisitos de alto-nível, algum nível de detalhamento deve ser levantado durante o ciclo de vida do desenvolvimento da *feature* e para isso são utilizados critérios de aceite, também chamado de teste de aceitação, descritos na seção "2.3.1.1 Testes de aceitação", que são breve descrições que validam que o que foi implementado está de acordo com o que foi solicitado.

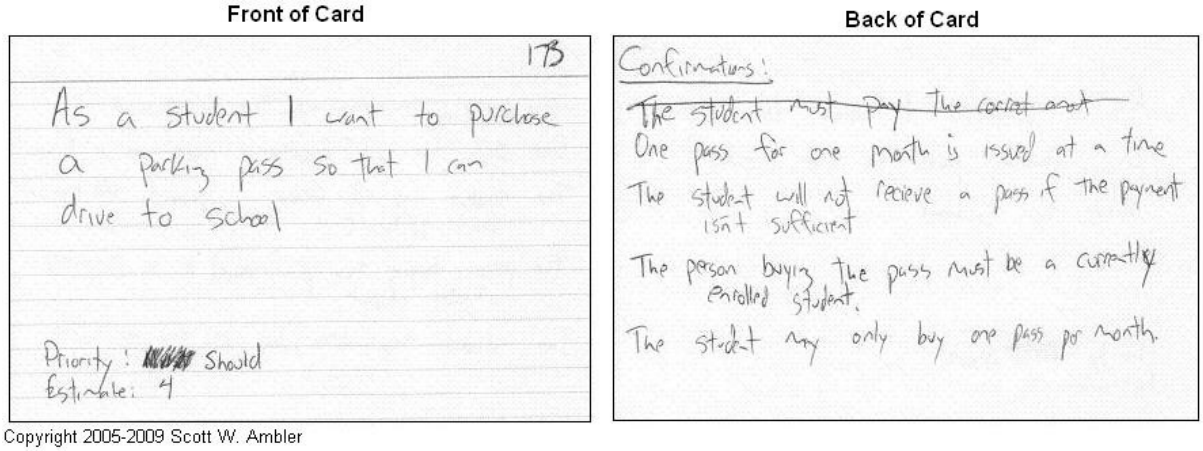


Figura 21 - Exemplo de história de usuário e critérios de aceito (AMBLER, 2001).

3. Processo Atual

3.1 Contexto

O projeto da organização que foi selecionada pois é a empresa onde o autor deste estudo de caso trabalha, desenvolve um software que fornece soluções web automatizadas para empresas da área da construção civil. O software é desenvolvido e mantido por uma equipe de desenvolvimento separada em 10 times, responsáveis por módulos específicos do sistema. A equipe que passou pelo estudo de caso consiste de cinco integrantes no time de desenvolvimento, um *scrum master*[1](SM), função exercida pelo autor do estudo de caso, e um *product owner*[2](PO), e desenvolve e mantém os módulos responsáveis pelas regras de negócio que envolvem os processos que tratam da parte comercial da área da construção civil.

Para melhorar o processo de desenvolvimento com uma forte orientação voltada à geração e entrega de valor para os clientes, a gestão e controle da manutenção evolutiva e corretiva do software é mantida por uma ferramenta chamada IBM Rational Team Concert, utilizada por todos os membros do projeto, e para a produção de novas *features*, evolução do sistema, os times foram introduzidos a uma metodologia de desenvolvimento ágil chamada *Scrum*, e vêm utilizando esta desde o ano de 2012.

Desde a adoção do *Scrum* muito foi aprendido, e as experiências de se trabalhar com um modelo de desenvolvimento iterativo trouxe melhoras significativas para os times do projeto e para o produto, entretanto a especificação de requisitos

[1] O Scrum Master ou SM é responsável por garantir que o Scrum seja entendido e aplicado. O Scrum Master faz isso para garantir que o Time Scrum adere à teoria, práticas e regras do Scrum. O Scrum Master é um servo-líder para o Time Scrum (SUTHERLAND e SCHWABER, 2013).

[2] O Product Owner, PO, ou dono do produto, é o responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento. Como isso é feito pode variar amplamente através das organizações, Times Scrum e indivíduos (SUTHERLAND e SCHWABER, 2013).

sempre foi um assunto levantado pelo time de desenvolvedores como um problema, e por isso, ao longo dos anos já sofreu diversas adaptações para melhor atender à necessidade de quem a produz e consome, porém, a identificação da potencial melhora que estas adaptações trouxeram foram feitas de uma forma subjetiva e não objetiva.

Assim, um programa de medição deve levantar dados que possam encontrar e identificar a potencial melhora de adaptações à especificação de requisitos de forma objetiva para deixar mais claro e transparente os benefícios e problemas que a ausência de certos tipos de artefatos de requisitos podem trazer com o objetivo de melhorar a qualidade funcional e estrutural do produto e a qualidade do processo de desenvolvimento.

3.2 Processo de Desenvolvimento

O processo de desenvolvimento utilizado no projeto do estudo é baseado no *Scrum*, um framework para desenvolver e manter produtos complexos (SUTHERLAND e SCHWABER, 2013), guiado pelos 4 princípios centrais do manifesto ágil (2001).

Um time que utiliza Scrum desenvolve software em intervalos de tempos chamados sprints[3]. A duração de uma sprint é consistente, porém pode variar de uma à quatro semanas dependendo do time. A equipe do estudo utiliza sprints com duração de uma semana.

[3] Um período com tempo determinado e fixo que pode ser definido em até 1 mês, durante o qual um incremento do software potencialmente utilizável do produto é criado (SUTHERLAND e SCHWABER, 2013).

A sprint é uma iteração que consiste de diversas etapas conforme Figura 22. A primera etapa da sprint é o processo de planejamento, onde são separados os itens do backlog do produto[4], em que o time julga possível de serem desenvolvidos até o final da iteração e se comprometem a finalizar. O backlog do produto contém todas as histórias a serem desenvolvidas para o software e estão ordenados por prioridade.

Para definir e incluir os itens, o time de desenvolvimento realiza estimativas para cada item, baseando-se em tamanho e complexidade, e pontuando em *story points*. *Story points* são uma unidade de medida para expressar uma estimativa do esforço global que será necessário para implementar plenamente um item do backlog do produto ou qualquer outro trabalho (COHN, 2016). Com base no histórico da quantidade de *story points* que foram concluídos em sprints passadas é possível abstrair a velocidade do time, que é o valor que determina a quantidade de itens que serão incluídos nas sprints seguintes.

Figura 22 - O processo de desenvolvimento Scrum (SCRUM ALLIANCE®, 201?).

[4] O Backlog do Produto é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto (SUTHERLAND e SCHWABER, 2013).

O conjunto de itens selecionados para desenvolvimento para a sprint constitui o backlog da sprint. Assim que o processo de planejamento é encerrada e o backlog da sprint é construído, dá-se início ao desenvolvimento dos itens selecionados. Diariamente o time de desenvolvedores se reúne para uma discussão de curto prazo (alguns minutos), chamada de *daily scrum,* quando é discutido sobre o progresso e verificado se o andamento está conforme planejado e o time está caminhando em direção aos seus objetivos, para caso contrário, seja possível tomar um plano de ação o quanto antes.

Ao final de uma sprint, é realizada então a cerimônia de *review* quando os itens implementados são apresentados aos *stakeholders* para verificar se estão de acordo com os requisitos levantados e nenhuma inconsistência foi encontrada, seguida por outra cerimônia de retrospectiva na qual o time faz uma análise do decorrer da sprint para determinar aspectos do produto ou do processo que precisam de melhoria.

3.2.1 Processo do Time do Estudo de Caso

O processo do time de desenvolvimento do estudo de caso, no qual o autor do presente trabalho desempenha o papel de Scrum Master, está divido em duas etapas, planejamento e execução (Figura 23), e inicia sempre na segunda feira seguinte ao término da sprint anterior. O resultado final ao término de uma sprint devem ser histórias de usuário no backlog da sprint implementadas, também chamadas de incremento[5], e histórias de usuário provenientes de solicitações de evolução do sistema devidamente refinadas, especificadas.

[5] O incremento é a soma de todos os itens do Backlog do Produto completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores (SUTHERLAND e SCHWABER, 2013).

No processo de planejamento, na primeira etapa da sprint, ocorre o *sprint planning*[6], a primeira tarefa do processo (Figura 23), que reúne o time de desenvolvimento para planejar os itens do backlog do produto que serão implementados e construir o sprint backlog, artefato que mostra quais histórias foram selecionadas para implementação na sprint e que são expostos em um quadro kanban (Figura 24). A primeira parte da reunião consiste de uma apresentação dos itens do backlog do produto com os requisitos já elicitados e especificados aos desenvolvedores, e se necessário, sanar dúvidas referentes às implementações. A segunda parte do planejamento fica reservada ao time para discutir sobre o desenvolvimento.

Após a etapa de planejamento, o restante dos dias da sprint é utilizado para a execução de dois processos paralelos, o *grooming*[7] ou refinamento (Figura 25) e a implementação de história (Figura 26).

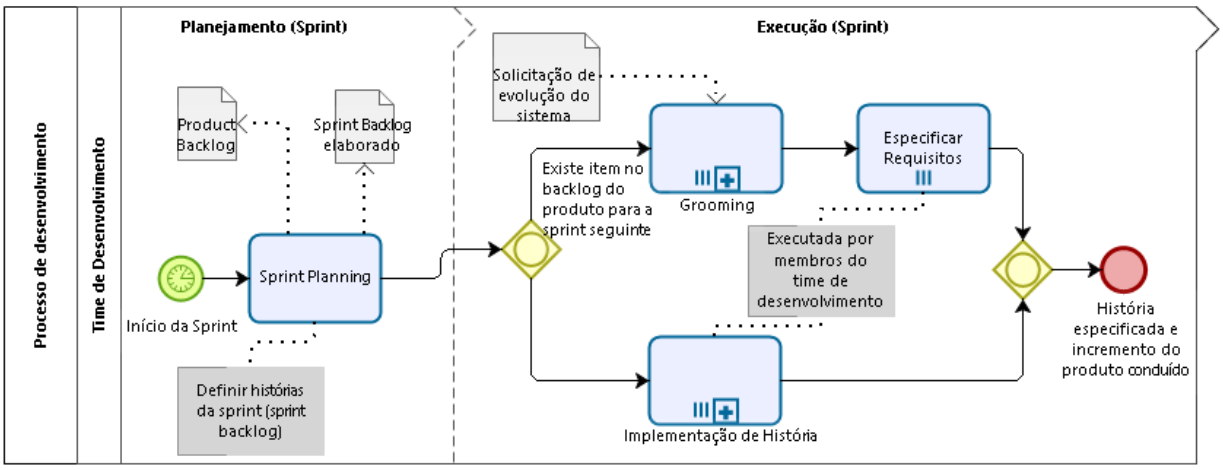


Figura 23 - Processo de desenvolvimento do time do estudo.

[6] O trabalho a ser realizado na Sprint é planejado na reunião de planejamento da Sprint. Este plano é criado com o trabalho colaborativo de todo o Time Scrum (SUTHERLAND e SCHWABER, 2013).
[7] O refinamento do Backlog do Produto é a ação de adicionar detalhes, estimativas e ordem aos itens no Backlog do Produto. Este é um processo contínuo em que o Product Owner e o Time de Desenvolvimento colaboram nos detalhes dos itens do Backlog do Produto (SUTHERLAND e SCHWABER, 2013).

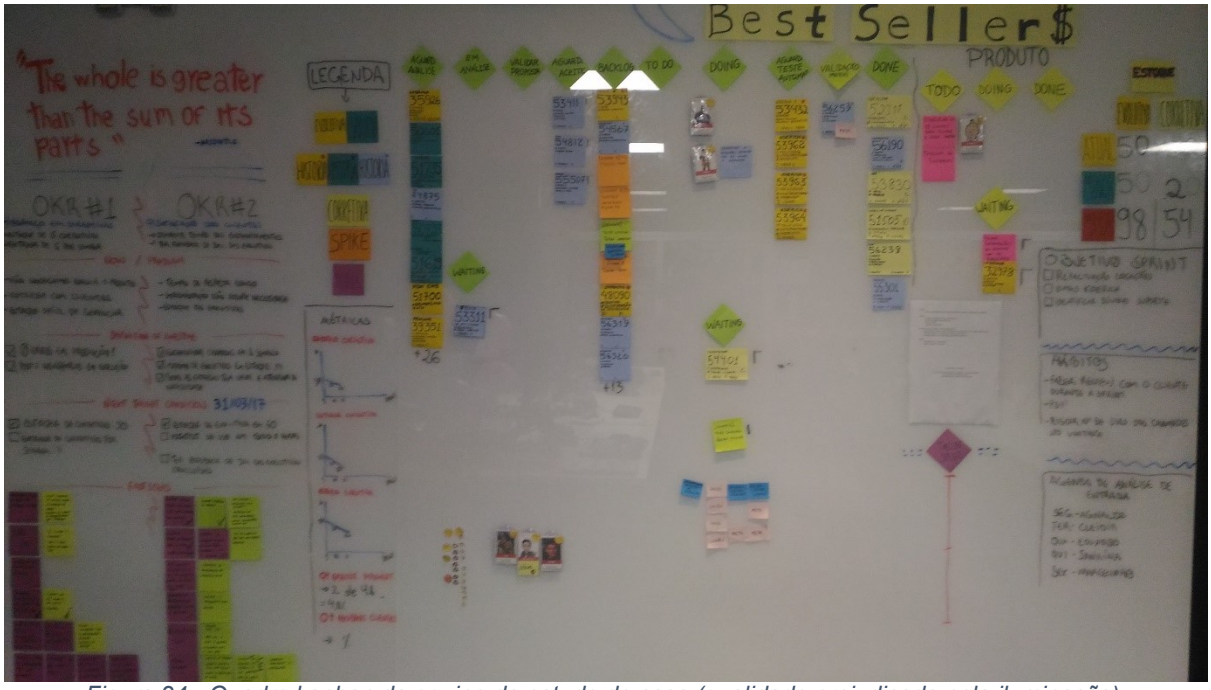


Figura 24 - Quadro kanban da equipe do estudo de caso (qualidade prejudicada pela iluminação).

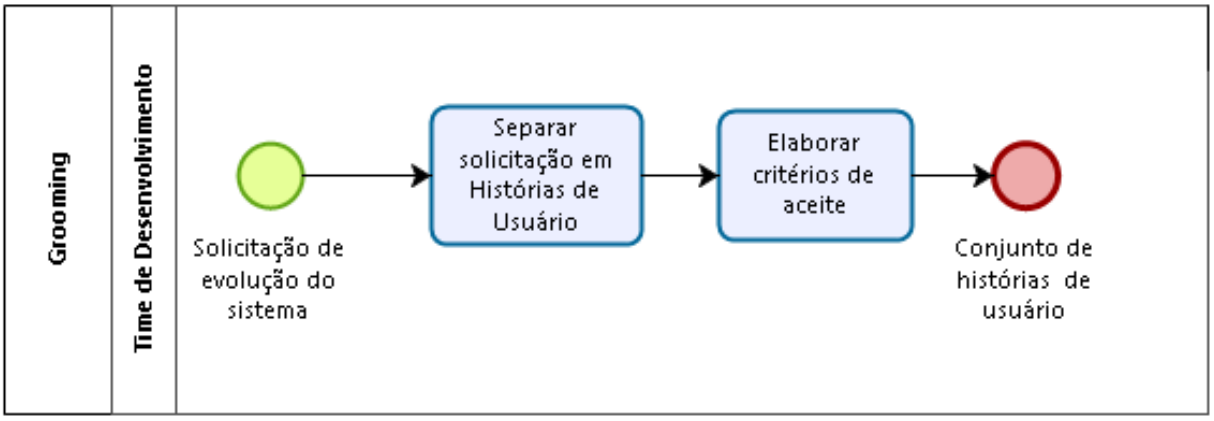


Figura 25 – Processo de grooming.

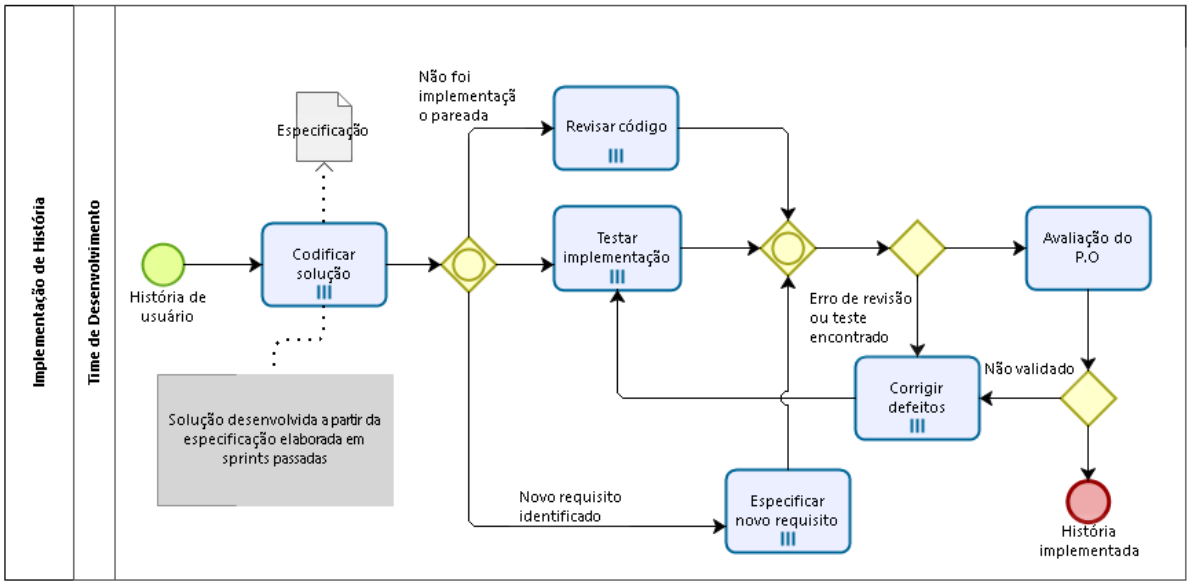


Figura 26 - Processo de implementação de história.

O processo do *grooming*, também chamado de refinamento, é realizado quando existe uma demanda de evolução do sistema (solicitação de evolução do sistema (Figura 23). O objetivo deste processo é fazer uma análise da demanda com o time de desenvolvimento, avaliar a forma mais eficiente e rápida de entregar valor para o cliente separando uma única solicitação de evolução em histórias de usuário menores, que podem ser implementadas em um menor espaço de tempo e entregues ao cliente antes que toda a solução seja desenvolvida, e estimar em pontos chamados de *story points* que são utilizados para medir a velocidade do time e definir quantas histórias serão colocadas no sprint backlog.

Para separar a demanda de evolução do sistema em histórias, o time utiliza o conceito de INVEST (WAKE, 2003). Segundo Wake (2003), as características que definem uma boa história de usuário são: independente; negociável; deve entregar valor; estimável; pequena; e testável. Para estimar as histórias de usuário, o time a avalia de acordo com três dimensões: o esforço; a complexidade; e a incerteza, e também faz comparações com histórias de pontuação semelhantes já implementadas em sprints passadas com intuito de obter uma melhor precisão avaliando se essas outras histórias teriam uma pontuação maior ou menor dentro das três dimensoes, e então calibrar a pontuação com valores mais próximos de histórias já conhecidas e com pontuações de valores semelhantes.

Após a elaboração de histórias de usuário são então levantados quais os critérios cada uma delas deve estar de acordo para que sejam consideradas como concluídas, e com isso, o processo é encerrado, gerando como saída, um conjunto de histórias de usuário com seus devidos critérios de aceite e estimadas em *story points*.

Tendo em mãos as histórias e critérios de aceite, os membros do time de desenvolvimento podem então dar início à tarefa de especificação dos requisitos (Figura 23), que deve elaborar os artefatos que servirão de suporte para os desenvolvedores implementar as histórias. Estas histórias então estão aptas para serem implementadas, portanto podem entrar no sprint backlog em futuras sprints.

As histórias de usuários elaboradas são compostas por 3 partes, uma curta descrição escrita com palavras chaves para planejar e relembrar o objetivo, uma seção chamada *conversation* também com curtas frases formadas por palavras chaves para declarar detalhes da história, e por último uma seção chamada de *confirmation* com testes de aceitação que são utilizados para determinar quando a história está completa (COHN, 2004), ou seja, quais os requisitos que o sistema deve estar cumprindo para que a história atenda ao que propõe fazer. Para complementar as histórias de usuário que necessitam de requisitos mais específicos e detalhados também é utilizado o artefato de regra de negócio, já comentado na seção 2.3.1.2 R deste trabalho.

O processo de implementação de história (Figura 26) é iniciado pelo trabalho de “Codificar solução”, que é quando um implementador pega a primeira tarefa definida para a história para dar início à codificação, tarefa essa que foi planejada com base na especificação de requisitos, que contém os artefatos elaborados para guiar o implementador durante o desenvolvimento da nova funcionalidade. Mais de uma história pode ser implementada ao mesmo tempo, ou a mesma história pode ser implementada por diferentes desenvolvedores ao mesmo tempo, chamado de programação pareada. A tarefa de codificar solução também inclui implementar testes

automatizados para garantir o funcionamento dos critérios de aceite definidos na especificação.

Ao concluir uma tarefa de codificar solução, o código gerado para desenvolver a história é revisado por outro desenvolvedor quando é executada a tarefa “Revisar código”, porém somente quando não houve codificação pareada. Os requisitos implementados e definidos pela história são testados em conjunto pelo desenvolvedor e por outro integrante do time de desenvolvimento quando é executada a tarefa chamada de “*budy testing*” ou “*pair testing*” e, quando um requisito não previsto for identificado, o mesmo deve então ser especificado. Enquanto erros forem encontrados durante uma revisão de código ou teste exploratório, estes devem então ser corrigidos e testados novamente até que a história possa então ser validada pelo PO, que executa a tarefa focado na necessidade que motivou a implementação da história.

3.2.2 Modelo Atual de Especificação

Desde a adoção da metodologia Scrum de desenvolvimento a especificação tem sofrido diversas adaptações, seguindo modelos diferentes e elaboradas com ferramentas diferentes. O último modelo elaborado foi motivado pela implantação de uma nova ferramenta, IBM Rational Doors, que seria adotada pela organização, a qual iria impactar diretamente na forma como seriam escritos os requisitos. A ferramenta anterior, Enterprise Architect, utilizava notações UML e tinha um detalhamento muito mais visual, enquanto a nova ferramenta não, focando mais em detalhamentos descritivos.

Para elaborar o modelo atualmente utilizado com definições de métodos e técnicas de especificação foi seguido um plano com reuniões e discussões entre diversos membros do projeto inlcuindo diversos perfis (desenvolvedores, testadores, analistas de negócio e requisitos) para deixar claro para quem são feitas as especificações e porquê ela é elaborada, sempre levando em consideração o contexto do projeto e quais são os problemas já conhecidos por eles, e então chegar nos seguintes princípios que devem guiar a construção de especificações dos projetos da organização:

1. “Simplicidade é essencial, evitar esforço em trabalhos que não precisam ser realizados”. O maior desperdício é fazer muito bem feito algo que não precisaria ser feito.
2. “A comunicação verbal entre o time de desenvolvimento deve ser encorajada, não importando o nível de detalhamento da especificação”. Especificação sozinha não substitui comunicação, é preciso unir especificação escrita com comunicação verbal e valorizar sempre a conversa cara-a-cara.
3. “As diferentes visões do time enriquecem a especificação, portanto, devem ser incentivadas e antecipadas”. Todos os papéis do time devem ser incentivados a participar da elaboração da especificação.
4. “Documentar somente o necessário, nada mais e nada menos”. Documento tem que ter um motivo, documentar sem uma real necessidade não faz sentido, não devemos perder tempo com documentação que nunca será usada por ninguém.
5. “Especificar deve ser fácil”. Especificar tem que ser rápido e fácil. Devemos usar ferramentas que facilitem o trabalho, tanto para a escrita, quanto para

leitura. Se for fácil especificar, as chances de você fazer bem feito serão maiores.

6. “A especificação deve deixar claro a necessidade do cliente”. A necessidade deve deixar claro o porquê do desenvolvimento, se não temos claro o motivo do desenvolvimento, as chances de entregarmos algo sem valor para o cliente são maiores.

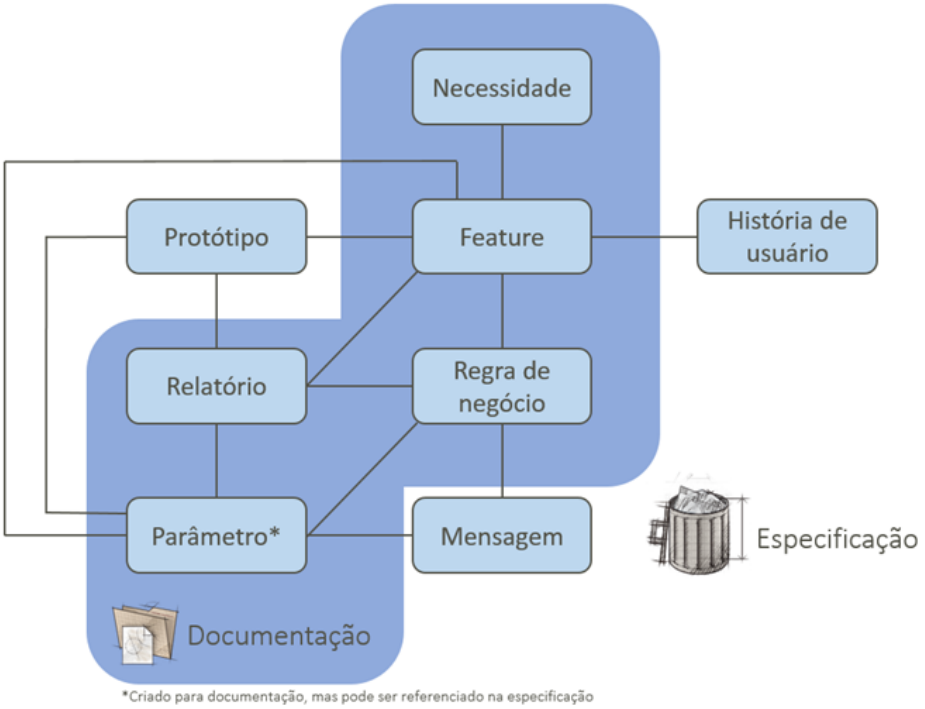


Figura 27 - Modelo de especificação e documentação.

Seguindo estes princípios, foram definidos que diferentes conjuntos de artefatos fariam parte da construção da documentação, e outro conjunto faria parte da construção da especificação (Figura 27), visto que são visões diferentes do software. A especificação de requisitos é uma coleção de artefatos do comportamento externo do sistema, criando um modelo conceitual do sistema a ser construído (LEFFINGWELL e WIDRIG, 2000) e a documentação diz respeito ao que é requerido do software fazer, e não “como” fazer (MACAULAY, 1996) portanto, aquilo que faz parte da especificação pode ser descartado após a implementação estar concluída já

que o “como” o sistema deve fazer pode mudar, ao passo que o que é requerido que faça, não.

As imagens a seguir mostram um exemplo do modelo atual de especificação com uma história (Figura 28), um protótipo (Figura 29) e uma regra de negócio (Figura 30).

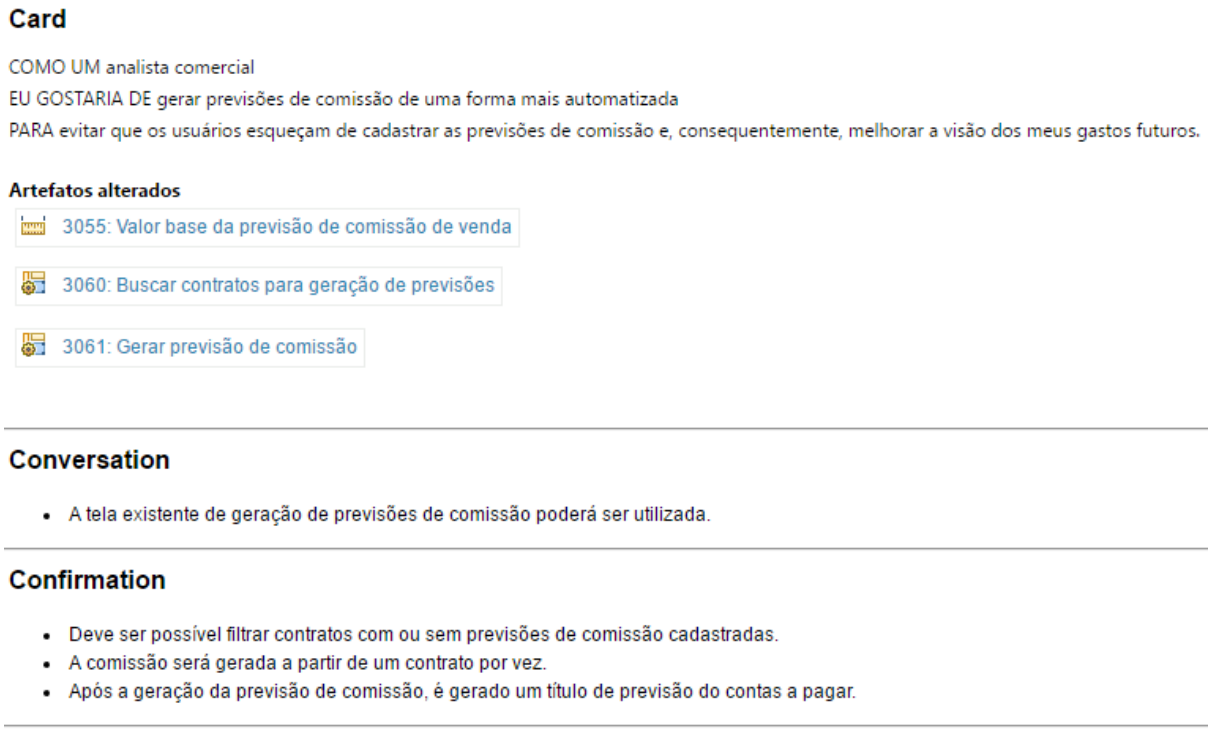


Figura 28 – Exemplo de história de usuário.

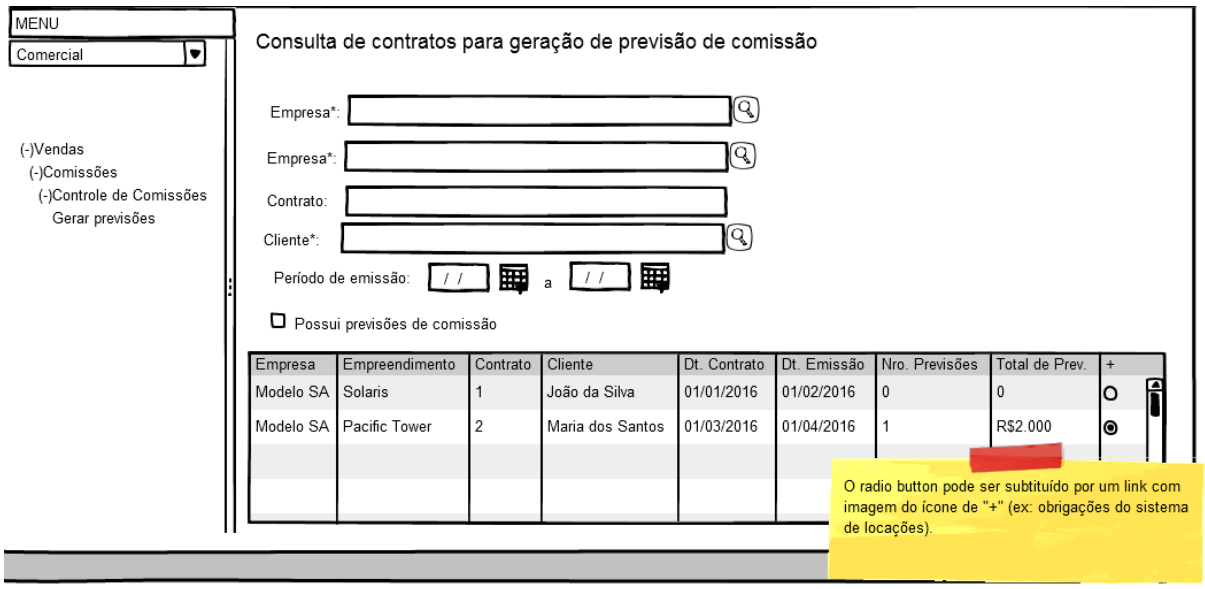


Figura 29 – Exemplo de protótipo de tela.

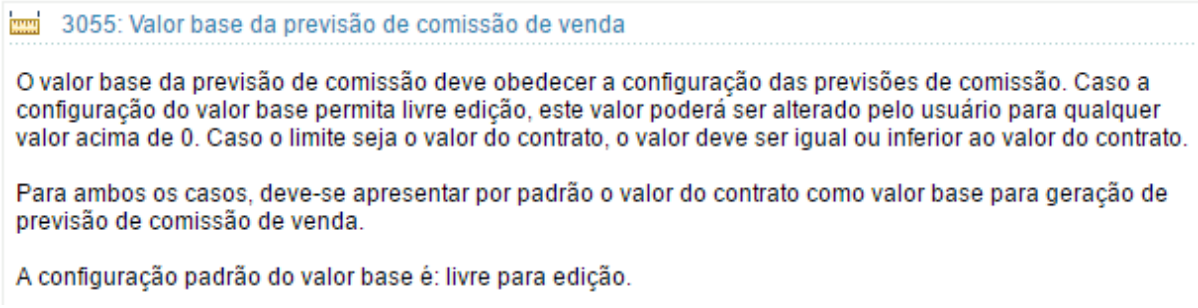


Figura 30 – Exemplo de regra de negócio.

3.2.3 Artefatos Propostos para o Estudo de Caso

Para a seleção dos artefatos propostos foi feita uma apresentação destes descritos na seção “2.3.1 Artefatos de Requisitos sob Práticas Ágeis” ao time de desenvolvimento selecionado para o estudo de caso com propostas de como utilizá-los. Por ser um time que segue os princípios ágeis (BECK, BEEDLE, *et al.*, 2001) e estarem mais familiarizados com os artefatos que metodologias e frameworks construídos pensando nestes princípios utiliza, a grande maioria foi recebida com um grau de desaprovação. Entretanto, foi ressaltado que no processo e modelo atual de especificação, não existe artefato que caracterize o comportamento do sistema. Com isso foi aceita a ideia da adoção de artefatos que descrevem o comportamento do

sistema para o estudo de caso, sendo um descritivo, o caso de uso (seção 2.3.1.7 Use Case), e outro gráfico, o diagrama de atividades (seção 2.3.1.13 Diagrama de Atividades), e que seriam experimentados em momentos diferentes, não em conjunto.

A forma como seriam adotados também foi proposta pelo time. Para que não afetasse o modelo atual e nem os princípios que guiam a construção da especificação, a proposta foi incluí-los na especificação em conjunto com os artefatos de história de usuário, sendo elaborados todos os comportamentos de fluxo esperado e de fluxo alternativo para cada uma delas.

4. Estudo de Caso

4.1 Aplicação do GQM em um Processo Ágil

Um processo ágil tem dois atores principais (HEIMANN, HENNESSEY e TRIPATHI, 2007). Um é o time de desenvolvimento, o qual se compromete a desenvolver software de acordo com um conjunto de requisitos dentro de um determinado limite de tempo. O outro é a gerência do projeto de forma geral. Heimann, Hennessey e Tripathi (2007) afirmam que de forma similar, as métricas para um processo ágil também devem ter duas partes, uma para o time de desenvolvimento e outra para gerência do projeto. Um procedimento de levantamento de métricas voltado para o time de desenvolvimento deve ser leve, onde a ênfase está em uma simples coleta de dados, com breves análises.

Processos guiados por planejamento definem previamente passos e procedimentos para o desenvolvimento de um projeto, onde é assumido que requisitos não mudam, os artefatos e documentação são detalhados e os testes realizados após o desenvolvimento. Do ponto de vista ágil, os processos são mais empíricos, baseados nas experiências e observações, pois mesmo com poucos procedimentos previamente definidos, muitos são construídos à medida que o projeto é desenvolvido, há muito espaço para ser flexível, e frequentemente surgem oportunidades para reavaliar, replanejar e refatorar. Métricas para um processo ágil devem refletir essa natureza empírica.

4.2 Planejamento

Com o propósito de melhorar a qualidade funcional e estrutural do produto e a qualidade do processo de desenvolvimento foi realizado um estudo de caso para comparar a utilização de artefatos que modelam o comportamento do sistema na

especificação de requisitos. A análise compara as abordagens em termos de qualidade funcional e estrutural do produto e qualidade do processo de desenvolvimento quando não é utilizado um artefato que define comportamento do sistema, quando é utilizado o artefato caso de uso, e quando é utilizado o artefato diagrama de atividades.

A medição foi realizada em apenas um time de apenas um projeto da organização, conforme citado no 1.4 Escopo deste trabalho. O time é composto de sete membros, sendo cinco integrantes na equipe de desenvolvimento, um *Scrum Master* e um *Product Owner (*Figura 31*)*, no qual todos já estão familiarizados com os artefatos e ferramentas do processo atual da organização e da medição, descartando a necessidade de envolver qualquer treinamento no estudo de caso. Apenas um repasse sobre o planejamento da coleta de dados foi feito para deixá-los cientes de que em momentos específicos das *sprints* seriam coletadas algumas medidas à respeito do processo e dos resultados gerados na semana.

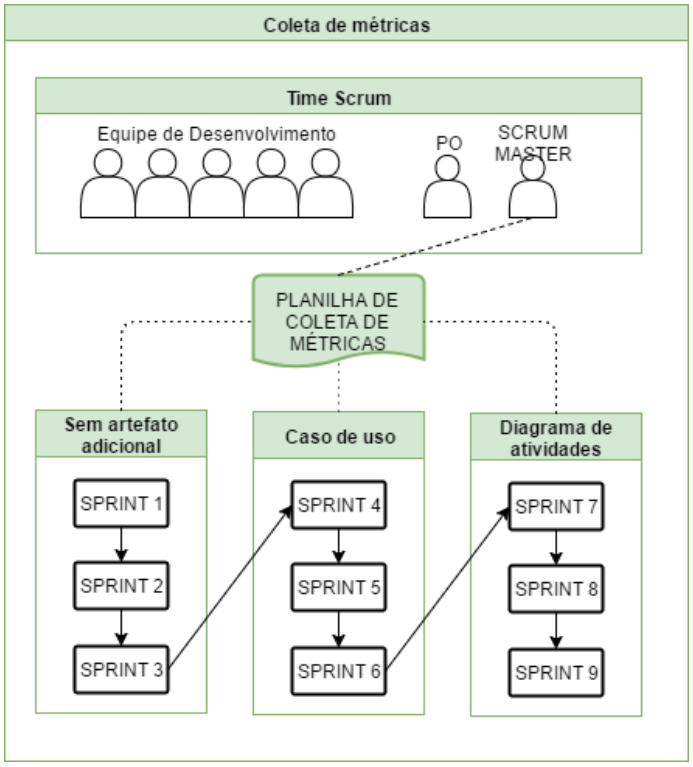


Figura 31 - Planejamento da coleta de métricas

Para a realização do estudo de caso foi construído um modelo de medição que no nível conceitual possui um objetivo que descreve o objeto do estudo relacionado ao propósito que se busca alcançar. No nível operacional define uma questão para cada frente do objetivo (qualidade funcional, estrutural e do processo de desenvolvimento) para caracterizar como a realização deste será alcançada e também qual o contexto foi levado em consideração para tal questionamento. Por fim, no nível quantitativo, são descritos os conjuntos de dados que serão coletados e associados à cada questão para que esta possa ser respondida quantitativamente, juntamente com uma descrição de como estas medidas serão interpretadas e quais alternativas se espera que aconteça.

4.3 Construção do modelo: Objetivo, Questões e Métricas

Em um ambiente onde muitas decisões são delegadas ao time de desenvolvimento, muitas informações relacionadas à medição do software são necessárias para dar assistência em suas atividades diárias. No nível operacional, as informações financeiras não são relevantes para o planejamento e progresso do time, então se faz necessário definir objetivos e métricas de medição que analisa como o desenvolvimento do software contribui para a criação de valor para o negócio, principal foco dos princípios ágeis.

Agilidade é definida por Highsmith (2011) como

> "... a habilidade de criar e responder à mudança para prosperar em um ambiente de negócios turbulento. Significa que temos que construir e sustentar uma máquina de entrega de valor contínuo, não apenas uma que libera diversas funcionalidades na primeira entrega e rapidamente enfraquece nesse quesito. A melhor expressão de agilidade da perspectiva de software é entrega e desenvolvimento contínuo."

A qualidade do software é um importante contribuinte para a entrega de valor ao negócio. As consequências da baixa qualidade de software com o passar do tempo acaba se tornando insustentável para o negócio (CHAPPELL, 201?). De acordo com Chappell (201?), a baixa qualidade de software leva a perdas financeiras por perdas de negociações, custo de manutenção, perda de clientes, ações na justiça e eventualmente, perda de credibilidade.

Segundo Chappel (201?), não existe uma única maneira correta de se pensar sobre qualidade de software, entretanto, fica mais fácil agrupando diversos componentes em três grupos distintos, qualidade funcional, estrutural e do processo (Figura 32).

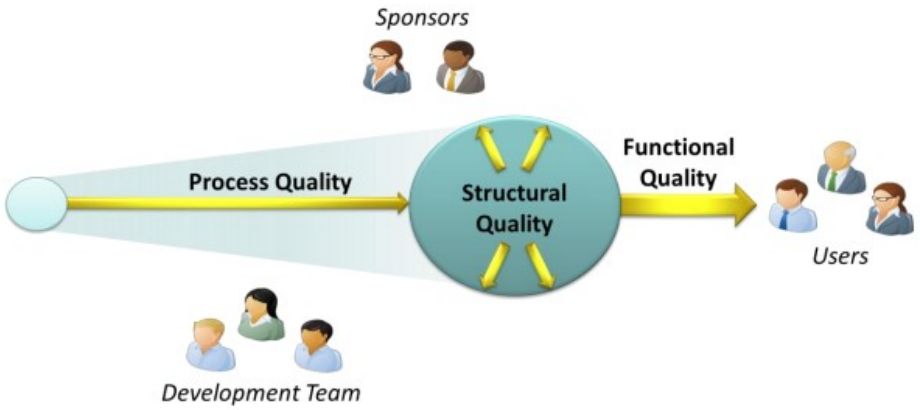


Figura 32 – Grupo de fatores que influenciam a qualidade do software (CHAPPELL, 201?).

A qualidade funcional lida com os aspectos importantes para os *stakeholders*, que avaliam o valor do software através dos atributos de qualidade funcional, por exemplo, se as funcionalidades presentes atendem à sua necessidade de negócio. A qualidade do processo lida com o estado do processo de desenvolvimento. Estar de acordo com os prazos e orçamentos planejados é responsabilidade do processo de desenvolvimento. Qualidade estrutural é uma visão de qualidade interna do produto como testabilidade e manutenibilidade do código fonte. Mesmo sendo uma visão interna, ela é refletida externamente através do produto (Tabela 9).

Tabela 9 – Exemplo de fatores para cada grupo que influencia na qualidade do software.

	Fatores
Qualidade funcional	1. Atender aos requisitos especificados; 2. Criar software com poucos defeitos; 3. Boa performance; 4. Fácil de aprender e usar.
Qualidade estrutural	1. Testabilidade do código; 2. Manutenibilidade do código; 3. Legibilidade do código; 4. Código eficiente; 5. Segurança.
Qualidade do processo	1. Entregar no prazo; 2. Permanecer dentro do orçamento; 3. Processo pode ser repetido e continuar obtendo bons resultados (entregar um software de boa qualidade)

Fonte tabela 9 - (CHAPPELL, 201?)

As características de qualidade funcional e estrutural definidas por Chappel também podem ser encontradas na ISO/IEC 25010 (Figura 33). Qualidade funcional e qualidade estrutural fazem referência à adequação funcional (*functional suitability*) e manutenibilidade (*maintainability*) respectivamente (OBERSCHEVEN, 2013). A adequação funcional diz respeito ao "nível ao qual o produto ou sistema provê funções que atendem às necessidades descritas e implícitas quando utilizado sob condições específicas" (ISO, 2011), e manutenibilidade é definida como o "nível de eficácia e eficiência com o qual um produto ou sistema pode ser modificado por seus mantenedores" (ISO, 2011).

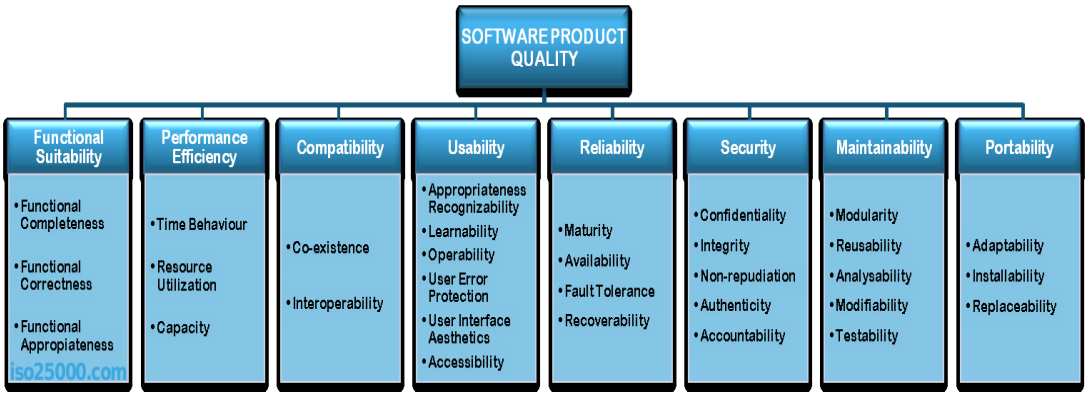


Figura 33 - As oito características de qualidade do produto segundo ISO 25010 (ISO25000 SOFTWARE PRODUCT QUALITY)

Vale ressaltar que outras características de qualidade descritas na ISO 25010 também são importantes, porém as características escolhidas tem mais relevância para avaliar se o desenvolvimento do software está produzindo um produto que agrade os usuários finais e crie uma plataforma viável para receber melhorias dentro dos seus limites de escopo, tempo e custo (HIGHSMITH, 2010).

Os objetivos descritos a seguir serão norteados tendo em mente que o objeto de estudo é a especificação, que o principal objetivo dos processos ágeis é a entrega de valor de negócio e que a qualidade do software é importante para isso.

4.3.1 Objetivos

4.3.1.1 Objetivo Global

Definir o impacto da presença de artefatos que definem comportamento do sistema no processo de desenvolvimento do ponto de vista das três dimensões da qualidade (funcional, estrutural e do processo) definidas por Chappel (201?).

4.3.1.2 Objetivo da Medição

Tendo como base o modelo de especificação de requisitos, o processo de desenvolvimento utilizado pela organização e os fatores de qualidade de software, avaliar:

1. A qualidade funcional observada ao final do desenvolvimento dos requisitos;
2. A qualidade estrutural observada ao final do desenvolvimento dos requisitos;
3. A qualidade do processo observada ao final do desenvolvimento dos requisitos.

4.3.2 Questões e Métricas

Para avaliar a influência da especificação na qualidade do software visando o objetivo de melhorar o processo de desenvolvimento e assim a entrega de valor de negócio foram elaboradas questões visando os três aspectos de qualidade do software citados por Chappell (201?).

Todas as medidas são coletadas pelo autor do trabalho.

4.3.2.1 Questão derivada da qualidade funcional (questão 1)

"A necessidade que motivou o desenvolvimento da funcionalidade foi completamente atendida?"

- **Contexto**

Para que exista a demanda de solicitação de uma nova funcionalidade, deve existir um motivador por parte dos stakeholders que trouxe essa necessidade. Para que seja vista entrega de valor, essa necessidade tem que ser entendida e atendida por essa nova funcionalidade, caso contrário, pode gerar insatisfação. Essa necessidade é exposta ao time de desenvolvimento através dos artefatos de requisitos da especificação.

- **Métricas**:
  - **Percentual de requisitos não atendidos**;

A taxa de requisitos não atendidos será um percentual calculado a partir da quantidade de requisitos levantados pela especificação que não foram atendidos pela implementação dividido pela quantidade total de requisitos da sprint. Quanto maior a taxa, menor foi a quantidade de critérios levantados pela especificação que não foram implementados, diminuindo assim a adequação da nova funcionalidade à necessidade do cliente.

$$\%\ requisitos\ não\ atendidos = \frac{\#\ requisitos\ não\ atendidos}{\#\ total\ de\ requisitos} \times 100$$

A quantidade total de requisitos será medida no início da sprint, ao final da reunião de planejamento (sprint planning) somando-se todos os critérios levantados pelo time durante a reunião de grooming. A quantidade de requisitos não atendidos

será medido durante o processo de avaliação da implementação somando-se todos os critérios que não foram atendidos pela implementação.

- **Percentual de requisitos atendidos**;

A taxa de requisitos atendidos será um percentual calculado dividindo o número de requisitos atendidos pelas histórias implementadas na sprint, pelo total de requisitos alocados para implementação nesta. Será o inverso da métrica anterior, taxa de requisitos não atendidos.

$$\%\, requisitos\ atendidos\ = \frac{\#\ requisitos\ atendidos}{\#\ total\ de\ requisitos} \times 100$$

A quantidade total de requisitos será capturada no início da sprint, ao final da reunião de planejamento (sprint planning) somando-se todos os critérios levantados pelo time durante a reunião de *grooming*. A quantidade de requisitos atendidos será capturada durante o processo de avaliação da implementação pelo PO somando-se todos os critérios que de fato foram atendidos pela implementação.

- **Percentual de requisitos levantados após início da implementação**;

A taxa de requisitos levantados após início da implementação será um percentual calculado pela divisão da quantidade de requisitos levantados após já ter sido dados início a implementação pela quantidade total de requisitos alocados para implementação na sprint. Um novo requisito levantado pode significar que deveria existir algum outro critério essencial para atender por completo as necessidades do usuário e que não seria implementado se não fosse percebido.

$$\%\, novos\ requisitos\ = \frac{\#\ requisitos\ levantados\ após\ início\ da\ implementação}{\#\ requisitos\ total} \times 100$$

A quantidade total de requisitos será capturada no início da sprint, ao final da reunião de planejamento (sprint planning) somando-se todos os critérios levantados pelo time durante a reunião de *grooming*. A quantidade de requisitos levantados após início da implementação será capturada durante o processo de avaliação da implementação somando-se todos os critérios levantados após início da implementação.

- **Quantidade de defeitos encontrados por testes exploratórios**.

A quantidade de defeitos encontrados por testes exploratórios será a quantidade de defeitos que foram descobertos após ser sinalizado pelo implementador que a implementação está concluída. Serão somados todos os erros encontrados pelas tarefas de testar implementação e validação do P.O.

$$\#\,defeitos = \#\,defeitos\ no\ teste\ do\ implemetador + \#\,defeitos\ na\ validação\ do\ PO$$

A quantidade total de defeitos será capturada durante o processo de implementação, após ser sinalizado pelo implementador que a implementação está concluída e então é executado o "buddy test" e após a tarefa de validação da história pelo P.O.

- **Interpretação das métricas**

Quanto mais artefatos de requisitos forem apresentados pela especificação, menores são as chances de que requisitos funcionais não sejam implementados, menores são as chances de que requisitos elicitados não sejam incluídos na especificação, e melhores são as chances de que não apareçam defeitos ou que sejam encontrados previamente para que possam ser corrigidos antes da entrega da funcionalidade.

- **Alternativas**:

❖ 1: as taxas não serão alteradas, e a necessidade do cliente continuará sendo atendida no mesmo nível, mesmo com a utilização de artefatos que definem o comportamento de sistema no modelo de especificação;

❖ 2: a taxa de requisitos atendidos irá aumentar, a taxa de requisitos não atendidos ou levantados após o início da implementação irá diminuir e menos defeitos serão encontrados, melhorando a qualidade funcional após a adoção dos artefatos que definem o comportamento do sistema no modelo de especificação;

❖ 3: a taxa de requisitos atendidos irá diminuir, a taxa de requisitos não atendidos ou levantados após o início da implementação irá aumentar e mais defeitos serão encontrados, piorando a qualidade funcional após a utilização dos artefatos que definem o comportamento do sistema no modelo de especificação.

4.3.2.2 Questão derivada da qualidade estrutural (questão 2)

"Qual a qualidade do código gerado para desenvolver a funcionalidade?"

- **Contexto**

Como o processo de desenvolvimento das histórias de usuário permite que a mesma história de usuário seja implementada por mais de um desenvolvedor ao mesmo tempo, é importante que o compreensão do que deve ser feito esteja claro e de forma alinhada entre todos os desenvolvedores do time, caso de requisitos que geram dúvidas ou estejam ambíguos podem refletir de maneira negativa no código gerado.

- **Métricas**:

❖ **Quantidade de falhas encontradas no processo de revisão de código;**

A quantidade de falhas encontradas na revisão será a quantidade de defeitos na qualidade do código que foram descobertos pelo processo de revisão de código.

$$\#\ falhas\ = \#\ falhas\ encontradas\ na\ revisão\ do\ código$$

A quantidade total de defeitos será capturada durante o processo de implementação, após ser sinalizado pelo implementador que a implementação está concluída e então é executada a revisão do código.

❖ **Qualidade do código gerado pela implementação;**

A qualidade do código gerado pela implementação diz respeito ao esforço feito para assimilar o código gerado pelo desenvolvedor aos requisitos que devem ser implementados. Essa medida foi dividida em uma escala ordinal: bom, regular e ruim. Sendo “bom” uma escala que significa que o código está legível e de fácil compreensão, “regular” significa que o código não está simples, porém de acordo com os requisitos e “ruim” significa que o código está muito complexo e difícil de fazer uma ligação do que está escrito no código com os requisitos da especificação.

A dificuldade de entendimento do código será capturada durante o processo de implementação, após ser sinalizado pelo implementador que a implementação está concluída e então é executada a revisão do código.

❖ **Quantidade de testes automatizados implementados.**

A quantidade de testes automatizados implementados será a quantidade total de novos testes implementados para cobrir os novos cenários introduzidos pela implementação.

$$\#\ testes = \#\ novos\ testes\ automatizados\ implementados$$

A quantidade total de novos de testes automatizados implementados será coletada ao final da sprint, após todas as histórias estarem concluídas.

- **Interpretação**

Uma especificação mais completa torna mais fácil a interpretação do “como” deve ser desenvolvida a funcionalidade, gerando códigos mais desacoplados e coesos, sendo assim mais legíveis, mais testáveis e de manutenção menos custosa. Códigos limpos são mais fáceis de gerar testes de unidade que são automatizados pelo sistema de integração contínua do desenvolvimento, sendo assim, é maior a garantia de que se eventualmente sofrerem manutenção, não tenham seu comportamento alterado ou acusem falha de testes quando houver um erro.

- **Alternativas**:

❖ 1: a quantidade de erros encontrados pela revisão não será alterada assim como a dificuldade para entendimento do código gerado e produção de testes unitários também, mantendo a qualidade estrutural mesmo utilizando os artefatos que definem o comportamento no modelo es especificação;

❖ 2: a quantidade de erros encontrados pela revisão vai diminuir, o esforço para entendimento do código gerado será menor e mais testes automatizados serão implementados, melhorando a qualidade estrutural após introdução dos artefatos que definem o comportamento do sistema no modelo atual de especificação;

❖ 3: a quantidade de erros encontrados pela revisão vai aumentar, o esforço para entendimento do código gerado será maior e menos testes automatizados serão implementados, piorando a qualidade estrutural após a aplicação dos artefatos que definem o comportamento do sistema no modelo atual de especificação.

4.3.2.3 Questão derivada da qualidade do processo (questão 3)

"Qual a precisão das estimativas levantadas no planejamento?"

- **Contexto**

Durante o processo de planejamento da sprint, as histórias são apresentadas ao time de desenvolvimento, o qual na segunda etapa deste processo, discute e sugere uma estimativa do esforço para o desenvolvimento em pontos de história. Quanto mais artefatos de requisitos forem apresentados ao time sobre o que deve ser desenvolvido, mais precisa será a estimativa do esforço necessário para implementar a funcionalidade.

- **Métricas**:

❖ **Taxa de histórias de usuário com valor de pontos de história recalibrados após implementação;**

Taxa de histórias de usuário com valor de pontos de história recalibrados após implementação será um percentual calculado pela quantidade de histórias de usuário que não foram precisos na estimativa em pontos de história acordado pelo time durante a reunião de *grooming* dividido pelo total de histórias de usuário alocadas para a sprint.

$$\%\ histórias\ recalibradas\ = \frac{\#\ histórias\ com\ valor\ de\ story\ points\ alterados}{\#\ total\ de\ histórias\ alocadas\ para\ sprint} \times 100$$

A taxa de histórias de usuário com valor de pontos de história recalibrados será coletada ao final da sprint, após todas as histórias estarem concluídas.

❖ **Taxa de alteração no valor do ponto de história das histórias alteradas;**

Taxa de alteração no valor do ponto de história alterado após implementação será um percentual que representa a diferença na precisão das estimativas. O cálculo feito é soma da quantidade de pontos de histórias de usuário após serem recalibrados ao final da sprint dividido pelo total de histórias de usuário alocadas para a sprint acordados na reunião de planejamento (sprint planning) subtraído de 1.

$$\%\ de\ alteração\ = \left(\frac{\#\ pontos\ histórias\ após\ recalibrar}{\#\ pontos\ história\ antes\ de\ recalibrar} - 1\right) \times 100$$

A taxa de histórias de usuário com valor de pontos de história recalibrados será coletada ao final da sprint, após todas as histórias estarem concluídas.

❖ **Esforço para especificar os requisitos.**

O esforço para especificar os requisitos será medido em horas e será a quantidade de pessoas alocadas para construir a especificação multiplicado pela quantidade de horas gastas por cada uma. Como a especificação será elaborada em dois momentos distintos quando for dado início à análise dos artefatos de caso de uso e diagrama de atividades, essa medição também será realizada em dois momentos distintos, sendo o primeiro momento durante o *grooming*, onde todo o time elabora a especificação em conjunto, e outro momento sendo o esforço de um analista elaborando o artefato naquela sprint.

$$esforço\ = (\#pessoas\ no\ grooming\ \times horas\ da\ cerimônia) + (horas\ para\ elaboração\ do\ artefato\ em\ análise)$$

O esforço para especificar requisitos será coletada após a conclusão da elaboração da especificação.

▪ **Interpretação**

Uma alta quantidade de histórias com valor de pontos alterados, e um alto percentual neste valor de alteração, indica que a visão do que precisa ser desenvolvido ainda não está ideal.

Um dos fatores que influenciam na qualidade do processo é o prazo de entrega, e um aumento muito significativo no esforço pode compromete-lo. Se os benefícios que esse esforço extra proporciona não compensam a diferença introduzida no processo, ele não vale a pena ser mantido.

- **Alternativas**:

❖ 1: a taxa de histórias com valor de pontos recalibrados após implementação, o percentual de alteração e o esforço para especificar não serão alterados, mantendo inalterada a qualidade do processo mesmo utilizando os artefatos que definem comportamento do sistema no modelo de especificação;

❖ 2: a taxa de histórias com valor de pontos recalibrados após implementação, o percentual de alteração e o esforço para especificar serão menores, melhorando a qualidade do processo após introdução dos artefatos que modelam o comportamento do sistema no modelo atual de especificação;

❖ 3: a taxa de histórias com valor de pontos recalibrados após implementação, o percentual de alteração e o esforço para especificar serão maiores, piorando a qualidade do processo após a aplicação dos artefatos que modelam o comportamento do sistema no modelo atual de especificação.

A imagem a seguir (Figura 34) mostra um resumo do modelo de questões e métricas alinhados ao objetivo da medição que foram elaborados nessa seção.

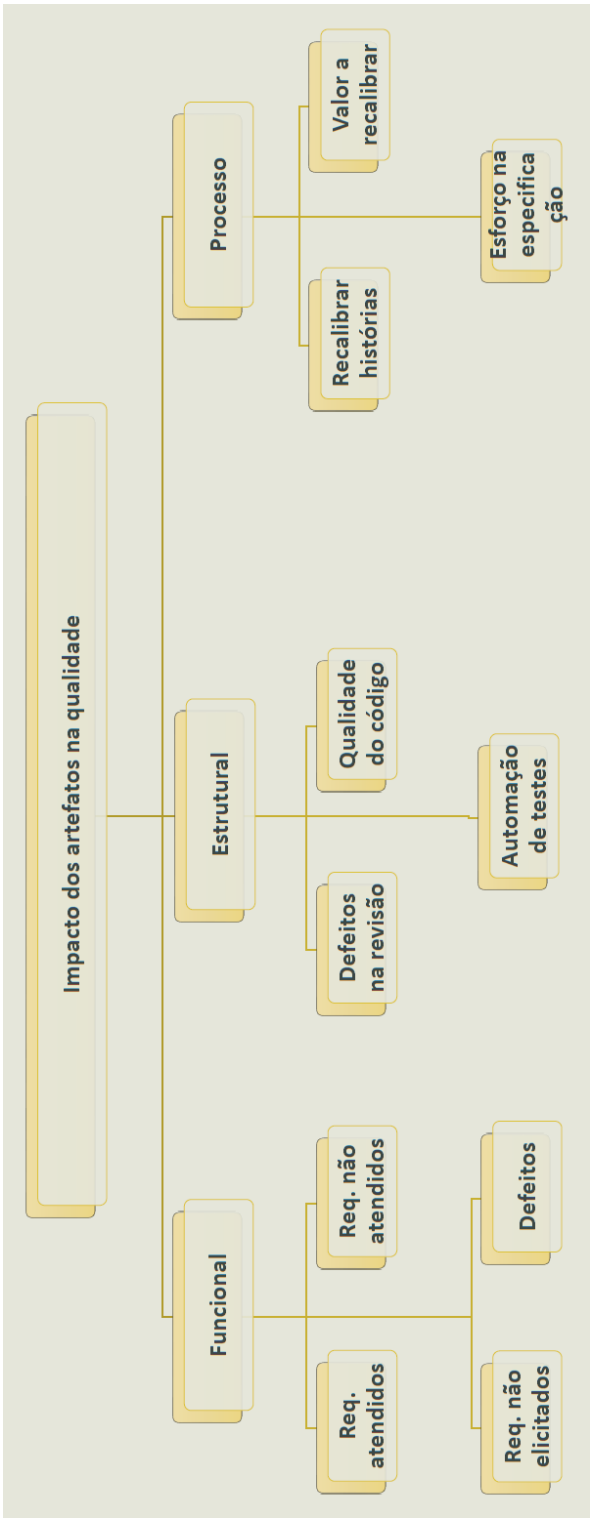


Figura 34 - Resumo das questões e métricas do estudo de caso

4.4 Coleta de Dados

Para executar o estudo as métricas foram coletadas por sprint do time (Figura 35). Foram utilizadas três sprints para cada formato apresentado . Nas três sprints

iniciais nenhuma alteração foi aplicada à especificação para a coleta dos dados. A três sprints seguintes seguiram um formato de especificação que utilizou além dos artefatos do modelo original, também o artefato de caso de uso, e então o mesmo se repetiu para as próximas três sprints porém, substituindo o caso de uso por diagrama de atividades. Os artefatos incluidos na especificação para a realização do estudo de caso foram elaborados pelo autor deste trabalho e alguns podem ser visualizados na seção de 6. Apêndices.

Para a coleta de dados, foram elaboradas planilhas, apresentadas nos 6. Apêndices deste trabalho, baseado no modelo de medição discutido na seção anterior, 4.3 Construção do modelo: Objetivo, Questões e Métricas". Todos os dados foram coletados pelo autor deste trabalho por meio de questionamento aos membros do time ou por ele mesmo em momentos específicos do processo de desenvolvimento onde o dado era gerado.

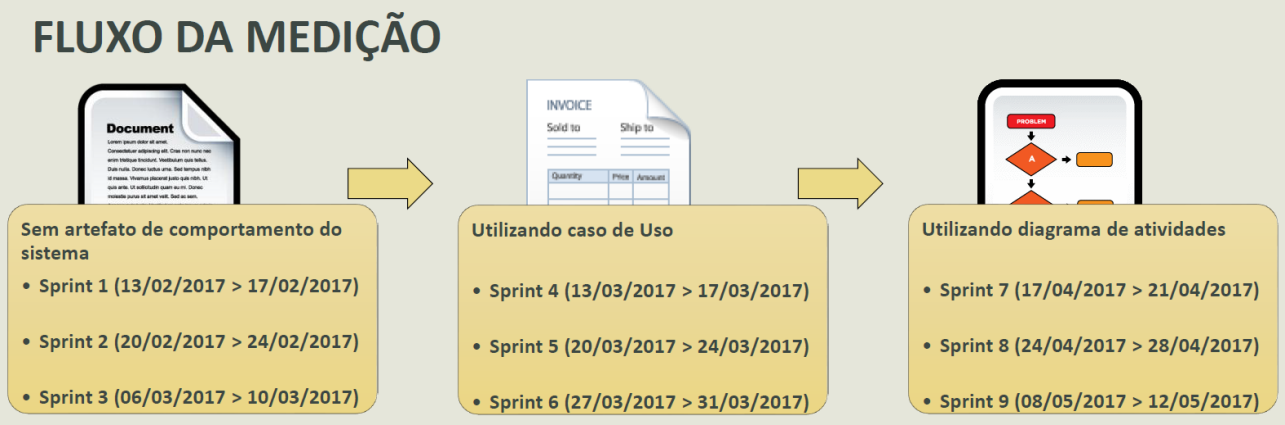


Figura 35 - Fluxo da medição

4.5 Interpretação

Devido ao tempo hábil para realizar a coleta, ao backlog do produto e às priorizações dos itens a serem desenvolvidos, apenas 9 *sprints* foram selecionadas

para a coleta de métricas, deixando assim, a amostra de dados não grande o suficiente para a aplicação de testes estatísticos, portanto, para a interpretação dos dados, foi feita uma análise descritiva das observações retiradas a partir dos dados.

Nesta seção é apresentada a interpretação dos dados obtidos pela etapa de coleta e averiguar as respostas para as questões levantadas no modelo de medição.

4.5.1 Questão 1

“A necessidade que motivou o desenvolvimento da funcionalidade foi completamente atendida?”

Essa questão foi aplicada no modelo com o intuito de avaliar qual formato de apresentação da especificação proveria melhor qualidade funcional ao produto através da medição do número de requisitos que estavam sendo atendidos ou não após a conclusão da implementação, se algum requisito funcional deveria estar na especificação, porém foi esquecido, ou se algum defeito foi encontrado impedindo que a funcionalidade atendesse por completo as necessidades explícitas e implícitas do usuário.

Conforme ilustram os gráficos das métricas coletadas para a primeira questão (Figura 36), 78% das sprints tiveram um número baixo de requisitos (sprints 1, 2, 4, 5, 6, 8, 9) se comparado com as sprints com mais requisitos totais para implementação (sprints 3 e 7), e em 7 das 9 sprints, a taxa de requisitos atendidos foi de 100% (sprints 2, 3, 4, 5, 6, 8, 9), deixando apenas uma sprint que não utilizou artefato adicional na especificação e uma sprint que utilizou diagrama de atividades, com um requisito não atendido. Na sprint sem artefato adicional o requisito não

atendido representa 33% de todos que foram alocados para desenvolvimento na sprint, enquanto que na sprint que utiliza diagrama de atividades, representa 6%.

Para as sprints que apresentam requisitos não atendidos, no caso das amostras coletadas quando não se utiliza um artefato adicional na especificação (sprint 1), não há indícios de que existe uma correspondência entre a taxa de requisitos não atendidos e a quantidade total, pois as outras duas sprints com o mesmo modelo de apresentação da especificação, sprints 2 e 3, possuem respectivamente, quantidades menores (um requisito a menos) e bem maiores (quatorze requisitos a mais) na quantidade total de requisitos, porém com 100% dos requisitos atendidos. No caso das amostras coletadas quando se utiliza diagrama de atividades, os dados apontam para uma interdependência entre a taxa de requisitos não atendidos e a quantidade total já que a única sprint que apresentou requisito não atendido foi aquela com o maior número de requisitos para implementar (sprint 7). As 3 sprints em que se utiliza caso de uso não apresentam nenhuma medição com requisito não atendido e foi o modelo de apresentação da especificação com a menor média no total de requisitos por sprint (média de 4 requisitos por sprint para amostras do grupo com caso de uso, e 7 e 9 para as amostras sem artefato adicional e diagrama de atividades respectivamente) e também foram mais similares, com a menor variabilidade na quantidade total de requisitos para implementar.

A métrica de quantidade de requisitos levantados após o início da implementação apresenta 4 novos requisitos durante a medição, sendo um na sprint 2 quando não se utiliza artefato adicional na especificação – valor que representa 50% da quantidade total de requisitos iniciais alocados para a sprint – dois nas sprints 5 e 6 quando se utiliza caso de uso – valor que representa 50% e 33% do total de

requisitos da sprint respectivamente – e um na sprint 7 quando se utiliza diagrama de atividades – valor que representa 6% do total de requisitos da sprint. Essa métrica só mostrou uma forte ligação com a quantidade total de requisitos e o fato de serem atendidos ou não quando se utiliza diagrama de atividades na especificação, pois foi o único formato de apresentação da especificação onde essa métrica mostrou resultado relevante na sprint com maior quantidade de requisitos para implementar, enquanto para os outros formatos, a medida relevante só apareceu nas sprints com a menor quantidade de requisitos para implementar.

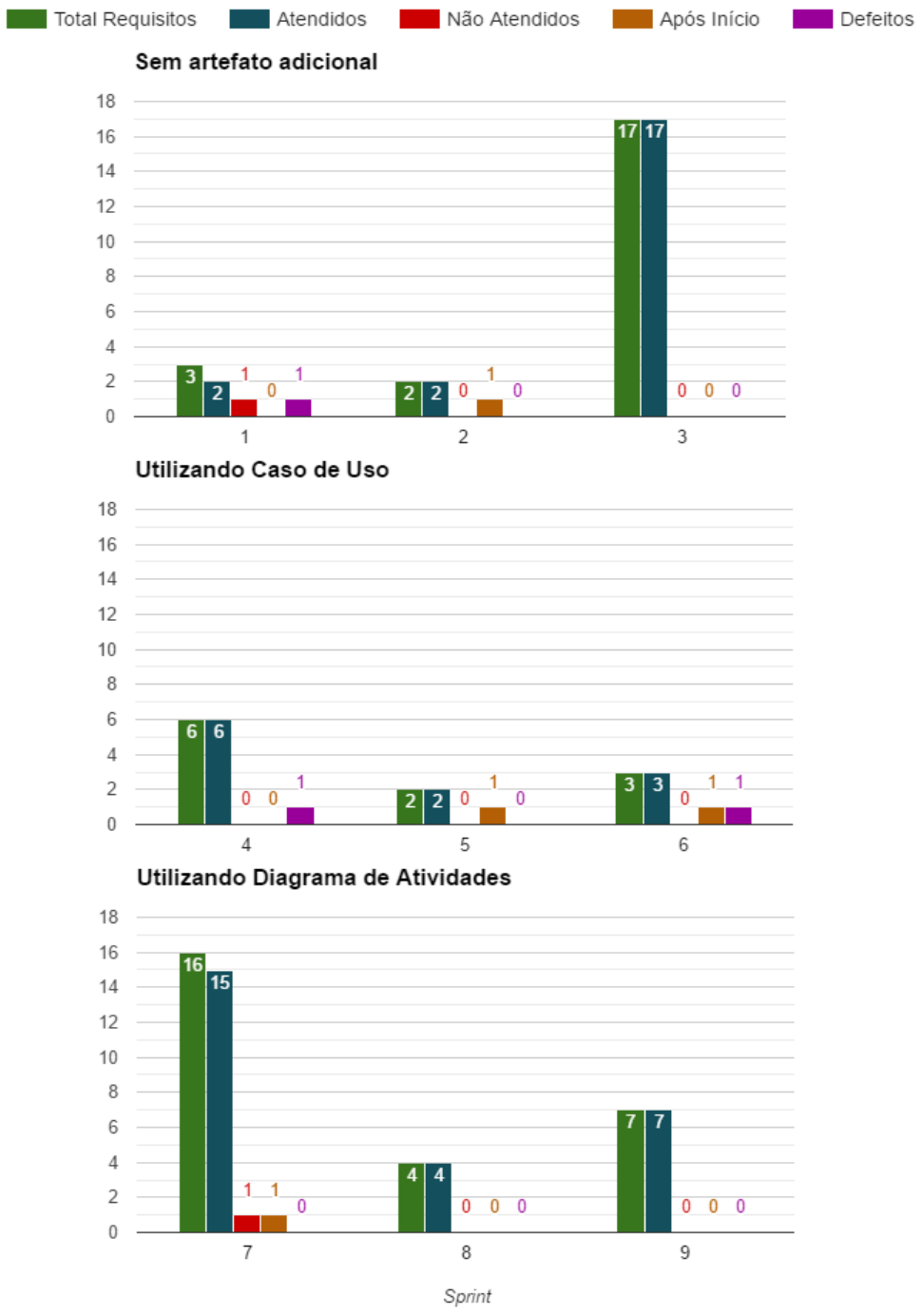


Figura 36 - Métricas da questão 1 (eixo X – Nº da Sprint, eixo Y – Quantidade)

A quantidade de defeitos tem um comportamento parecido com a métrica de quantidade de requisitos não atendidos no grupo que não utiliza artefato adicional no formato de apresentação da especificação. As amostras desse grupo não exibem um indicativo de reciprocidade entre a quantidade total de defeitos e quantidade total de

requisitos, diferente das amostras coletadas quando se utiliza caso de uso, que o fato de somente a sprint com a menor quantidade de requisitos para implementar não apresentar defeitos aponta que existe uma relação. Apenas as sprints que utilizam diagrama de atividades não apresentaram nenhum valor relevante na quantidade de defeitos independente das outras medidas.

Observa-se que as amostras do grupo que não utiliza artefato adicional no formato de apresentação da especificação não exibe uma concomitância entre quase todas as medidas coletadas, com exceção da relação entre a quantidade total de requisitos e a taxa de requisitos atendidos, onde há indicativo de que crescem e diminuem em proporções similares. Isso aponta que esse formato tem pouca influência nos aspectos negativos escolhidos para medição com relação a qualidade funcional do produto. As amostras do grupo que utiliza caso de uso mostraram quantidade nula de requisitos não atendidos, porém foram as que mostraram maior proporção na quantidade de requisitos levantados após o início da implementação e defeitos encontrados por testes exploratórios. Quando é utilizado diagrama de atividades, o resultado mostrou que a quantidade de defeitos encontrados por testes exploratório é nulo e as outras medidas coletadas tem uma relação mais próximas entre si ou seja, quanto maior for uma das medidas, possivelmente maior serão as outras.

4.5.2 Questão 2

“Qual a qualidade do código gerado para desenvolver a funcionalidade?”

Essa questão foi incluída no modelo para averiguar qual dos 3 formatos de apresentação da especificação propicia melhor qualidade estrutural ao produto

através de métricas de quantidade de falhas encontradas na revisão de código, quantidade de testes automatizados implementados e a qualidade do código gerado medido nas escalas: “Bom” para código legível e de fácil compreensão; “Regular” para código não simples, porém de acordo com os requisitos; e “Ruim” para código gerado muito complexo e difícil de assimilar aos requisitos (Figura 37).

A métrica de falhas encontradas na revisão não está sendo apresentada pois todas as medidas coletadas foram nulas visto que o processo de revisão de código só é executado quando não existe programação pareada para implementação dos requisitos, e para todos os requisitos implementados durante a medição, o processo seguido foi através de programação pareada.

A métrica qualitativa sobre a qualidade do código gerado apresentou melhores resultados nas amostras que não utilizam artefato adicional no formato de apresentação da especificação visto que apresenta o maior percentual de avaliações na escala “Bom” (67%), que é a mesma proporção das amostras com caso de uso, porém as avaliações remanescentes estão na escala “Regular”, enquanto que as amostras que utilizam caso de uso, estão na escala “Ruim”. Essa métrica demonstra sofrer uma influência muito grande do código já existente no ambiente onde esse novo código é aplicado. As observações desta métrica descritas na planilha de coleta, Apendice G, mostram que em todos os casos que a avaliação do código foi classificada como “Ruim” ou “Regular” estavam inseridas em um ambiente onde o código era legado.

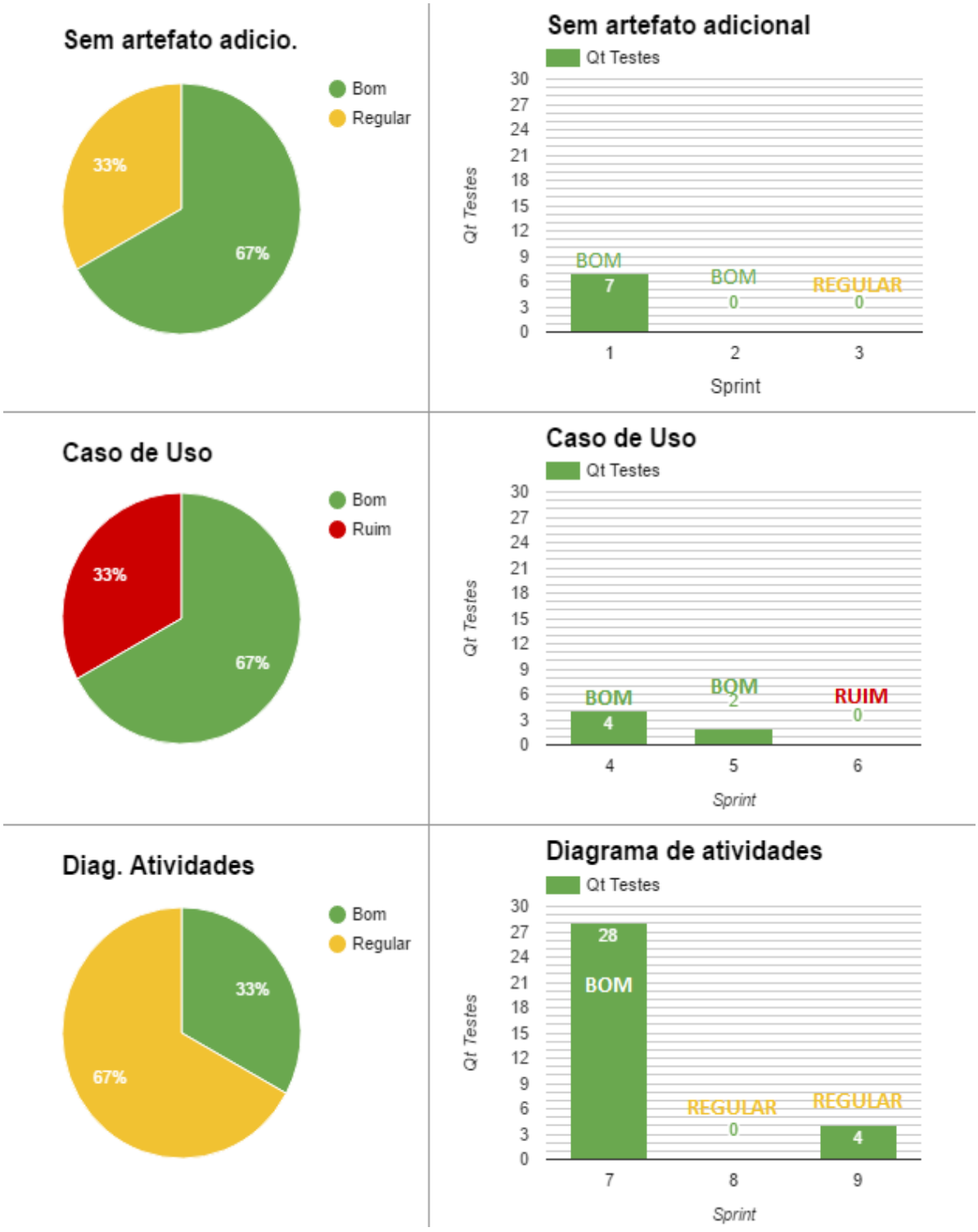


Figura 37 - Métricas da questão 2

A métrica de quantidade de testes automatizados demonstrou receber o mesmo impacto da qualidade do código gerado quando se observa também o código

já existente no ambiente onde os novos requisitos são implementados, visto que a única observação que descreve “Código novo” é a que apresenta uma quantidade bem elevada de testes automatizados se comparado com as outras amostras. Outra observação a se colocar é que essa mesma métrica não parece sofrer impacto sob a quantidade total de requisitos alocados para a sprint quando é visto que as sprints 3 e 7 são as amostras com a maior quantidade de requisitos (Figura 36), entretanto uma tem quantidade nula de testes automatizado e não utilizou artefato adicional na apresentação da especificação, enquanto a outra tem 28 testes automatizado e utilizou diagrama de atividades.

De maneira geral, as métricas retiradas na medição não trouxeram resultados muito conclusivos do ponto de vista do formato de se apresentar a especificação, e sua influência sobre as métricas de qualidade estrutural escolhidas para medição parecem ter pouco impacto.

4.5.3 Questão 3

“Qual a precisão das estimativas levantadas no planejamento?”

Com o intuito de avaliar a qualidade do processo o modelo de medição inclui 3 métricas que visam observar a precisão das estimativas de implementação geradas pelo time de desenvolvimento e o esforço utilizado para elaborar as especificações (Figura 38).

A primeira métrica mede a taxa de quantidade de histórias que tiveram suas estimativas alteradas após a conclusão da implementação. Todos os grupos observados apresentam pelo menos uma sprint na qual existiram histórias que tiveram suas estimativas alteradas, sendo o grupo que não utilizou nenhum artefato

adicional no formato de apresentação da especificação com uma taxa de 57%, a maior de todas se comparado com os 33% do grupo que utiliza caso de uso e 15% o que utiliza diagrama de atividades.

A segunda métrica observa a diferença entre o tamanho das estimativas para implementar as histórias de usuário alocadas para sprint antes da implementação e após, e então calcula a taxa de alteração no valor da história. As amostras do grupo que não utiliza artefato adicional têm duas sprints que apresentam divergência entre o valor da estimativa nos dois momentos distintos da implementação, sendo a sprint 2 com uma taxa de -38% – redução de 13 story points para 8 – e a sprint 3 com uma taxa de -24% – redução de 25 story points para 19. Os outros dois grupos tem apenas uma sprint cada com alteração na estimativa inicial, com uma taxa de 67% de aumento na sprint 6 – de 3 story points para 5 – que utiliza caso de uso, e uma taxa de 15% na sprint 9 – aumento de 13 story points para 15 – que utiliza diagrama de atividades.

Do ponto de vista de esforço para especificar os requisitos, não há indícios de uma relação direta com a quantidade total de requisitos para os 3 grupos observados, pois pouco varia o esforço para diferentes quantidades de requisitos, porém é observável uma interdependência com a estimativa inicial em story points dada pelo time nos grupos que utilizam artefato adicional na especificação. Pode-se interpretar que o grupo que não utiliza um artefato adicional despende um esforço muito menor, principalmente na amostra coletada na sprint 3, em que o total de story points é o mais elevado dentre todas as amostras e o esforço está entre os mais baixos.

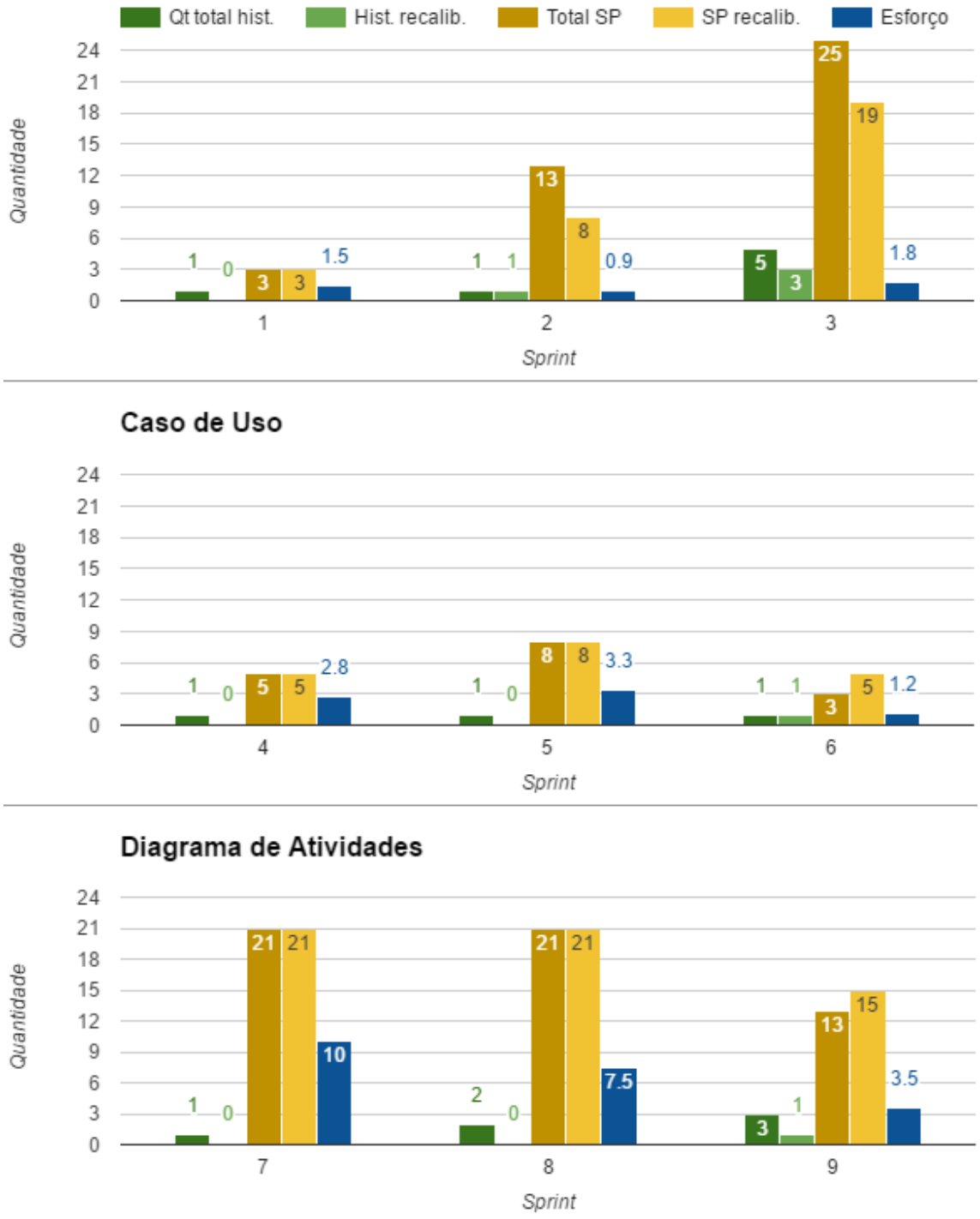


Figura 38 - Métricas da questão 3

Embora exista esse esforço extra para especificar os requisitos utilizando um artefato adicional, a representatividade dele é muito baixa se comparada com o

esforço total coletivo utilizado no processo de *grooming* (Figura 25). Nos dois grupos que utilizam artefato adicional, o esforço extra para especificar o requisito representa apenas 6,7% na amostra com a menor representatividade (sprint 8, com diagrama de atividades) e 16,7% na amostra com maior representatividade (sprint 6, com caso de uso).

O resultado dessas métrica pode ser interpretado de uma forma que indica que a utilização de um artefato adicional diminui o nível de incerteza no processo de estimativa e apresenta resultados com maior grau de precisão no processo de desenvolvimento, e a utilização do diagrama de atividades trouxe benefício visto que o maior esforço não está no momento de especificar o artefato que representa o comportamento do sistema e sim no processo de *grooming*.

4.6 Ameaças a Validade

Para a validação do estudo de caso, são consideradas algumas ameaças que podem afetar o resultado e a validade das conclusões retiradas a partir da interpretação dos dados coletados. São elas:

1. Tempo hábil para desenvolvimento do estudo: devido ao tempo hábil para a coleta de dados, somente 3 sprints por grupo foram selecionadas, deixando a quantidade de amostras pequena para aplicação de testes estatísticos, portanto apenas a análise descritiva foi aplicada;
2. Produto já inserido no mercado: por ser um produto já inserido no mercado, o esforço de implementação do time de desenvolvimento é competido por correção de bugs e evolução do sistema, portanto muitas das amostras coletadas foram de sprints com um backlog de requisitos para implementação

muito pequeno, onde apenas poucas alterações em funcionalidades já existentes eram feitas, e poucas novas funcionalidades surgiram. Dessa forma, não existe uma consistência na quantidade de requisitos e tamanho das alterações realizadas entre as amostras tanto do mesmo grupo, quanto de grupos diferentes;

3. Autor do trabalho também trabalha no time do estudo de caso como Scrum Master: o autor do estudo de caso exerce suas funções profissionais dentro do time selecionado no estudo de caso e foi responsável por elaborar o modelo de medição, realizar a coleta de dados e também elaborar os artefatos adicionais das especificações de requisitos das amostras dos grupos que utilizaram caso de uso e diagrama de atividades. Por já estar inserido dentro do processo de desenvolvimento do estudo de caso diariamente, seu conhecimento implícito pode colaborar com o resultado das conclusões, porém pode afetar seu julgamento durante uma análise descritiva;
4. Tempo de casa de alguns integrantes do time: embora seja um time montado recentemente, dois dos integrantes que o compõe possuem grande conhecimento da camada de negócios do software que desenvolvem por trabalharem na organização por muitos anos. Esse conhecimento pode afetar as métricas coletadas, principalmente àquelas que tratam da qualidade funcional e precisão de estimativas.
5. Quantidade de código antigo: por ser um software que vem evoluindo por muito tempo, muita codificação existente está presente no software por muitos anos, implementado por pessoas que já não trabalham no projeto. Por ser uma grande quantidade, esse fator pode afetar as métricas relacionadas à

qualidade estrutural, visto que grande parte dos requisitos implementados durante todo o estudo de caso trabalhavam com esses códigos antigos.

5. Conclusões

O principal objetivo deste trabalho é identificar o impacto dos artefatos utilizados na especificação de requisito sobre a qualidade do software. Com esse propósito foi construido um modelo de medição baseado no framework GQM com questões e métricas para avaliar esse impacto e feita a aplicação do modelo no contexto de um time de uma organização que segue práticas ágeis. Esta seção sumariza as descobertas interpretadas a partir dos resultados obtidos com a aplicação do modelo de medição nesse contexto e ressaltar aspectos relevantes a serem considerados pelo estudo de caso.

O primeiro impacto avaliado dentro do objetivo do trabalho foi sobre a qualidade funcional do produto e foi trazido através da questão “A necessidade que motivou o desenvolvimento da funcionalidade foi completamente atendida?”. A utilização do caso de uso aponta para um impacto negativo, pois mesmo com sprints mais consistentes e menores, demonstrou uma alta nos aspectos negativos medidos no estudo de caso, com exceção do número de requisitos não atendidos. Já a utilização do diagrama de atividades parece ter causado um impacto positivo, pois embora tenha obtido valores relevantes nos aspectos indesejados, estes só foram encontrados na amostra com maior densidade de requisitos, o que gera uma melhor previsibilidade e transparência do resultado, permitindo tomar ações preventivas para que isso não ocorra novamente, um princípio muito reforçado pelas práticas ágeis do Scrum, também conhecida com *kaizen*[8], prática adotada do “Lean”.

[8] Kaizen significa processo contínuo de melhoria que envolve a alta gerência de uma empresa, pessoal de gestão e todos os funcionários. Requer mudanças relevantes no comportamento e autoridade das pessoas com base na experiência, autoridade do líder. Kaizen baseia-se na suposição de que todos os funcionários possuem habilidades que podem ser usadas de forma melhor (JAKUBIEC e BRODNICK, 2016)

A pergunta do modelo de medição, “Qual a qualidade do código gerado para desenvolver a funcionalidade?” foi incluída para avaliar o impacto sobre a qualidade estrutural do produto. Com relação a essa questão, as medidas coletadas pelo estudo de caso apontam os melhores resultados também para os grupo que não utilizam artefato adicional ou utilizam diagrama de atividades, visto que um grupo gera um código de melhor qualidade e no outro grupo mais testes automatizados foram implementados, entretanto as observações incluídas no documento de coleta deixam um questionamento com relação ao ambiente onde será trabalhada a codificação dos novos requisitos (código legado) e então fica a dúvida se dentro do contexto do estudo de caso existiu de fato um impacto na qualidade estrutural causado pela alteração do formato de apresentação da especificação.

Por fim, a última pergunta do modelo, “Qual a precisão das estimativas levantadas no planejamento?” avalia o impacto do ponto de vista da qualidade do processo observando métricas de precisão nas estimativas e esforço para gerar os artefatos adicionais. Essa métrica demonstrou um resultado mais sólido e que aponta que o impacto mais positivo aparece quando é utilizado diagrama de atividades que trouxe à realidade a alternativa 2 colocada no modelo de medição, que espera que a taxa de alteração na estimativa das histórias de usuário seja menor. Embora exista o esforço adicional para elaborar o artefato, a representatividade deste esforço se comparado ao esforço já despendido pelo modelo tradicional já executado pelo time do estudo de caso é muito baixa.

Vale ressaltar que alguns aspectos do contexto do estudo de caso tem relevância para os resultados obtidos e então devem ser levados em consideração na situação do modelo de medição ser replicado para outros times que se encaixam

em contextos parecidos. O projeto onde trabalha o time escolhido para o estudo de caso tem passado por um processo de muitas transformações em pouco tempo, um fato que também dificultou a realização do estudo de caso. O time não possuia mais de 6 meses de experiência trabalhando em conjunto de forma fixa, entretanto 2 integrantes do time de desenvolvedores tem grande expertise nas regras de negócio por terem tempo de casa, inclusive em outras áreas do projeto como suporte ao usuário e consultoria, o que gera consequências quando se mede precisão nas estimativas e até mesmo a qualidade do código gerada por ser uma métrica subjetiva, pois tem grande dependência do conhecimento e julgamento de quem faz a avaliação.

Outro aspecto relevante que vale citar é que o projeto trabalha com um produto que já está estabelecido no mercado, portanto as prioridades de implementação variam e existe também uma concorrência entre a evolução do produto e manutenção, causando a disparidade na quantidade de histórias de usuário e requisitos alocados para implementar em cada uma das sprints como foi visto pela questão 1 do modelo de medição, que acabou também prejudicando no momento de interpretar os dados gerados, pois o tempo disponível para a realização do estudo de caso impossibilitou que as amostras fossem retiradas em sprints com características parecidas.

Por fim, com os resultados obtidos no estudo de caso, a recomendação que fica estabelecida para o time que participou é a utilização do artefato de diagrama de atividades para especificar histórias de usuários com estimativas de valor alto (acima de 21 story points), pois pela interpretação dos resultados obtidos, foram essas amostras que demonstraram maior impacto positivo, já que as outras tiveram

resultados inconclusivos ou impacto negativo, e continuar medindo e experimentando para se obter mais amostras e inspecionar se os resultados desse esforço permanecem ajudando a criar e evoluir o produto mantendo e melhorando a qualidade da entrega de valor.

5.1 Trabalhos Futuros

Nesta seção é apresentado como pode ser dada continuação ao trabalho deste estudo de caso.

5.1.1 Replicar o modelo de medição em um ambiente controlado

Conforme citado na seção anterior, alguns aspectos do contexto do estudo de caso impediram de tirar algumas conclusões mais sólidas. Uma sugestão de trabalho futuro seria aplicar o modelo em um ambiente controlado onde os aspectos como tempo e replanejamentos constantes não sejam um problema para a coleta de dados e a iteração para a coleta de métricas envolva sprints com características mais parecidas e dessa forma obter resultados mais claros.

5.1.2 Reavaliar métricas do modelo

As métricas do primeiro questionamento do modelo foram medidas com o intuito de avaliar a qualidade funcional do produto, no entanto a pesquisa não extrapola os questionamentos para fora do time da organização e não alcança o usuário final, que é o principal foco que trouxe a necessidade da funcionalidade e deve ser o principal consultor para avaliar a adequação funcional do produto. Uma sugestão para trabalhos futuros seria incluir no modelo de medição algumas métricas que meçam a qualidade do produto do ponto de vista do usuário final.

5.1.3 Utilizar outros artefatos de especificação de requisitos

O estudo de caso foi realizado com o objetivo de avaliar o impacto da especificação de requisitos no processo de desenvolvimento de software, porém, no estudo deste trabalho, somente dois artefatos foram medidos e em grupos distintos. Um trabalho futuro poderia avaliar formas diferentes de apresentar a especificação de requisitos ao time de desenvolvimento que sejam diferentes das experimentadas neste trabalho.

5.1.4 Avaliar a causa raiz dos impactos causados pela alteração na forma de apresentação dos requisitos

Através do estudo de caso realizado por este trabalho foi possível avaliar o impacto que o formato de apresentação da especificação de requisitos causa em um time de desenvolvimento que segue práticas ágeis, porém, não se sabe qual a causa raiz que gerou este impacto. Um trabalho futuro poderia levantar hipóteses de quais foram os fatores essenciais para que esse impacto tenho sido causado.

5. Bibliografia

AMBLER, S. W. Agile Modeling. **agilemodeling.com**, 2001. Disponivel em: <http://agilemodeling.com/>. Acesso em: out. 2016.

AMBLER, S. W. **The Object Primer**. 2nd. ed. [S.l.]: Cambridge University Press, 2001.

ANDA, B.; SJøBERG, D. I. K. Towards an inspection technique for use case models. **SEKE '02 Proceedings of the 14th international conference on Software engineering and knowledge engineering**, Ischia, 15 July 2002. 127-134.

BASILI, V. What's So Hard About Replication of Software Engineering? **What's So Hard About Replication of Software Engineering?**, 2011. Disponivel em: <http://chess.cs.umd.edu/~basili/presentations/RESER%20Keynote.pdf>. Acesso em: 17 Maio 2017.

BASILI, V. et al. **Aligning Organizations Through Measurement:** The GQM+Strategies Approach. [S.l.]: [s.n.], 2014.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. **The Goal Question Metric Approach**. [S.l.]: [s.n.], 1994.

BASILI, V. R.; SELBY, R.; HUTCHENS, D. Experimentation in Software Engineering. **IEEE Transactions on Software Engineering**, v. SE-12, n. 7, p. 733-743, Julho 1986.

BASILI, V.; CALDIERA, G.; ROMBACH, D. Goal Question Metric Paradigm. In: MARCINIAK, J. J. **Encyclopedia of Software Engineering**. [S.l.]: Wiley-Interscience, v. 2, 2001.

BECK, K. et al. Agile Manifesto. **Manifesto for Agile Software Development**, 2001. Disponivel em: <http://agilemanifesto.org/>.

BECK, K.; CUNNINGHAM, W. A Laboratory For Teaching Object-Oriented Thinking. **SIGPLAN Notices**, New Orleans, v. 24, n. 10, p. 1-6, October 1989.

CHAPPELL, D. David Chappell. **David Chappell & Associates**, 201? Disponivel em: <http://www.davidchappell.com/writing/white_papers.php>. Acesso em: 03 nov. 2016.

CMMI. What Is Capability Maturity Model Integration (CMMI)®? **CMMI® Institute**. Disponivel em: <http://cmmiinstitute.com/capability-maturity-model-integration>. Acesso em: 24 Junho 2017.

COCKBURN, A. **Writing Effective Use Cases**. [S.l.]: Addison Wesley, 2001.

COHN, M. **User Stories Applied:** For Agile Software Development. 1st. ed. [S.l.]: Addison-Wesley Professional, 2004.

COHN, M. What Are Story Points? **Mountain Goat Software**, 2016. Disponivel em: <https://www.mountaingoatsoftware.com/blog/what-are-story-points>. Acesso em: 17 Maio 2017.

FLORAC, W. A.; CARLETON, A. D. **Measuring de Software Process**. [S.l.]: Addison-Wesley, 1999.

FLORAC, W.; PARK, R.; CARLETON, A. **Practical Software Measurement:** Measuring for Process Management and Improvement. Pittsburgh: Carnegie Mellon University, 1997.

GUTIÉRREZ, J. J. et al. Visualization of Use Cases through Automatically Generated Activity Diagrams. **Lecture Notes in Computer Science**, Berlin, Heidelberg, 2008. 83-96.

HEIMANN, D.; HENNESSEY, P.; TRIPATHI, A. A Bipartite Empiricallt Oriented Metrics Process fo Agile Sofware Development. **Software Quality Professional (SQP)**, v. 9, 2007.

HIGHSMITH, J. Jim Highsmith. **Beyond Scope, Schedule, and Cost:** The Agile Triangle, 2010. Disponivel em: <http://jimhighsmith.com/beyond-scope-schedule-and-cost-the-agile-triangle/>. Acesso em: 24 Maio 2017.

HIGHSMITH, J. Jim Highsmith. **Velocity is Killing Agility!**, 2011. Disponivel em: <http://jimhighsmith.com/velocity-is-killing-agility/>. Acesso em: October 2016.

INAYAT, I. et al. A systematic literature review on agile requirements engineering practices and challanges. **Computers in Human Behavior**, v. 51, Part B, p. 915 - 929, 2014.

ISO. **Iso/iec 25010:** Systems and software engineering systems and software quality requirements and evaluation (square) system and software quality models. [S.l.]: [s.n.], 2011.

ISO25000 SOFTWARE PRODUCT QUALITY. **ISO/IEC 25010**. Disponivel em: <http://iso25000.com/index.php/en/iso-25000-standards/iso-25010>. Acesso em: 24 Maio 2017.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. [S.l.]: Addison Wesley, 1998.

JAKUBIEC, M.; BRODNICK, E. Kaizen Concept in the Process of a Quality Improvement in the Company. **Przedsiębiorstwo we współczesnej gospodarce-teoria i praktyka**, p. 89-101, 2016.

LEFFINGWELL, D. **Agile Software Requirements**. Boston: Addison-Wesley, 2011.

LEFFINGWELL, D.; WIDRIG, D. **Managing Software Requirements**. 1st. ed. [S.l.]: Addison Wesley, 2000.

MACAULAY, L. A. Requirements Engineering. London: Springer-Verlag, 1996. p. 202.

MORGAN, T. **Business Rules and Information Systems Aligning IT with Business Goals**. Indianapolis: Addison Wesley, 2002.

OBERSCHEVEN, F. M. **Software Quality Assessment in an Agile Environment**. Faculty of Science of Radboud University in Nijmegen. [S.l.]. 2013.

PAETSCH, F.; EBERLEIN, A.; MAURER, F. Requirements Engineering and Agile Software Development. **Proc. 12th IEEE Int'l Workshop Enabling Technologies: Infrastructure for Collaborative Enterprises**, 2003.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Elsevier**, Sweden, p. 1-18, Fevereiro 2015.

POHL, K. **Process-Centered Requirements Engineering**. New York: John Wiley & Sons, 1996.

POHL, K.; RUPP, C. **Requirements engineering fundamentals**. 1st. ed. [S.l.]: Rocky Nook, 2012.

ROCHA, A. R. C. D.; SOUZA, G. D. S.; BARCELLOS, M. P. **Medição de Software e Controle Estatístico de Processos**. Brasília: Ministério da Ciência, Tecnologia e Inovação, 2012.

ROSS, R. G. **Principles of the Business Rule Approach**. 1st. ed. [S.l.]: Addison-Wesley Professional, 2003.

SCRUM ALLIANCE®. Scrum Alliance. **Scrum Alliance**, 201? Disponivel em: <https://www.scrumalliance.org/>. Acesso em: out. 2016.

SOFTEX. MPS.BR|Softex. **SOFTEX**. Disponivel em: <http://www.softex.br/mpsbr/>. Acesso em: 24 Junho 2017.

SOLINGEN, R.; BERGHOUT, E. **The Goal/Question/Metric Method:** a practical guide for quality improvement of software development. [S.l.]: David Hatter, 1999.

SOMMERVILLE, I. **Software Engineering**. 9th. ed. [S.l.]: Pearson, 2010.

SUTHERLAND, J.; SCHWABER, K. **The Scrum Guide:** The Definitive Guide to Scrum. [S.l.]: Scrum.org and ScrumInc, 2013.

THOMAS, M. IT Projects Sink or Swim. **British Computer Society Review**, 2001.

TIWARI, S.; GUPTA, A. A systematic literature review of use case specifications research. **Journal Information and Software Technology**, Butterworth-Heinemann Newton, November 2015. 128-158.

TRAVASSOS, G. H. **Introdução à Engenharia de Software Experimental**. COPPE/UFRJ. Rio de Janeiro, p. 52. 2002.

WAKE, B. XP123. **XP123 Exploring Extreme Programming**, 2003. Disponivel em: <http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>.

WANG, L. Measuring and Improving Software Process in China. **Proceedings of International Symposium on Empirical Software Engineering**, 2005. 183-192.

YOURDON, E. **Modern Structured Analysis**. 1st. ed. [S.l.]: Prentice Hall, 1988.

6. Apêndices

Apêndice A – Planila de coleta de métricas resumo das questões e métricas

QUESTÃO DERIVADA DA QUALIDADE FUNCIONAL
A necessidade que motivou o desenvolvimento da funcionalidade foi completamente atendida?
Contexto: Para que exista a demanda de solicitação de uma nova funcionalidade, deve existir um motivador por parte dos stakeholders que trouxe essa necessidade. Para que seja vista entrega de valor, essa necessidade tem que ser entendida e atendida por essa nova funcionalidade, caso contrário, pode gerar insatisfação. Essa necessidade é exposta ao time de desenvolvimento através dos artefatos de requisitos da especificação.
Interpretação: Quanto mais clara e objetiva for a especificação, menores são as chances de que requisitos funcionais não sejam implementados, menores são as chances de que requisitos elicitados não sejam incluídos na especificação, e melhores são as chances de que não apareçam defeitos ou que sejam encontrados previamente para que possam ser corrigidos antes da entrega da funcionalidade.
Métrica 1 (Q1M1) - Taxa de requisitos não atendidos
Métrica 2 (Q1M2) - Taxa de requisitos atendidos
Métrica 3 (Q1M3) - Taxa de requisitos levantados após início da implementação
Métrica 4 (Q1M4)- Quantidade de defeitos encontrados por testes exploratórios

QUESTÃO DERIVADA DA QUALIDADE ESTRUTURAL
Qual a qualidade do código gerado para desenvolver a funcionalidade?
Contexto: Como o processo de desenvolvimento das histórias de usuário permite que a mesma história de usuário seja implementada por mais de um desenvolvedor ao mesmo tempo, é importante que o compreensão do que deve ser feito esteja claro e de forma alinhada entre todos os desenvolvedores do time, caso de requisitos que geram dúvidas ou estejam ambíguos podem refletir de maneira negativa no código gerado.
Interpretação: Uma especificação mais completa torna mais fácil a interpretação do "como" deve ser desenvolvida a funcionalidade, gerando códigos mais desacoplados e coesos, sendo assim mais legíveis, mais testáveis e de manutenção menos custosa. Códigos limpos são mais fáceis de gerar testes de unidade que são automatizados pelo sistema de integração contínua do desenvolvimento, sendo assim, é maior a garantia de que se eventualmente sofrerem manutenção, não tenham seu comportamento alterado ou acusem falha de testes quando houver um erro.
Métrica 1 (Q2M1) - Quantidade de falhas encontradas na revisão
Métrica 2 (Q2M2) - Dificuldade de entendimento do código
Métrica 3 (Q2M3) - Quantidade de testes automatizados implementados

QUESTÃO DERIVADA DA QUALIDADE DO PROCESSO
Qual a assertividade das estimativas levantadas no planejamento?
Contexto: Durante a fase de planejamento da sprint, as histórias são apresentadas ao time de desenvolvimento, o qual na segunda etapa desta fase, discute e sugere uma estimativa do esforço para o desenvolvimento em pontos de história. Quanto melhor for o entendimento do time sobre o que deve ser desenvolvido, mais fácil será estimar o esforço necessário para implementar a funcionalidade.
Interpretação: Uma alta quantidade de histórias com valor de pontos alterados, e um alto percentual neste valor de alteração, indica que a visão do que precisa ser desenvolvido ainda não está ideal. Um dos fatores que influenciam na qualidade do processo é o prazo de entrega, e um aumento muito significativo no esforço pode compromete-lo. Se os benefícios que esse esforço extra proporciona não compensam a diferença introduzida no processo, ele não vale a pena ser mantido.
Métrica 1 (Q3M1) - Taxa de histórias de usuário com valor de pontos de história recalibrados após implementação
Métrica 2 (Q3M2) - Percentual de alteração no valor do ponto de história das histórias alteradas
Métrica 3 (Q3M3) - Esforço para especificar requisitos

Apendice A

Apêndice B – Planilha de coleta de métricas questão 1 métrica 1

Q1M1 - Taxa de requisitos não atendidos

A taxa de requisitos não atendidos será a quantidade de requisitos levantados pela especificação que não foram atendidos pela implementação em comparação com a quantidade total de requisitos.
Taxa = quantidade de requisitos não atendidos / quantidade total de requisitos.

A quantidade total de requisitos será capturada no início da sprint, ao final da reunião de planejamento (Sprint Planning) somando-se todos os critérios levantados pelo time durante a reunião de grooming. A quantidade de requisitos não atendidos será capturada durante a fase de validação da implementação somando-se todos os critérios que não foram atendidos pela implementação.
Todas as coletas serão realizadas pelo autor do trabalho.

Sprint	Período		Total de requisitos			Requisitos não atendidos			Taxa de requisitos não atendidos	Observações
	Início	Fim	Quantidade	Coletado por	Data coleta	Quantidade	Coletado por	Data coleta		
1	13/02/17	17/02/17	3	Guilherme	13/02/17	1	Guilherme	15/02/17	33%	Colaborador da sessão no workflow
2	20/02/17	24/02/17	2	Guilherme	20/02/17	0	Guilherme	24/02/17	0%	Integração contrato via web service / CRM Impacto
3	06/03/17	10/03/17	17	Guilherme	06/03/17	0	Guilherme	10/03/17	0%	Repactuação contrato migrado e notas promissórias
4	13/03/17	17/03/17	6	Guilherme	13/03/17	0	Guilherme	14/03/17	0%	Bloqueio de alteração do analista responsável
5	20/03/17	24/03/17	2	Guilherme	20/03/17	0	Guilherme	22/03/17	0%	Pasta incompleta no workflow
6	27/03/17	31/03/17	3	Guilherme	27/03/17	0	Guilherme	31/03/17	0%	Campos obrigatórios do cadastro de prospect
7	17/04/17	21/04/17	16	Guilherme	17/04/17	1	Guilherme	21/04/17	6%	Aditivo Contratual
8	24/04/17	28/04/17	4	Guilherme	24/04/17	0	Guilherme	26/04/17	0%	Build In
9	08/05/17	12/05/17	7	Guilherme	08/05/17	0	Guilherme	10/04/17	0%	Cálculo acrescimo e empreend rel. de interesse

Apêndice C – Planilha de coleta de métricas questão 1 métrica 2

Q1M2 - Taxa de requisitos atendidos

A taxa de requisitos atendidos será a quantidade de requisitos levantados pela especificação que foram atendidos pela implementação em comparação com a quantidade total de requisitos.
Taxa = quantidade de requisitos atendidos / quantidade total de requisitos.

A quantidade total de requisitos será capturada no início da sprint, ao final da reunião de planejamento (Sprint Planning) somando-se todos os critérios levantados pelo time durante a reunião de grooming. A quantidade de requisitos atendidos será capturada durante a fase de validação da implementação somando-se todos os critérios que de fato foram atendidos pela implementação.
Todas as coletas serão realizadas pelo autor do trabalho.

Sprint	Período		Total de requisitos			Requisitos atendidos			Taxa de requisitos atendidos	Observações
	Início	Fim	Quantidade	Coletado por	Data coleta	Quantidade	Coletado por	Data coleta		
1	13/02/17	17/02/17	3	Guilherme	13/02/17	2	Guilherme	15/02/17	67%	
2	20/02/17	24/02/17	2	Guilherme	20/02/17	2	Guilherme	24/02/17	100%	
3	06/03/17	10/03/17	17	Guilherme	06/03/17	17	Guilherme	10/03/17	100%	
4	13/03/17	17/03/17	6	Guilherme	13/03/17	6	Guilherme	14/03/17	100%	
5	20/03/17	24/03/17	2	Guilherme	20/03/17	2	Guilherme	22/03/17	100%	
6	27/03/17	31/03/17	3	Guilherme	27/03/17	0	Guilherme	31/03/17	0%	
7	17/04/17	21/04/17	16	Guilherme	17/04/17	15	Guilherme	21/04/17	94%	
8	24/04/17	28/04/17	4	Guilherme	24/04/17	4	Guilherme	26/04/17	100%	
9	08/05/17	12/05/17	7	Guilherme	08/05/17	7	Guilherme	10/04/17	100%	

Apendice C

Apêndice D – Planilha de coleta de métricas questão 1 métrica 3

Q1M3 - Taxa de requisitos levantados após início da implementação

A taxa de requisitos levantados após início da implementação será a quantidade de requisitos levantados após já ter sido dados início a implementação em comparação com a quantidade total de requisitos.
Taxa = quantidade de requisitos levantados após início da implementação / quantidade total de requisitos.

A quantidade total de requisitos será capturada no início da sprint, ao final da reunião de planejamento (Sprint Planning) somando-se todos os critérios levantados pelo time durante a reunião de grooming. A quantidade de requisitos levantados após início da implementação será capturada durante a fase de validação da implementação somando-se todos os critérios levantados após início da implementação.
Todas as coletas serão realizadas pelo autor do trabalho.

Sprint	Período		Total de requisitos			Requisitos levantados após início da implementação			Taxa de requisitos levantados após início da implementação	Observações
	Início	Fim	Quantidade	Coletado por	Data coleta	Quantidade	Coletado por	Data coleta		
1	13/02/17	17/02/17	3	Guilherme	13/02/17	0	Guilherme	15/02/17	0%	
2	20/02/17	24/02/17	2	Guilherme	20/02/17	1	Guilherme	24/02/17	50%	Acessar API e ignorar web service.
3	06/03/17	10/03/17	17	Guilherme	06/03/17	0	Guilherme	10/03/17	0%	
4	13/03/17	17/03/17	6	Guilherme	13/03/17	0	Guilherme	14/03/17	0%	
5	20/03/17	24/03/17	2	Guilherme	20/03/17	1	Guilherme	22/03/17	50%	
6	27/03/17	31/03/17	3	Guilherme	27/03/17	1	Guilherme	31/03/17	33%	Alterar posição dos campos obrigatórios
7	17/04/17	21/04/17	16	Guilherme	17/04/17	1	Guilherme	21/04/17	6%	Bloquear aditivo com data anterior ao último
8	24/04/17	28/04/17	4	Guilherme	24/04/17	0	Guilherme	26/04/17	0%	
9	08/05/17	12/05/17	7	Guilherme	08/05/17	0	Guilherme	10/04/17	0%	

Apêndice E – Planilha de coleta de métricas questão 1 métrica 4

Q1M4 - Quantidade de defeitos encontrados por testes exploratórios

A quantidade de defeitos encontrados por testes exploratórios será a quantidade de defeitos que foram descobertos após ser sinalizado pelo implementador que a implementação está concluída.
Quantidade = total de defeitos encontrados.

A quantidade total de defeitos será capturada durante a fase de implementação, após ser sinalizado pelo implementador que a implementação está concluída e então é executado o "*buddy test*". Somente defeitos não oriundos de código legado serão considerados.

Sprint	Período		Total de defeitos			Observações
	Início	Fim	Quantidade	Testado por	Data coleta	
1	13/02/17	17/02/17	1	Guilherme	15/02/17	Não estava passando para a atividade seguinte do workflow
2	20/02/17	24/02/17	0	Guilherme	24/02/17	
3	06/03/17	10/03/17	0	Guilherme	10/03/17	
4	13/03/17	17/03/17	1	Guilherme	14/03/17	
5	20/03/17	24/03/17	0	Guilherme	22/03/17	
6	27/03/17	31/03/17	1	Guilherme	31/03/17	Não validava tipo do estado civil para obrigar dados do cônjuge
7	17/04/17	21/04/17	0	Guilherme	21/04/17	
8	24/04/17	28/04/17	0	Guilherme	26/04/17	
9	08/05/17	12/05/17	0	Guilherme	10/04/17	

Apêndice F – Planilha de coleta de métricas questão 2 métrica 1

Q2M1 - Quantidade de falhas encontradas na revisão

A quantidade de falhas encontradas na revisão será a quantidade de defeitos na qualidade do código que foram descobertos pelo processo de revisão de código.
Quantidade = total de defeitos encontrados.

A quantidade total de defeitos será capturada durante a fase de implementação, após ser sinalizado pelo implementador que a implementação está concluída e então é executada a revisão do código. Somente defeitos não oriundos de código legado serão considerados.

Sprint	Período		Falhas na revisão			Observações
	Início	Fim	Quantidade	Revisado por	Data coleta	
1	13/02/17	17/02/17	0	Guilherme	15/02/17	
2	20/02/17	24/02/17	0	Guilherme	24/02/17	
3	06/03/17	10/03/17	0	Guilherme	10/03/17	
4	13/03/17	17/03/17	0	Guilherme	14/03/17	
5	20/03/17	24/03/17	0	Guilherme	22/03/17	
6	27/03/17	31/03/17	0	Guilherme	31/03/17	
7	17/04/17	21/04/17	0	Guilherme	21/04/17	
8	24/04/17	28/04/17	0	Guilherme	26/04/17	
9	08/05/17	12/05/17	0	Guilherme	10/04/17	

Apendice F

Apêndice G – Planilha de coleta de métricas questão 2 métrica 2

Q2M2 - Dificuldade de entendimento do código

A dificuldade de entendimento do código diz respeito ao esforço feito para assimilar o código gerado pelo desenvolvedor aos requisitos que devem ser implementados.

Escala:

Bom = Código legível e de fácil compreensão

Regular = Não está simples, porém de acordo com os requisitos

Ruim = Muito complexo. Não consigo assimilar aos requisitos

A dificuldade de entendimento do código será capturada durante a fase de implementação, após ser sinalizado pelo implementador que a implementação está concluída e então é executada a revisão do código. Códigos do legado não serão considerados.

Sprint	Período		Nível de entendimento do código			Observações
	Início	Fim	Nível	Revisado por	Data coleta	
1	13/02/17	17/02/17	Bom	Guilherme	15/02/17	Código existente já é bom
2	20/02/17	24/02/17	Bom	Guilherme	24/02/17	Código legado
3	06/03/17	10/03/17	Regular	Guilherme	10/03/17	Código legado
4	13/03/17	17/03/17	Bom	Guilherme	14/03/17	Código existente já é bom
5	20/03/17	24/03/17	Bom	Guilherme	22/03/17	Código existente já é bom
6	27/03/17	31/03/17	Ruim	Guilherme	31/03/17	Código legado
7	17/04/17	21/04/17	Bom	Guilherme	21/04/17	Código novo
8	24/04/17	28/04/17	Regular	Guilherme	26/04/17	Código legado
9	08/05/17	12/05/17	Regular	Guilherme	10/04/17	Código legado

Apêndice H – Planilha de coleta de métricas questão 2 métrica 3

Q2M3 - Quantidade de testes automatizados implementados

A quantidade de testes automatizados implementados será a quantidade total de novos testes implementados para cobrir os novos cenários introduzidos pela implementação.
Quantidade = total de testes automatizados implementados.

A quantidade total de novos de testes automatizados implementados será coletada ao final da sprint, após todas as histórias estarem concluídas.

Sprint	Período		Testes automatizados			Observações
	Início	Fim	Quantidade	Revisado por	Data coleta	
1	13-02-17	17-02-17	7	Guilherme	15-02-17	
2	20-02-17	24-02-17	0	Guilherme	24-02-17	Já existem testes na funcionalidade
3	06-03-17	10-03-17	0	Guilherme	10-03-17	Já existem testes na funcionalidade
4	13-03-17	17-03-17	4	Guilherme	14-03-17	
5	20-03-17	24-03-17	2	Guilherme	22-03-17	
6	27-03-17	31-03-17	0	Guilherme	31-03-17	
7	17-04-17	21-04-17	28	Guilherme	21-04-17	
8	24-04-17	28-04-17	0	Guilherme	26-04-17	
9	08-05-17	12-05-17	4	Guilherme	10-04-17	

Apêndice I – Planilha de coleta de métricas questão 3 métrica 1

Q3M1 - Taxa de histórias de usuário com valor de pontos de história recalibrados após implementação

Taxa de histórias de usuário com valor de pontos de história recalibrados após implementação será a quantidade de histórias de usuário que não foram assertivos na estimativa em pontos de história acordado pelo time durante a reunião de grooming dividido pelo total de histórias de usuário alocadas para a sprint.

Taxa = total de histórias de usuário com estimativa recalibrada / total de histórias alocadas para a sprint.

A taxa de histórias de usuário com valor de pontos de história recalibrados será coletada ao final da sprint, após todas as histórias estarem concluídas.

Sprint	Período		Histórias de usuário			Taxa	Observações
	Início	Fim	Total	Recalibradas	Data coleta		
1	13/02/17	17/02/17	1	0	15/02/17	0%	
2	20/02/17	24/02/17	1	1	24/02/17	100%	
3	06/03/17	10/03/17	5	3	10/03/17	60%	
4	13/03/17	17/03/17	1	0	14/03/17	0%	
5	20/03/17	24/03/17	1	0	22/03/17	0%	
6	27/03/17	31/03/17	1	1	31/03/17	100%	
7	17/04/17	21/04/17	1	0	21/04/17	0%	
8	24/04/17	28/04/17	2	0	26/04/17	0%	
9	08/05/17	12/05/17	3	1	10/04/17	33%	

Apendice I

Apêndice J – Planilha de coleta de métricas questão 3 métrica 2

Q3M2 - Percentual de alteração no valor do ponto de história das histórias alteradas

Percentual de alteração no valor do ponto de história alterado após implementação será a quantidade de histórias de usuário que não foram assertivos na estimativa em pontos de história acordado pelo time durante a reunião de grooming dividido pelo total de histórias de usuário alocadas para a sprint.

Taxa = (total de pontos histórias de usuário / total de pontos histórias recalibrado) - 1.

A taxa de histórias de usuário com valor de pontos de história recalibrados será coletada ao final da sprint, após todas as histórias estarem concluídas.

Sprint	Período		Histórias de usuário			Taxa	Observações
	Início	Fim	Total	Recalibradas	Data coleta		
1	13/02/17	17/02/17	3	3	15/02/17	0%	
2	20/02/17	24/02/17	13	8	24/02/17	-38%	
3	06/03/17	10/03/17	25	19	10/03/17	-24%	
4	13/03/17	17/03/17	5	5	14/03/17	0%	
5	20/03/17	24/03/17	8	8	22/03/17	0%	
6	27/03/17	31/03/17	3	5	31/03/17	67%	
7	17/04/17	21/04/17	21	21	21/04/17	0%	
8	24/04/17	28/04/17	21	21	26/04/17	0%	
9	08/05/17	12/05/17	13	15	10/04/17	15%	

Apêndice K – Planilha de coleta de métricas questão 3 métrica 3

Q3M3 - Esforço para especificar requisitos

O esforço para especificar os requisitos será a quantidade de pessoas alocadas para construir a especificação multiplicado pela quantidade de horas gastas por cada uma. Como a especificação será elaborada em dois momentos distintos quando for dado início à análise dos artefatos de caso de uso e diagrama de atividades, essa medição também será realizada em dois momentos distintos, sendo o primeiro momento durante o grooming, onde todo o time elabora a especificação em conjunto, e outro momento sendo o esforço de um analista elaborando o artefato naquela sprint.

Esforço = total pessoas alocadas para elaborar a especificação * total de horas de cada pessoa.

O esforço para especificar requisitos será coletada após a conclusão da elaboração da especificação.

Sprint	Período		Esforço no grooming			Esforço individual do artefato em análise			Esforço total	Observações
	Início	Fim	Qt Pessoas	Horas	Total coletivo	Data coleta	Quantidade	Horas	Data coleta		
1	13-02-17	17-02-17	6	0.25	1.5	09-02-17		0	09-02-17	1.5	
2	20-02-17	24-02-17	6	0.15	0.9	16-02-17		0	16-02-17	0.9	
3	06-03-17	10-03-17	6	0.3	1.8	06-03-17		0	06-03-17	1.8	
4	13-03-17	17-03-17	5	0.5	2.5	09-03-17	1	0.3	09-03-17	2.8	
5	20-03-17	24-03-17	5	0.6	3	16-03-17	1	0.3	16-03-17	3.3	
6	27-03-17	31-03-17	4	0.25	1	23-03-17	1	0.2	23-03-17	1.2	
7	17-04-17	21-04-17	6	1.5	9	13-04-17	1	1	13-04-17	10.0	
8	24-04-17	28-04-17	7	1	7	20-04-17	1	0.5	20-04-17	7.5	
9	08-05-17	12-05-17	6	0.5	3	04-05-17	1	0.5	04-05-17	3.5	

Apendice K

Apêndice L – Caso de Uso 1 Sprint 4

ID: 56317.3

Caso de uso: alteração do analista responsável pelo workflow da proposta de contrato do prospect por usuário autorizado.

Pré-condições:

- O usuário é do tipo comum;
- O usuário **ESTÁ** autorizado a utilizar a funcionalide de Workflow e editar o analista responsável dos fluxos de proposta de contrato do prospect;
- O usuário está associado à um credor do tipo "Colaborador";
- O usuário tem visibilidade liberada para todos os prospects, empreendimentos e empresas do sistema.

Pós-condições:

- Nenhuma.

INTENÇÃO DO USUÁRIO/ATOR	RESPONSABILIDADE DO SISTEMA
Consulta fluxo de proposta de contrato do prospect	Verifca se o usuário logado está associado à um credor cadastrado no sistema e que seja do tipo "Colaborador". Verifica se o usuário está autorizado à utilizar a funcionalidade de Workflow. Valida se o usuário tem visibilidade sobre o prospect, empresa e empreendimento consultado. Valida se o usuário está autorizado para editar o analista responsável do fluxo Apresenta a informações do fluxo. Libera a edição do analista responsável do fluxo.
Edita o analista responsável e confirma edição do fluxo de proposta de contrato do prospect	Valida se o fluxo não foi encerrado verificando se a atividade corrente não é "Fim de fluxo". Valida se o fluxo deve passar para a atividade seguinte verificando se a atividade seguinte foi alterada. Altera o analista responsável do fluxo. Apresenta mensagem de sucesso.

Apêndice M – Caso de Uso 2 Sprint 4

ID: 56317.2

Caso de uso: alteração do analista responsável pelo workflow da proposta de contrato do prospect por usuário não autorizado.

Pré-condições:

- O usuário é do tipo comum;
- O usuário está autorizado a utilizar a funcionalide de Workflow.
- O usuário **NÃO ESTÁ** autorizado a editar o analista responsável dos fluxos de proposta de contrato do prospect;
- O usuário está associado à um credor do tipo "Colaborador";
- O usuário tem visibilidade liberada para todos os prospects, empreendimentos e empresas do sistema.

Pós-condições:

- Nenhuma.

INTENÇÃO DO USUÁRIO/ATOR	RESPONSABILIDADE DO SISTEMA
Acessa o sistema de Pró-Vendas	Verifca se o usuário logado está associado à um credor cadastrado no sistema e que seja do tipo "Colaborador".
Consulta fluxo de proposta de contrato do prospect	Verifica se o usuário está autorizado à utilizar a funcionalidade de Workflow. Valida se o usuário tem visibilidade sobre o prospect, empresa e empreendimento consultado. Valida se o usuário está autorizado para editar o analista responsável do fluxo Apresenta a informações do fluxo. Não libera a edição do analista responsável do fluxo.

Apêndice N – Caso de Uso 3 Sprint 4

ID: 56317.1

Caso de uso: dar autorização de alteração do analista responsável pelo workflow de proposta de contrato de prospect para usuários do sistema.

Pré-condições:

- O usuário é do tipo comum;
- O usuário está autorizado a editar as configurações gerais da funcionalidade do Workflow;
- O usuário está associado à um credor do tipo "Colaborador";

Pós-condições:

- Nenhuma.

INTENÇÃO DO USUÁRIO/ATOR	RESPONSABILIDADE DO SISTEMA
Acessa o sistema de Pró-Vendas	Verifca se o usuário logado está associado à um credor cadastrado no sistema e que seja do tipo "Colaborador".
Acessa as configurações gerais da funcionalidade de Workflow.	Verifica se o usuário está autorizado a editar as configurações gerais do Workflow. Apresenta a opção de incluir quaisquer usuários do sistema ao grupo de usuários autorizados à editar o analista responsável dos fluxos.

Apêndice O – Caso de Uso 1 Sprint 5

ID: 56319.1

Caso de uso: associar um workflow à um status de pasta incompleta por ausência de documento obrigatório.

Pré-condições:

- O usuário é do tipo comum;
- O usuário está autorizado a utilizar a funcionalidade de Workflow;
- O usuário está autorizado a cadastrar status de pasta incompleta;
- O usuário está associado à um credor do tipo "Colaborador";

Pós-condições:

- Nenhuma.

INTENÇÃO DO USUÁRIO/ATOR	RESPONSABILIDADE DO SISTEMA
Acessa o sistema de Pró-Vendas	Verifca se o usuário logado está associado à um credor cadastrado no sistema e que seja do tipo "Colaborador".
Cadastra um status de pasta incompleta "Ausência de documento obrigatório"	Verifica se o usuário está autorizado a cadastrar statude pasta incompleta. Apresenta mensagem de sucesso.
Consulta fluxo de proposta de contrato do prospect	Verifica se o usuário está autorizado à utilizar a funcionalidade de Workflow. Valida se o usuário tem visibilidade sobre o prospect, empresa e empreendimento consultado. Apresenta a informações do fluxo.
Edita status de pasta incompleta	Apresenta todos os status de pasta incompleta cadastrados no sistema.
Associa status de pasta incompleta à um fluxo de proposta de contrato de prospect e confirma edição.	Valida se o fluxo não foi encerrado verificando se a atividade corrente não é "Fim de fluxo". Valida se o fluxo deve passar para a atividade seguinte verificando se a atividade seguinte foi alterada. Altera o status de pasta incompleta do fluxo. Apresenta mensagem de sucesso.

Apêndice P – Caso de Uso 1 Sprint 6

ID: 56320.1

Caso de uso: o cadastro de prospects deve obrigar o preenchimento do CPF, endereço, data de nascimento, RG, renda, mídia de captação e no caso do estado civil do tipo casado deve obrigar o preenchimento do nome, cpf e rg do cônjuge.

Pré-condições:

- O usuário é do tipo comum;
- O usuário **ESTÁ** autorizado a cadastrar novos prospects no sistema;
- O usuário está associado à um credor do tipo "Colaborador";

Pós-condições:

- Nenhuma.

INTENÇÃO DO USUÁRIO/ATOR	RESPONSABILIDADE DO SISTEMA
Carrega o cadastro de prospect	Verifca se o usuário logado está associado à um credor cadastrado no sistema e que seja do tipo "Colaborador". Verifica se o usuário está autorizado a cadastrar novos prospects no sistema.
Preenche cpf, endereço, data de nascimento, rg, renda, mídia de captação e seleciona um estado civil do tipo casado.	Expande automaticamente a seção de informações do cônjuge. Os campos "Nome", "CPF" e "RG" da seção de informações do cônjuge são sinalizados como obrigatórios com um "*".
Confirma cadastro do prospect sem preencher as informações obrigatórias do cônjuge	Impede que o cadastro seja realizado e sinaliza quais campos obrigatórios do cônjuge não foram preenchidos.

Apêndice Q – Caso de Uso 2 Sprint 6

ID: 56320.2

Caso de uso: o cadastro de prospects deve obrigar o preenchimento do CPF, endereço, data de nascimento, RG, renda, mídia de captação e no caso do estado civil do tipo casado deve obrigar o preenchimento do nome, cpf e rg do cônjuge.

Pré-condições:

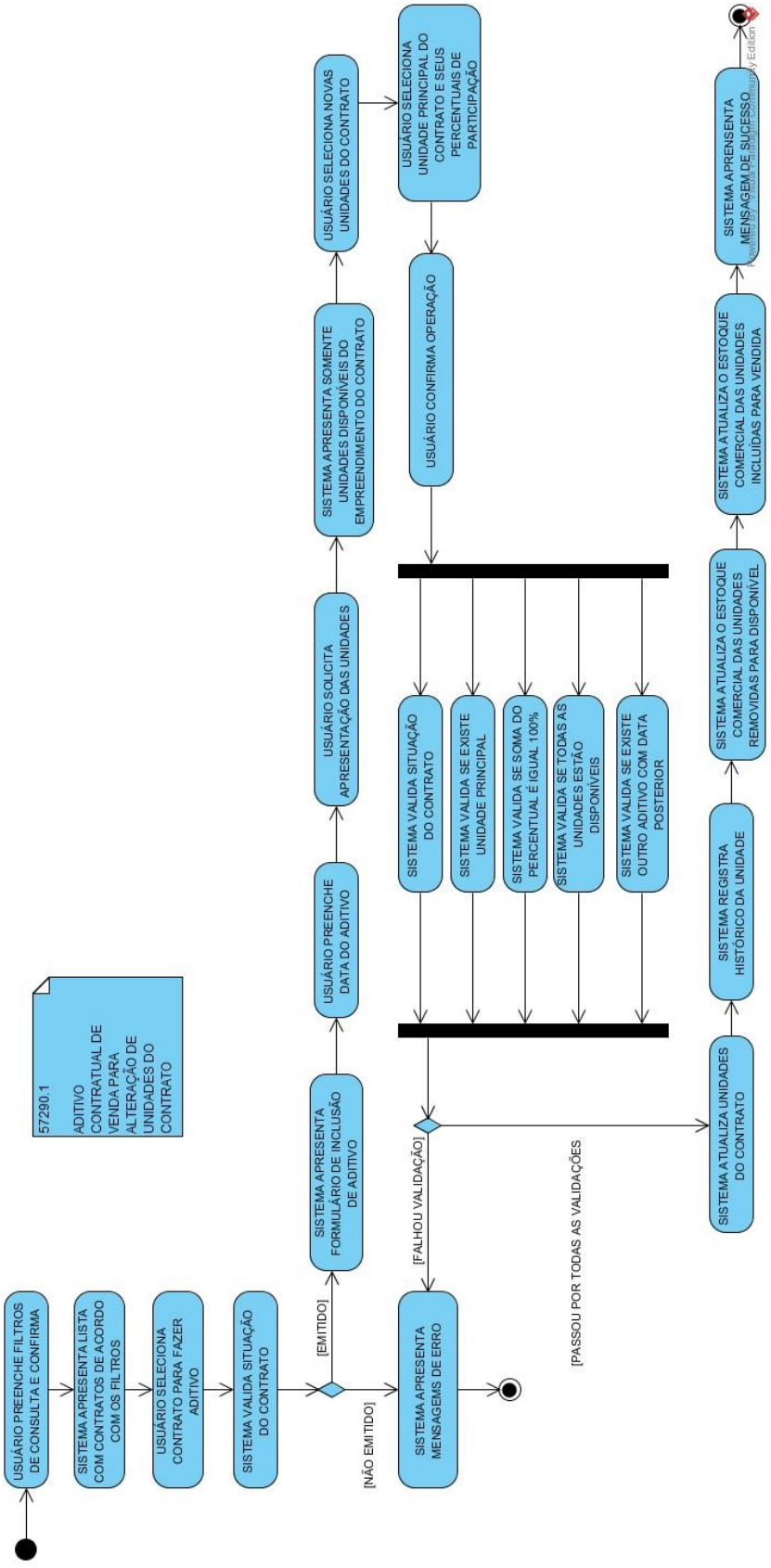
- O usuário é do tipo comum;
- O usuário **ESTÁ** autorizado a cadastrar novos prospects no sistema;
- O usuário está associado à um credor do tipo "Colaborador";

Pós-condições:

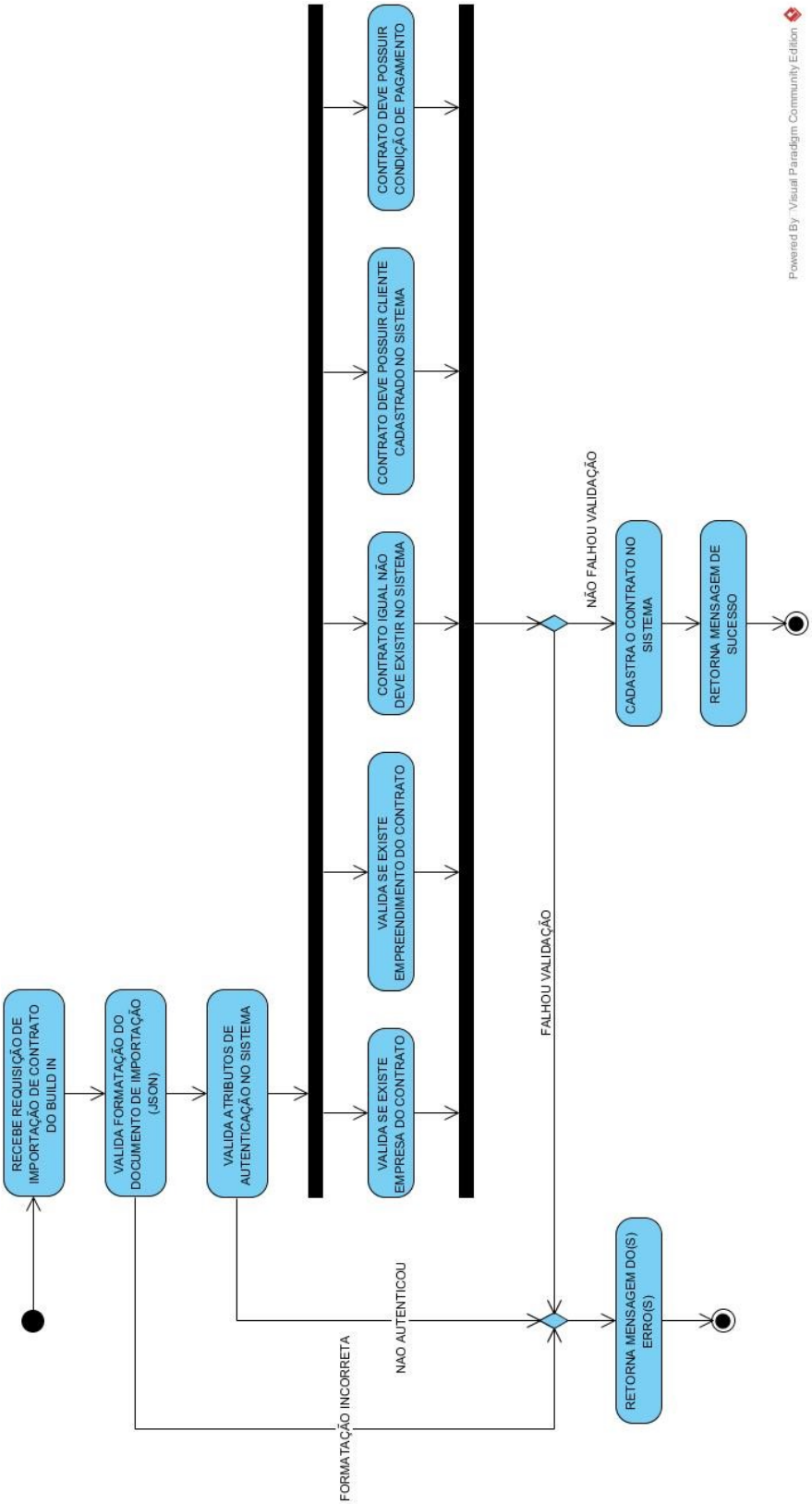
- Nenhuma.

INTENÇÃO DO USUÁRIO/ATOR	RESPONSABILIDADE DO SISTEMA
Carrega o cadastro de prospect	Verifca se o usuário logado está associado à um credor cadastrado no sistema e que seja do tipo "Colaborador". Verifica se o usuário está autorizado a cadastrar novos prospects no sistema.
Preenche cpf, endereço, data de nascimento, rg, renda, mídia de captação e seleciona um estado civil do tipo diferente de casado.	Recolhe automaticamente a seção de informações do cônjuge se estiver aberta.
Confirma cadastro do prospect sem preencher as informações cônjuge	Apresenta mensagem de sucesso.

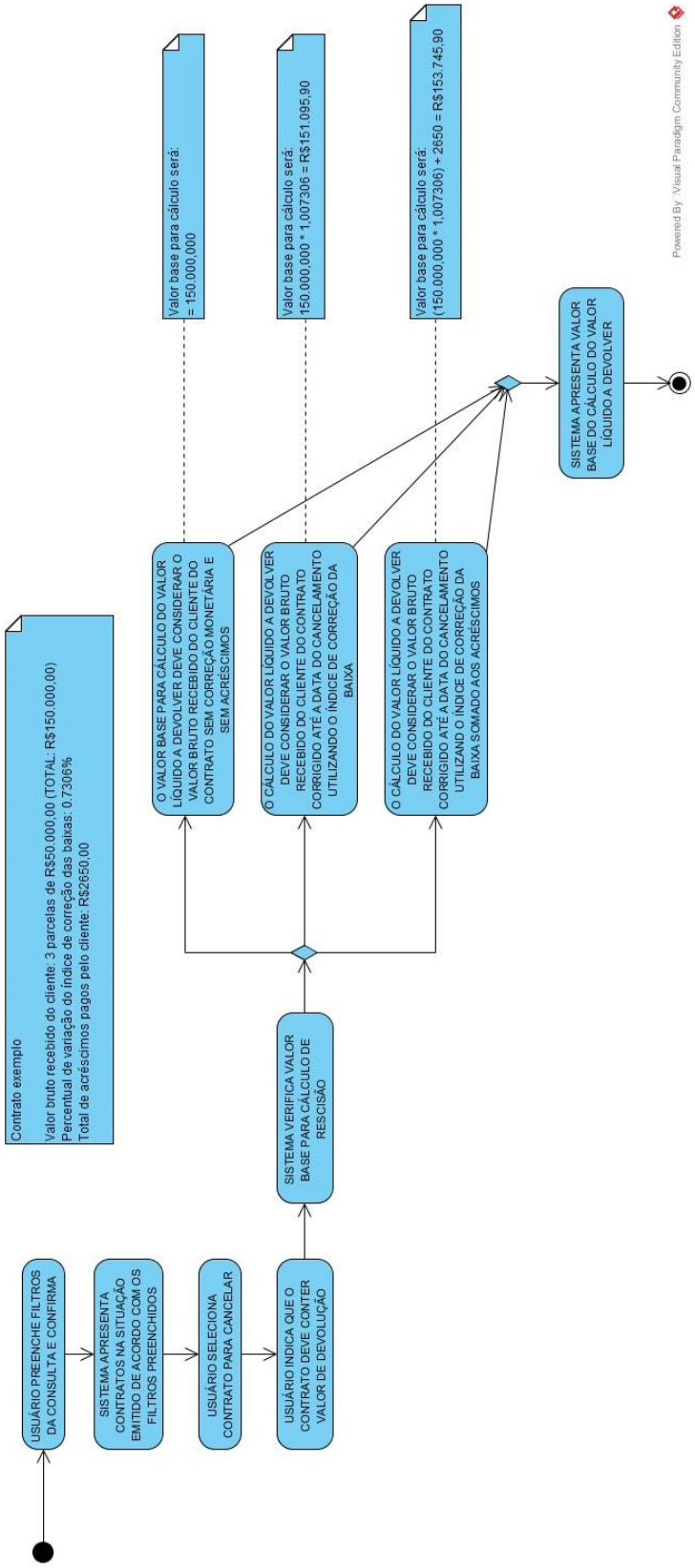
Apêndice R– Diagrama de atividade 1 Sprint 7



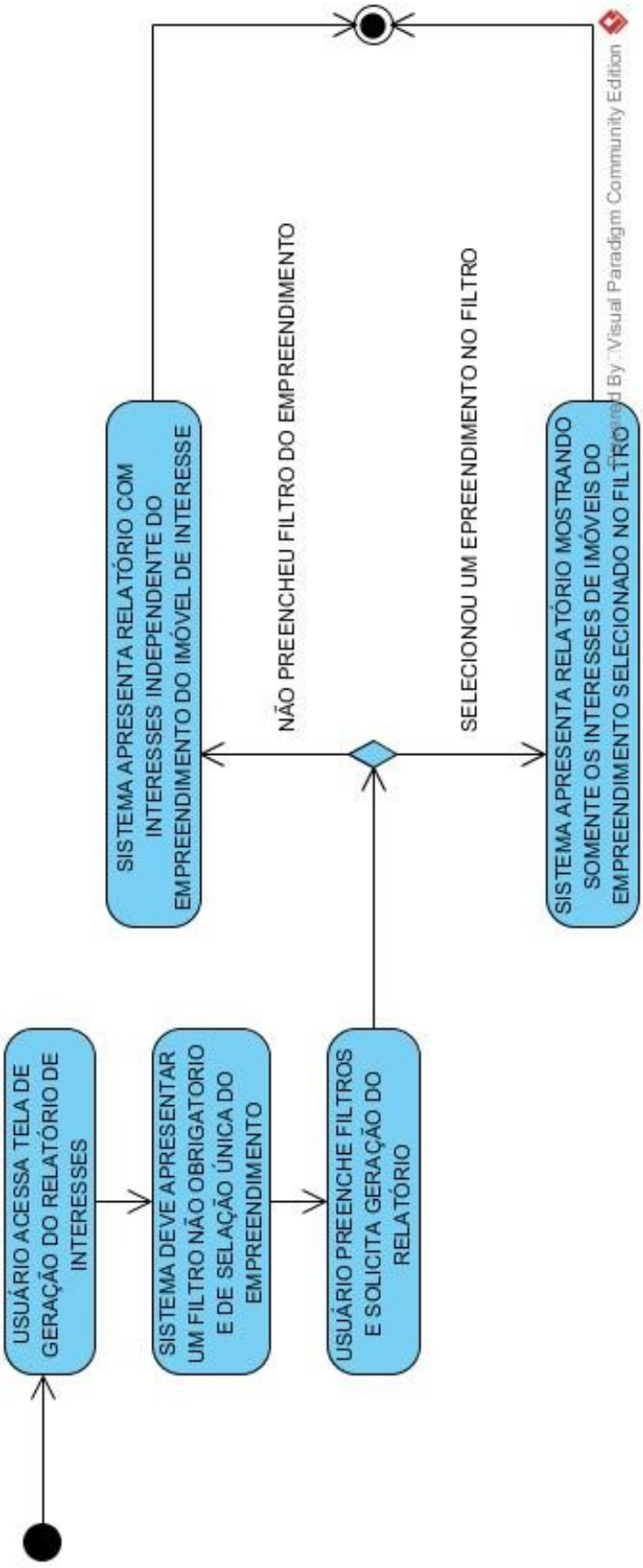
Apêndice S– Diagrama de atividade 1 Sprint 8



Apêndice T– Diagrama de atividade 1 Sprint 9



Apêndice U– Diagrama de atividade 2 Sprint 9



Análise Comparativa do Impacto dos Artefatos de Requisitos no Processo de Desenvolvimento de Software

Fabiane Barreto V. Benitti, Guilherme M. Tatibana

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

Fabiane.benitti@ufsc.br, gmitsuo@grad.ufsc.br

Abstract. *This paper aimed to analyze and measure the effects caused within the software development process by approaching two different ways to present the specification of requirements to a team that follows agile practices and compare with the approach already used. The two approaches selected were to add the artifacts of use case and activity diagram to the specification. Finally, with the results obtained in the case study, the recommendation that was established for the team that participated is the use of the activity diagram artifact to specify user stories with high estimate values.*

Resumo. *Este artigo teve como objetivo analisar e mensurar os efeitos causados dentro do processo de desenvolvimento de software abordando dois formatos diferentes de apresentar a especificação de requisitos à um time que segue práticas ágeis e comparar com a abordagem já utilizada. As duas abordagens selecionadas foram acrescentar os artefatos de caso de uso e diagrama de atividades à especificação. Por fim, com os resultados obtidos no estudo de caso, a recomendação que ficou estabelecida para o time que participou é a utilização do artefato de diagrama de atividades para especificar histórias de usuários com estimativas de valor alto.*

1. Introdução

Processos devem ser tecnicamente corretos e devem ser capazes de atender às necessidades do negócio. Entretanto, processos podem estar corretos do ponto de vista da engenharia de software e não serem competitivos. Podem consumir demasiado tempo e esforço ou não produzirem produtos com a qualidade necessária para satisfazer as necessidades de seus usuários. Processos podem apresentar problemas e devem ser objeto, continuamente, de melhorias. É importante, nesse contexto, dispor-se de mecanismos capazes de evidenciar problemas nos processos e apoiar na identificação de objetivos de melhoria [Rocha, Souza e Barcellos, 2012].

Dentro da organização e nos processos que se enquadram no contexto da pesquisa, os fornecedores e consumidores das especificações de requisitos muitas vezes fazem críticas à forma como ela é apresentada e elaborada. Para que se adeque ao processo, aos métodos ágeis, e agrade os responsáveis por elaborá-las e os desenvolvedores que irão consumi-las, desde o momento da concepção até o seu desenvolvimento, a especificação é vista e revisada por diversas vezes, os feedbacks nem sempre são positivos e o resultado final pode não ser satisfatório.

Desde 1986 Basili já afirmava que as variáveis presentes nos processos de desenvolvimento de software das organizações são diversas, portanto, não existe um modelo de processo padrão que irá atender de forma geral à todas elas. Cada organização tem seus projetos, seus ambientes e times de desenvolvimento. Há também uma variação de fatores em termos de custo e qualidade, experiências, domínio de problema, metodologias e limitações.

Muitas vezes é difícil demonstrar como os programas de melhoria geram valor comercial. Geralmente projetos de software não são capazes de demonstrar explicitamente suas contribuições para metas de nível superior e sucesso dos negócios [Basili, Trendowiz, *et al.*, 2014].

Para tanto, um estudo de caso foi desenvolvido dentro de um ambiente real de uma empresa de desenvolvimento de software de forma a demonstrar como a apresentação dos artefatos de especificação de requisitos influenciam o desenvolvimento de software do ponto de vista de qualidade funcional e estrutural do produto e qualidade do processo de desenvolvimento.

O passos para realização deste trabalho foram, elaborar um modelo de medição baseado no framework GQM, levantar diferentes alternativas de apresentar a especificação de requisitos e aplicá-las na prática, analisar os impactos sofridos pelos processos para cada tipo diferente aplicado dentro do ciclo de desenvolvimento do software através dos dados obtidos e discutir sobre os resultados através da análise descritiva dos dados.

Este artigo teve como objetivo avaliar o impacto de diferentes formas de especificação de requisitos em um time de desenvolvimento que utiliza práticas ágeis em uma organização específica de desenvolvimento de software por meio de um estudo de caso.

2. GQM

Segundo Basili, Caldiera e Rombach (2001), de acordo com muitos estudos sobre aplicação de métricas e modelos em ambientes industriais, para que a medição seja efetiva é necessário que esta seja: focada em objetivos específicos, aplicada em todo o ciclo de vida do produto, processo e recursos e interpretada baseado na caracterização e entendimento do contexto da organização, ambiente e objetivos. Isso significa que a medição precisa ser planejada com uma perspectiva top-down para que esteja de acordo com modelos e objetivos definidos pelo contexto da organização, visto que se feito da maneira inversa, bottom-up, a quantidade de variáveis e métricas a se observar é grande, e a forma de interpretá-las não ficará clara e visível sem um modelo com objetivos definidos. Já a sua execução, no processo de análise e interpretação dos dados coletados, a perspectiva deve ser bottom-up pois as métricas foram definidas focadas em um objetivo e as informações fornecidas através das métricas deve ser interpretadas e analisadas respeitando-os.

O método GQM é composto por quatro fases (**Figura 39**): planejamento, definição, coleta de dados e interpretação [Soligen e Berghout, 1999]. Na fase de planejamento contempla-se todo tipo de medida que deve ser tomada para que seja possível realizar a medição, definindo as pessoas envolvidas, treinamento necessários e o planejamento do projeto onde será feito o estudo. A fase de definição é reservada à elaboração do modelo com objetivos, questões e métricas definidas. Durante a fase de coleta de dados são definidos os formulários para a coleta dos dados que são preenchidos com os valores medidos e armazenados para posterior interpretação. Por fim, a fase de interpretação é utilizada para que os valores da medição sejam interpretados para responder às questões levantadas e afirmar se os objetivos foram de fato alcançados.

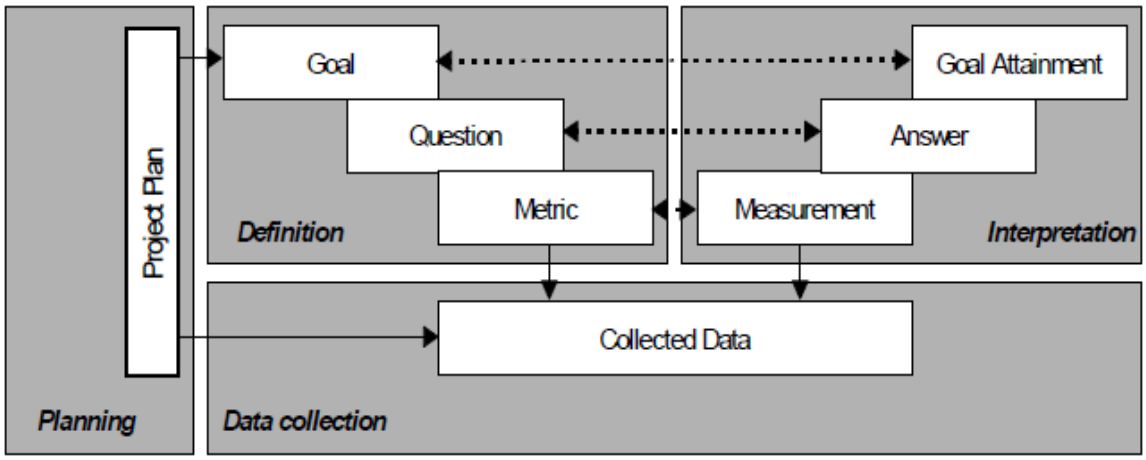


Figura 39. As quatro fases do GQM [Soligen e Berghout, 1999]

Um modelo GQM é uma estrutura hierárquica (**Figura 40**) que se inicia com um objetivo de medição, o qual especifica propósitos, objeto de medição, assuntos a serem medidos e pontos de vista dos quais a medição será feita. O refinamento deste objetivo deve então gerar as questões, que especificam os assuntos em perguntas, as quais serão respondidas por mais uma etapa de refinamento que geram as métricas, que podem ser objetivas ou subjetivas. Algumas métricas podem ser utilizadas para responder as mesmas perguntas sobre o mesmo objetivo, inclusive, modelos de GQM podem conter questões e métricas em comum, desde que seja levado em consideração o ponto de vista de onde são coletados.

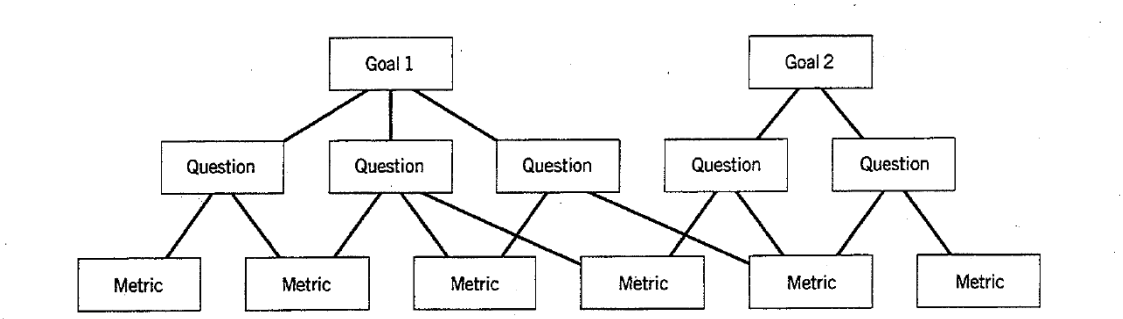


Figura 40. Estrutura do modelo hierárquico do GQM [Basili, Caldiera e Rombach, 2001]

A saída do processo de aplicação do GQM é uma especificação de um sistema de medições voltado para assuntos particulares e um conjunto de regras de interpretação dos dados obtidos.

3. Processos

O processo de desenvolvimento utilizado no projeto do estudo é baseado no Scrum, um framework para desenvolver e manter produtos complexos [Sutherlant e Schwaber, 2013], guiado pelos 4 princípios centrais do manifesto ágil [Beck, Beedle, *et al.*, 2001]. O time é formado por 7 integrantes, sendo um Product Owner, um Scrum Master, e 5 desenvolvedores, e realizam sprints de 1 semana.

O processo do time de desenvolvimento do estudo de caso, está divido em duas etapas, planejamento e execução (**Figura 41**), e inicia sempre na segunda feira seguinte ao término da sprint anterior. O resultado final ao término de uma sprint devem ser histórias de usuário no backlog da sprint implementadas, também chamadas de incremento , e histórias de usuário provenientes de solicitações de evolução do sistema devidamente refinadas, especificadas.

Na processo de planejamento, na primeira etapa da sprint, ocorre o sprint planning , a primeira tarefa do processo, que reúne o time de desenvolvimento para planejar os itens do backlog do produto que serão implementados e construir o sprint backlog, artefato que mostra quais histórias foram selecionadas para implementação na sprint. A primeira parte da reunião consiste de uma apresentação dos itens do backlog do produto com os requisitos já elicitados e especificados aos desenvolvedores, e se necessário, sanar dúvidas referentes às implementações. A segunda parte do planejamento fica reservada ao time para discutir sobre o desenvolvimento.

Após a etapa de planejamento, o restante dos dias da sprint é utilizado para a execução de dois processos paralelos, o grooming ou refinamento e a implementação de história.

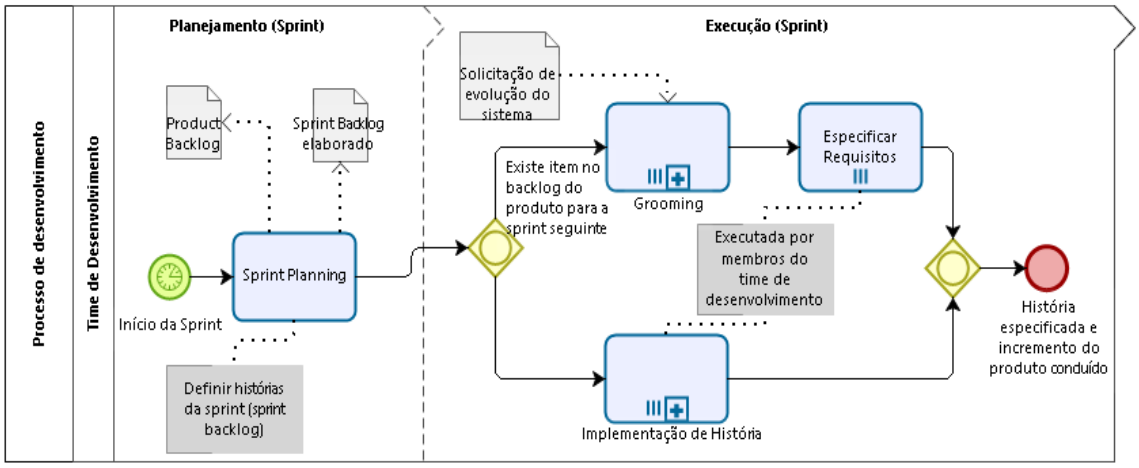


Figura 41. Processo de desenvolvimento do time do do estudo

O processo do grooming, também chamado de refinamento, é realizado quando existe uma demanda de evolução do sistema. O objetivo deste processo é fazer uma análise da demanda com o time inteiro, avaliar a forma mais eficiente e rápida de entregar valor para o cliente separando uma única solicitação de evolução em histórias de usuário menores, que podem ser implementadas em um menor espaço de tempo e entregues ao cliente antes que toda a solução seja desenvolvida, e estimar em pontos chamados de story points que são utilizados para medir a velocidade do time e definir quantas histórias serão colocadas no sprint backlog.

Após a elaboração de histórias de usuário são então levantados quais os critérios cada uma delas deve estar de acordo para que sejam consideradas como concluídas, e com isso, o processo é encerrado, gerando como saída, um conjunto de histórias de usuário com seus devidos critérios de aceite e estimadas em story points.

Tendo em mãos as histórias e critérios de aceite, os membros do time de desenvolvimento podem então dar início à tarefa de especificação dos requisitos, que deve elaborar os artefatos que servirão de suporte para os desenvolvedores implementar as histórias. Estas histórias então estão aptas para serem implementadas, portanto podem entrar no sprint backlog em futuras sprints.

O processo de implementação de história (**Figura 42**) é iniciado pelo trabalho de "Codificar solução", que é quando um implementador pega a primeira tarefa definida para a história para dar início à codificação, tarefa essa que foi planejada com base na especificação de requisitos, que contém os artefatos elaborados para guiar o implementador durante o desenvolvimento da nova funcionalidade. Mais de uma história pode ser implementada ao mesmo tempo, ou a mesma história pode ser implementada por diferentes desenvolvedores ao mesmo tempo, chamado de programação pareada. A tarefa de codificar solução também inclui

implementar testes automatizados para garantir o funcionamento dos critérios de aceite definidos na especificação.

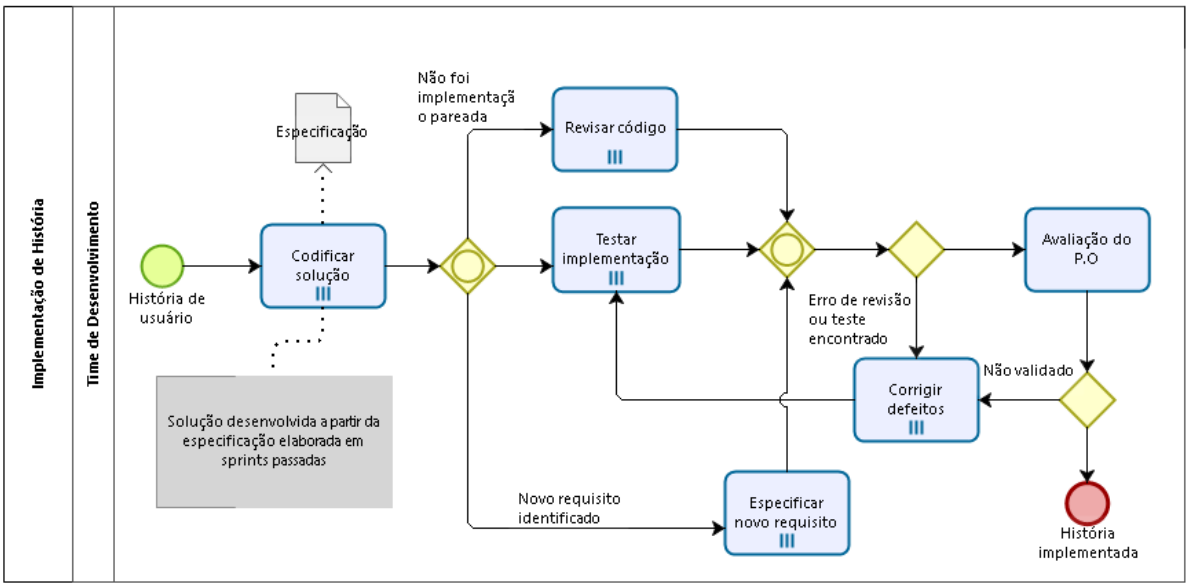


Figura 42. Processo de implementação de história

Ao concluir uma tarefa de codificar solução, o código gerado para desenvolver a história é revisado por outro desenvolvedor quando é executada a tarefa "Revisar código", porém somente quando não houve codificação pareada. Os requisitos implementados e definidos pela história são testados em conjunto pelo desenvolvedor e por outro integrante do time de desenvolvimento quando é executada a tarefa chamada de "budy testing" ou "pair testing" e, quando um requisito não previsto for identificado, o mesmo deve então ser especificado. Enquanto erros forem encontrados durante uma revisão de código ou teste exploratório, estes devem então ser corrigidos e testados novamente até que a história possa então ser validada pelo PO, que executa a tarefa focado na necessidade que motivou a implementação da história.

4. Estudo de Caso

A qualidade do software é um importante contribuinte para a entrega de valor ao negócio. As consequências da baixa qualidade de software com o passar do tempo acaba se tornando insustentável para o negócio. A baixa qualidade de software leva a perdas financeiras por perdas de negociações, custo de manutenção, perda de clientes, ações na justiça e eventualmente, perda de credibilidade. Não existe uma única maneira correta de se pensar sobre qualidade de software, entretanto, fica mais fácil agrupando diversos componentes em três grupos distintos, qualidade funcional, estrutural e do processo [Chappell, 201?].

A qualidade funcional lida com os aspectos importantes para os stakeholders, que avaliam o valor do software através dos atributos de qualidade funcional, por exemplo, se as funcionalidades presentes atendem à sua necessidade de negócio. A qualidade do processo lida com o estado do processo de desenvolvimento. Estar de acordo com os prazos e orçamentos planejados é responsabilidade do processo de desenvolvimento. Qualidade estrutural é uma visão de qualidade interna do produto como testabilidade e manutenibilidade do código fonte. Mesmo sendo uma visão interna, ela é refletida externamente através do produto.

O objetivo do estudo de caso está descrito nas seção seguinte e foi construído tendo em mente que o objeto de estudo foi a especificação, que o principal objetivo dos processos ágeis é a entrega de valor de negócio e que a qualidade do software é importante para isso.

4.1. Planejamento

A medição foi realizada em apenas um time de apenas um projeto da organização. O time é composto de sete membros, no qual todos já estavam familiarizados com os artefatos e ferramentas do processo da organização e da medição, descartando a necessidade de envolver qualquer treinamento no estudo de caso. Apenas um repasse sobre o planejamento da coleta de dados foi feito para deixá-los cientes de que em momentos específicos das sprints seriam coletadas algumas medidas à respeito do processo e dos resultados gerados na semana.

Para a realização do estudo de caso foi construído um modelo de medição que no nível conceitual possui um objetivo que descreve o objeto do estudo relacionado ao propósito que se busca alcançar. No nível operacional define uma questão para cada frente do objetivo (qualidade funcional, estrutural e do processo de desenvolvimento) para caracterizar como a realização deste será alcançada e também qual o contexto foi levado em consideração para tal questionamento. Por fim, no nível quantitativo, são descritos os conjuntos de dados que serão coletados e associados à cada questão para que esta possa ser respondida quantitativamente, juntamente com uma descrição de como estas medidas serão interpretadas.

4.2. Construção do modelo

4.2.1 Objetivo

Tendo como base o modelo de especificação de requisitos, o processo de desenvolvimento utilizado pela organização e os fatores de qualidade de software, o objetivo da medição foi avaliar: a qualidade funcional, estrutural e do processo observada ao final do desenvolvimento dos requisitos;

4.2.1 Questões e métricas

A primeira questão do modelo, relacionada a qualidade funcional, foi: a necessidade que motivou o desenvolvimento da funcionalidade foi compeltamente atendida? Para que exista a demanda de solicitação de uma nova funcionalidade, deve existir um motivador por parte dos stakeholders que trouxe essa necessidade. Para que seja vista entrega de valor, essa necessidade tem que ser entendida e atendida por essa nova funcionalidade, caso contrário, pode gerar insatisfação. Essa necessidade é exposta ao time de desenvolvimento através dos artefatos de requisitos da especificação.

Para esta questão, as métricas coletadas foram, o percentual de requisitos não atendidos, o percentual de requisitos atendidos, o percentual de requisitos levantados após o início da implementação e a quantidade de defeitos encontrados por testes exploratórios. Quanto mais artefatos de requisitos forem apresentados pela especificação, menores são as chances de que requisitos funcionais não sejam implementados, menores são as chances de que requisitos elicitados não sejam incluídos na especificação, e melhores são as chances de que não apareçam defeitos ou que sejam encontrados previamente para que possam ser corrigidos antes da entrega da funcionalidade.

A segunda questão incluída no modelo, relacionada a qualidade estrutural, foi: qual a qualidade do código gerado para desenvolver a funcionalidade? Como o processo de desenvolvimento das histórias de usuário permite que a mesma história de usuário seja

implementada por mais de um desenvolvedor ao mesmo tempo, é importante que o compreensão do que deve ser feito esteja claro e de forma alinhada entre todos os desenvolvedores do time, caso de requisitos que geram dúvidas ou estejam ambíguos podem refletir de maneira negativa no código gerado.

Para esta questão, as métricas coletadas foram, a quantidade de falhas encontradas no processo de revisão do código, a qualidade do código gerado pela implementação em uma escala ordinal (bom, regular, ruim), e a quantidade de testes automatizados implementados. Uma especificação mais completa torna mais fácil a interpretação do "como" deve ser desenvolvida a funcionalidade, gerando códigos mais desacoplados e coesos, sendo assim mais legíveis, mais testáveis e de manutenção menos custosa. Códigos limpos são mais fáceis de gerar testes de unidade que são automatizados pelo sistema de integração contínua do desenvolvimento, sendo assim, é maior a garantia de que se eventualmente sofrerem manutenção, não tenham seu comportamento alterado ou acusem falha de testes quando houver um erro.

A terceira questão, relacionada com a qualidade do processo, foi: qual a precisão das estimativas levantadas no planejamento? Durante o processo de planejamento da sprint, as histórias são apresentadas ao time de desenvolvimento, o qual na segunda etapa deste processo, discute e sugere uma estimativa do esforço para o desenvolvimento em pontos de história. Quanto mais artefatos de requisitos forem apresentados ao time sobre o que deve ser desenvolvido, mais precisa será a estimativa do esforço necessário para implementar a funcionalidade.

Para esta questão, as métricas coletadas foram, taxa de histórias de usuário com valor de pontos de história recalibrados após implementação, taxa de alteração no valor do ponto de história das histórias alteradas e esforço para especificar os requisitos, que será medido em horas gastas por cada indivíduo do time. Uma alta quantidade de histórias com valor de pontos alterados, e um alto percentual neste valor de alteração, indica que a visão do que precisa ser desenvolvido ainda não está ideal. Um dos fatores que influenciam na qualidade do processo é o prazo de entrega, e um aumento muito significativo no esforço pode compromete-lo. Se os benefícios que esse esforço extra proporciona não compensam a diferença introduzida no processo, ele não vale a pena ser mantido.

4.3. Coleta de dados

Para executar o estudo as métricas foram coletadas por sprint do time. Foram utilizadas três sprints para cada formato apresentado. Nas três sprints iniciais nenhuma alteração foi aplicada à especificação para a coleta dos dados. A três sprints seguintes seguiram um formato de especificação que utilizou além dos artefatos do modelo original, também o artefato de caso de uso, e então o mesmo se repetiu para as próximas três sprints porém, substituindo o caso de uso por diagrama de atividades.

Para a coleta de dados, foram elaboradas planilhas, baseadas no modelo de medição discutido na seção anterior. Todos os dados foram coletados pelo autor deste trabalho por meio de questionamento aos membros do time ou por ele mesmo em momentos específicos do processo de desenvolvimento onde o dado era gerado.

4.4. Interpretação

Devido ao tempo hábil para realizar a coleta, ao backlog do produto e às priorizações dos itens a serem desenvolvidos, apenas 9 sprints foram selecionadas para a coleta de métricas, deixando assim, a amostra de dados não grande o suficiente para a aplicação de testes estatísticos,

portanto, para a interpretação dos dados, foi feita uma análise descritiva das observações retiradas a partir dos dados.

Nesta seção é apresentada a interpretação dos dados obtidos pela etapa de coleta e averiguar as respostas para as questões levantadas no modelo de medição. Os gráficos com os dados coletados pelo estudo são apresentados nas **Figura 43**.

A questão 1 foi aplicada no modelo com o intuito de avaliar qual formato de apresentação da especificação proveria melhor qualidade funcional ao produto através da medição do número de requisitos que estavam sendo atendidos ou não após a conclusão da implementação, se algum requisito funcional deveria estar na especificação, porém foi esquecido, ou se algum defeito foi encontrado impedindo que a funcionalidade atendesse por completo as necessidades explícitas e implícitas do usuário.

Conforme ilustram os gráficos das métricas coletadas para a primeira questão (**Figura 43**), 78% das sprints tiveram um número baixo de requisitos (sprints 1, 2, 4, 5, 6, 8, 9) se comparado com as sprints com mais requisitos totais para implementação (sprints 3 e 7), e em 7 das 9 sprints, a taxa de requisitos atendidos foi de 100% (sprints 2, 3, 4, 5, 6, 8, 9), deixando apenas uma sprint que não utilizou artefato adicional na especificação e uma sprint que utilizou diagrama de atividades, com um requisito não atendido. Na sprint sem artefato adicional o requisito não atendido representa 33% de todos que foram alocados para desenvolvimento na sprint, enquanto que na sprint que utiliza diagrama de atividades, representa 6%.

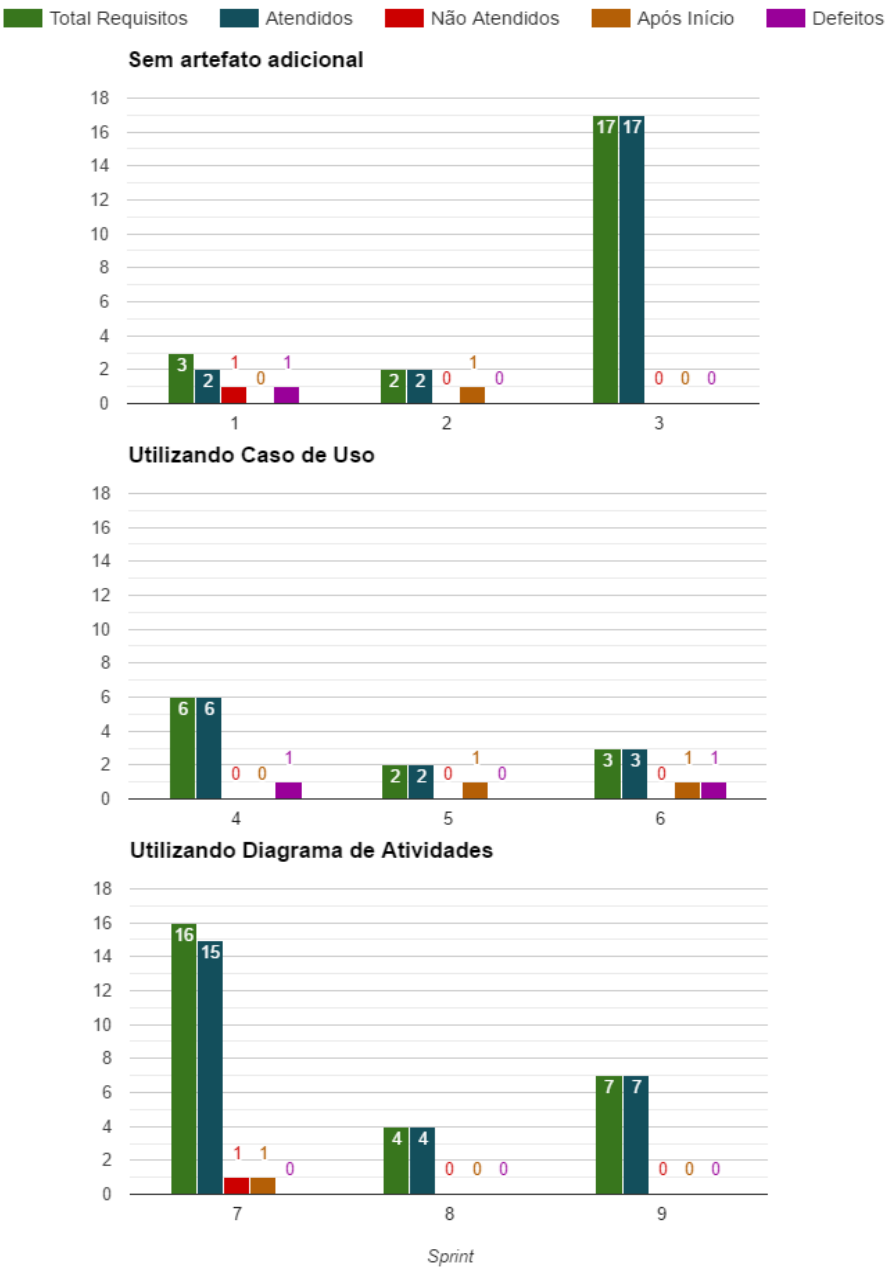


Figura 43. Métricas da questão 1

Para as sprints que apresentam requisitos não atendidos, no caso das amostras coletadas quando não se utiliza um artefato adicional na especificação (sprint 1), não há indícios de que existe uma correspondência entre a taxa de requisitos não atendidos e a quantidade total, pois as outras duas sprints com o mesmo modelo de apresentação da especificação, sprints 2 e 3, possuem respectivamente, quantidades menores (um requisito a menos) e bem maiores (quatorze requisitos a mais) na quantidade total de requisitos, porém com 100% dos requisitos atendidos. No caso das amostras coletadas quando se utiliza diagrama de atividades, os dados apontam para uma interdependência entre a taxa de requisitos não atendidos e a quantidade total já que a única sprint que apresentou requisito não atendido foi aquela com o maior número de requisitos para implementar (sprint 7). As 3 sprints em que se utiliza caso de uso não apresentam nenhuma medição com requisito não atendido e foi o modelo de apresentação da especificação com a menor média no total de requisitos por sprint (média de 4 requisitos por

sprint para amostras do grupo com caso de uso, e 7 e 9 para as amostras sem artefato adicional e diagrama de atividades respectivamente) e também foram mais similares, com a menor variabilidade na quantidade total de requisitos para implementar.

A métrica de quantidade de requisitos levantados após o início da implementação apresenta 4 novos requisitos durante a medição, sendo um na sprint 2 quando não se utiliza artefato adicional na especificação – valor que representa 50% da quantidade total de requisitos iniciais alocados para a sprint – dois nas sprints 5 e 6 quando se utiliza caso de uso – valor que representa 50% e 33% do total de requisitos da sprint respectivamente – e um na sprint 7 quando se utiliza diagrama de atividades – valor que representa 6% do total de requisitos da sprint. Essa métrica só mostrou uma forte ligação com a quantidade total de requisitos e o fato de serem atendidos ou não quando se utiliza diagrama de atividades na especificação, pois foi o único formato de apresentação da especificação onde essa métrica mostrou resultado relevante na sprint com maior quantidade de requisitos para implementar, enquanto para os outros formatos, a medida relevante só apareceu nas sprints com a menor quantidade de requisitos para implementar.

A questão 2 foi incluída no modelo para averiguar qual dos 3 formatos de apresentação da especificação propicia melhor qualidade estrutural ao produto através de métricas de quantidade de falhas encontradas na revisão de código, quantidade de testes automatizados implementados e a qualidade do código gerado medido nas escalas: “Bom” para código legível e de fácil compreensão; “Regular” para código não simples, porém de acordo com os requisitos; e “Ruim” para código gerado muito complexo e difícil de assimilar aos requisitos (**Figura 44**).

A métrica de falhas encontradas na revisão não está sendo apresentada pois todas as medidas coletadas foram nulas visto que o processo de revisão de código só é executado quando não existe programação pareada para implementação dos requisitos, e para todos os requisitos implementados durante a medição, o processo seguido foi através de programação pareada.

A métrica qualitativa sobre a qualidade do código gerado apresentou melhores resultados nas amostras que não utilizam artefato adicional no formato de apresentação da especificação visto que apresenta o maior percentual de avaliações na escala “Bom” (67%), que é a mesma proporção das amostras com caso de uso, porém as avaliações remanescentes estão na escala “Regular”, enquanto que as amostras que utilizam caso de uso, estão na escala “Ruim”. Essa métrica demonstra sofrer uma influência muito grande do código já existente no ambiente onde esse novo código é aplicado. As observações desta métrica descritas na planilha de coleta mostram que em todos os casos que a avaliação do código foi classificada como “Ruim” ou “Regular” estavam inseridas em um ambiente onde o código era legado.

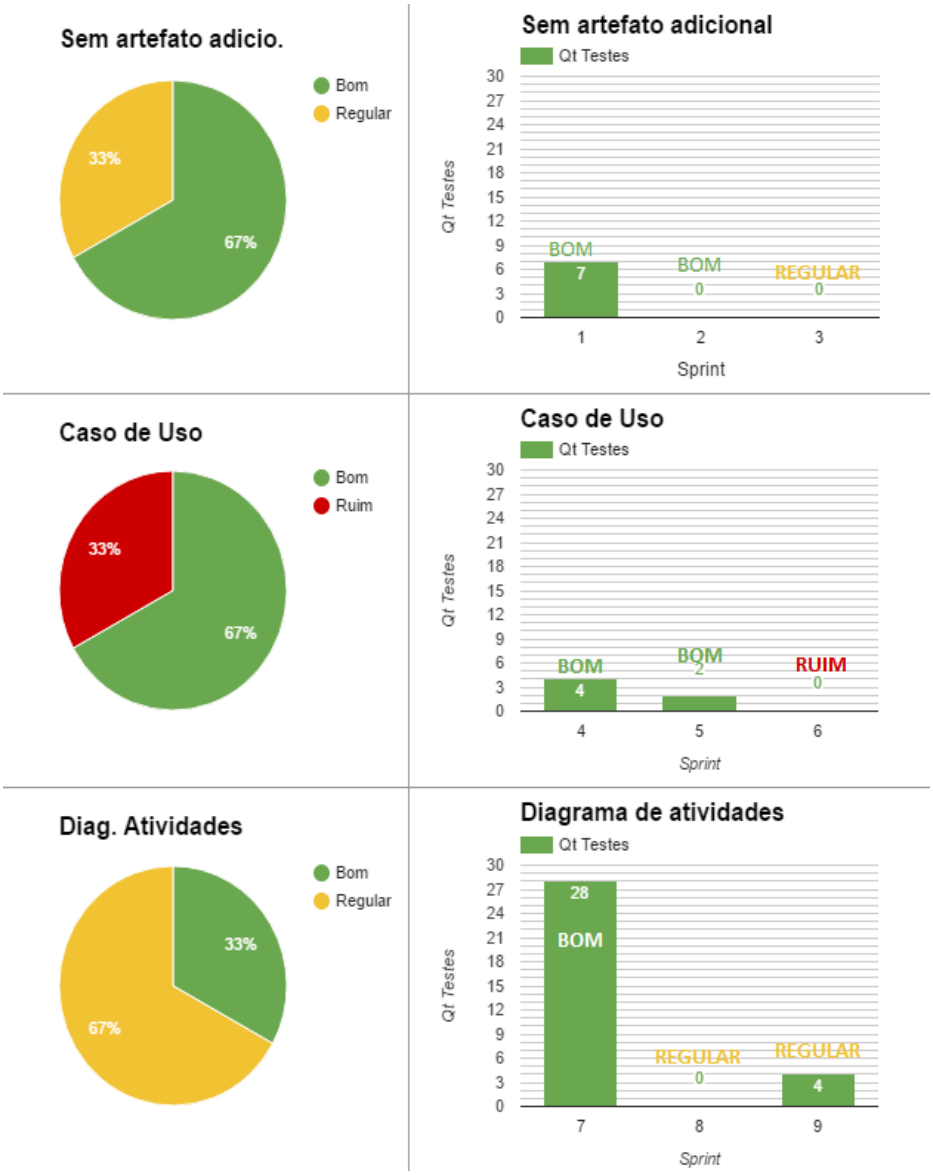


Figura 44. Métricas da questão 2

A métrica de quantidade de testes automatizados demonstrou receber o mesmo impacto da qualidade do código gerado quando se observa também o código já existente no ambiente onde os novos requisitos são implementados, visto que a única observação que descreve "Código novo" é a que apresenta uma quantidade bem elevada de testes automatizados se comparado com as outras amostras. Outra observação a se colocar é que essa mesma métrica não parece sofrer impacto sob a quantidade total de requisitos alocados para a sprint quando é visto que as sprints 3 e 7 são as amostras com a maior quantidade de requisitos (**Figura 43**), entretanto uma tem quantidade nula de testes automatizado e não utilizou artefato adicional na apresentação da especificação, enquanto a outra tem 28 testes automatizado e utilizou diagrama de atividades.

De maneira geral, as métricas retiradas na medição não trouxeram resultados muito conclusivos do ponto de vista do formato de se apresentar a especificação, e sua influência sobre as métricas de qualidade estrutural escolhidas para medição parecem ter pouco impacto.

A questão 3, com o intuito de avaliar a qualidade do processo o modelo de medição inclui 3 métricas que visam observar a precisão das estimativas de implementação geradas pelo time de desenvolvimento e o esforço utilizado para elaborar as especificações (**Figura 45**).

A primeira métrica mede a taxa de quantidade de histórias que tiveram suas estimativas alteradas após a conclusão da implementação. Todos os grupos observados apresentam pelo menos uma sprint na qual existiram histórias que tiveram suas estimativas alteradas, sendo o grupo que não utilizou nenhum artefato adicional no formato de apresentação da especificação com uma taxa de 57%, a maior de todas se comparado com os 33% do grupo que utiliza caso de uso e 15% o que utiliza diagrama de atividades.

A segunda métrica observa a diferença entre o tamanho das estimativas para implementar as histórias de usuário alocadas para sprint antes da implementação e após, e então calcula a taxa de alteração no valor da história. As amostras do grupo que não utiliza artefato adicional têm duas sprints que apresentam divergência entre o valor da estimativa nos dois momentos distintos da implementação, sendo a sprint 2 com uma taxa de -38% – redução de 13 story points para 8 – e a sprint 3 com uma taxa de -24% – redução de 25 story points para 19. Os outros dois grupos tem apenas uma sprint cada com alteração na estimativa inicial, com uma taxa de 67% de aumento na sprint 6 – de 3 story points para 5 – que utiliza caso de uso, e uma taxa de 15% na sprint 9 – aumento de 13 story points para 15 – que utiliza diagrama de atividades.

Do ponto de vista de esforço para especificar os requisitos, não há indícios de uma relação direta com a quantidade total de requisitos para os 3 grupos observados, pois pouco varia o esforço para diferentes quantidades de requisitos, porém é observável uma interdependência com a estimativa inicial em story points dada pelo time nos grupos que utilizam artefato adicional na especificação. Pode-se interpretar que o grupo que não utiliza um artefato adicional despende um esforço muito menor, principalmente na amostra coletada na sprint 3, em que o total de story points é o mais elevado dentre todas as amostras e o esforço está entre os mais baixos.

Embora exista esse esforço extra para especificar os requisitos utilizando um artefato adicional, a representatividade dele é muito baixa se comparada com o esforço total coletivo utilizado no processo de grooming. Nos dois grupos que utilizam artefato adicional, o esforço extra para especificar o requisito representa apenas 6,7% na amostra com a menor representatividade (sprint 8, com diagrama de atividades) e 16,7% na amostra com maior representatividade (sprint 6, com caso de uso).

O resultado dessas métrica pode ser interpretado de uma forma que indica que a utilização de um artefato adicional diminui o nível de incerteza no processo de estimativa e apresenta resultados com maior grau de precisão no processo de desenvolvimento, e a utilização do diagrama de atividades trouxe benefício visto que o maior esforço não está no momento de especificar o artefato que representa o comportamento do sistema e sim no processo de grooming.

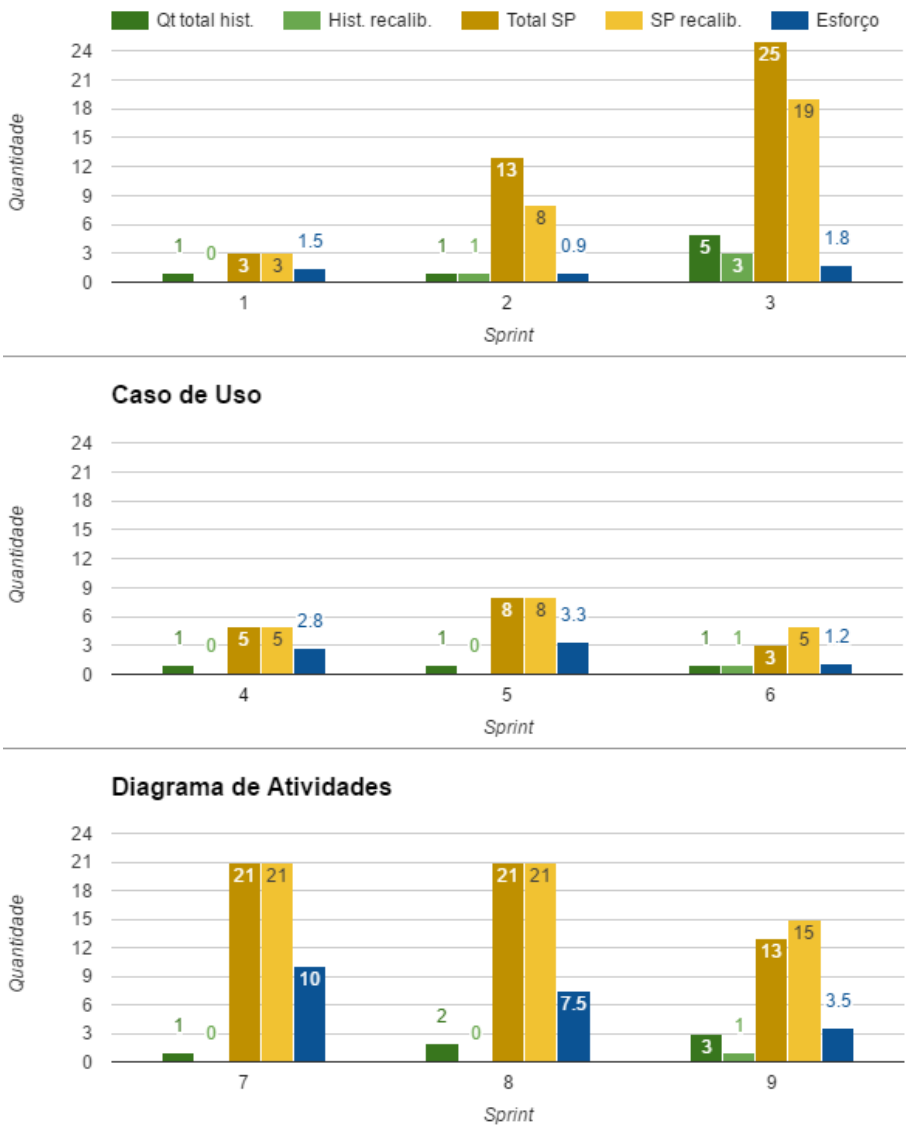


Figura 45. Métricas da questão 3

4.5. Ameaças a validade

Para a validação do estudo de caso, são consideradas algumas ameaças que podem afetar o resultado e a validade das conclusões retiradas a partir da interpretação dos dados coletados. Estas ameaças estão descritas nos parágrafos seguintes.

Devido ao tempo hábil para a coleta de dados, somente 3 sprints por grupo foram selecionadas, deixando a quantidade de amostras pequena para aplicação de testes estatísticos, portanto apenas a análise descritiva foi aplicada.

Por ser um produto já inserido no mercado, o esforço de implementação do time de desenvolvimento é competido por correção de bugs e evolução do sistema, portanto muitas das amostras coletadas foram de sprints com um backlog de requisitos para implementação muito pequeno, onde apenas poucas alterações em funcionalidades já existentes eram feitas, e poucas novas funcionalidades surgiram. Dessa forma, não existe uma consistência na quantidade de requisitos e tamanho das alterações realizadas entre as amostras tanto do mesmo grupo, quanto de grupos diferentes.

O autor do estudo de caso exerce suas funções profissionais dentro do time selecionado no estudo de caso e foi responsável por elaborar o modelo de medição, realizar a coleta de dados e também elaborar os artefatos adicionais das especificações de requisitos das amostras dos grupos que utilizaram caso de uso e diagrama de atividades. Por já estar inserido dentro do processo de desenvolvimento do estudo de caso diariamente, seu conhecimento implícito pode colaborar com o resultado das conclusões, porém pode afetar seu julgamento durante uma análise descritiva.

Tempo de casa de alguns integrantes do time: embora seja um time montado recentemente, dois dos integrantes que o compõe possuem grande conhecimento da camada de negócios do software que desenvolvem por trabalharem na organização por muitos anos. Esse conhecimento pode afetar as métricas coletadas, principalmente àquelas que tratam da qualidade funcional e precisão de estimativas.

Por ser um software que vem evoluindo por muito tempo, muita codificação existente está presente no software por muitos anos, implementado por pessoas que já não trabalham no projeto. Por ser uma grande quantidade, esse fator pode afetar as métricas relacionadas à qualidade estrutural, visto que grande parte dos requisitos implementados durante todo o estudo de caso trabalhavam com esses códigos antigos.

5. Conclusões

O primeiro impacto avaliado dentro do objetivo do trabalho foi sobre a qualidade funcional do produto e foi trazido através da questão “A necessidade que motivou o desenvolvimento da funcionalidade foi completamente atendida?”. A utilização do caso de uso aponta para um impacto negativo, pois mesmo com sprints mais consistentes e menores, demonstrou uma alta nos aspectos negativos medidos no estudo de caso, com exceção do número de requisitos não atendidos. Já a utilização do diagrama de atividades parece ter causado um impacto positivo, pois embora tenha obtido valores relevantes nos aspectos indesejados, estes só foram encontrados na amostra com maior densidade de requisitos, o que gera uma melhor previsibilidade e transparência do resultado, permitindo tomar ações preventivas para que isso não ocorra novamente, um princípio muito reforçado pelas práticas ágeis do Scrum, também conhecida com kaizen, prática adotada do Lean.

A pergunta do modelo de medição, “Qual a qualidade do código gerado para desenvolver a funcionalidade?” foi incluída para avaliar o impacto sobre a qualidade estrutural do produto. Com relação a essa questão, as medidas coletadas pelo estudo de caso apontam os melhores resultados também para os grupo que não utilizam artefato adicional ou utilizam diagrama de atividades, visto que um grupo gera um código de melhor qualidade e no outro grupo mais testes automatizados foram implementados, entretanto as observações incluídas no documento de coleta deixam um questionamento com relação ao ambiente onde será trabalhada a codificação dos novos requisitos (código legado) e então fica a dúvida se dentro do contexto do estudo de caso existiu de fato um impacto na qualidade estrutural causado pela alteração do formato de apresentação da especificação.

Por fim, a última pergunta do modelo, “Qual a precisão das estimativas levantadas no planejamento?” avalia o impacto do ponto de vista da qualidade do processo observando métricas de precisão nas estimativas e esforço para gerar os artefatos adicionais. Essa métrica demonstrou um resultado mais sólido e que aponta que o impacto mais positivo aparece quando é utilizado diagrama de atividades que trouxe à realidade a alternativa 2 colocada no modelo de medição, que espera que a taxa de alteração na estimativa das histórias de usuário seja menor. Embora exista o esforço adicional para elaborar o artefato, a representatividade deste

esforço se comparado ao esforço já despendido pelo modelo tradicional já executado pelo time do estudo de caso é muito baixa.

Concluindo, com os resultados obtidos no estudo de caso, a recomendação que ficou estabelecida para o time que participou do é a utilização do artefato de diagrama de atividades para especificar histórias de usuários com estimativas de valor alto (acima de 21 story points), pois pela interpretação dos resultados obtidos, foram essas amostras que demonstraram maior impacto positivo, já que as outras tiveram resultados inconclusivos ou impacto negativo, e continuar medindo e experimentando para se obter mais amostras e inspecionar se os resultados desse esforço permanecem ajudando a criar e evoluir o produto mantendo e melhorando a qualidade da entrega de valor.

7. Referências

Basili, V. et al. (2014) “Aligning Organizations Through Measurement: The GQM+Strategies Approach.” In:

BASILI, V. et al. Aligning Organizations Through Measurement: The GQM+Strategies Approach. [S.l.]: [s.n.], 2014.

BASILI, V.; CALDIERA, G.; ROMBACH, D. Goal Question Metric Paradigm. In: MARCINIAK, J. J. Encyclopedia of Software Engineering. [S.l.]: Wiley-Interscience, v. 2, 2001.

BECK, K. et al. Agile Manifesto. Manifesto for Agile Software Development, 2001. Disponivel em: <http://agilemanifesto.org/>.

CHAPPELL, D. David Chappell. David Chappell & Associates, 201? Disponivel em: <http://www.davidchappell.com/writing/white_papers.php>. Acesso em: 03 nov. 2016.

ROCHA, A. R. C. D.; SOUZA, G. D. S.; BARCELLOS, M. P. Medição de Software e Controle Estatístico de Processos. Brasília: Ministério da Ciência, Tecnologia e Inovação, 2012.

SOLINGEN, R.; BERGHOUT, E. The Goal/Question/Metric Method: a practical guide for quality improvement of software development. [S.l.]: David Hatter, 1999.

SUTHERLAND, J.; SCHWABER, K. The Scrum Guide: The Definitive Guide to Scrum. [S.l.]: Scrum.org and ScrumInc, 2013.