UNIVERSIDADE SÃO JUDAS TADEU
MBA EM GESTÃO DE PROJETOS

FÁBIO REIS SANTOS

AS TÉCNICAS DE GERENCIAMENTO DE ESCOPO DO PROJETO E A LIDERANÇA DE TIMES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DAS TÉCNICAS E FERRAMENTAS DE GERENCIAMENTO DE ESCOPO DO PROJETO VOLTADAS PARA ÁREA DE DESENVOLVIMENTO DE SOFTWARE.

TRABALHO DE CONCLUSÃO DE CURSO

SÃO PAULO
2018

FÁBIO REIS SANTOS

AS TÉCNICAS DE GERENCIAMENTO DE ESCOPO DO PROJETO E A LIDERANÇA DE TIMES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DAS TÉCNICAS E FERRAMENTAS DE GERENCIAMENTO DE ESCOPO DO PROJETO VOLTADAS PARA ÁREA DE DESENVOLVIMENTO DE SOFTWARE.

Trabalho de Conclusão de Curso do MBA em Gestão de Projetos com ênfase em PMI pela Universidade São Judas Tadeu, unidade Mooca como requisito para conclusão do curso.

Orientador: Prof. Sergio Bonato, PMP

SÃO PAULO

2018

A simplicidade é o último grau da sofisticação (Leonardo DaVinci)

RESUMO

SANTOS, Fábio Reis. AS TÉCNICAS DE GERENCIAMENTO DE ESCOPO DO PROJETO E A LIDERANÇA DE TIMES DE DESENVOLVIMENTO de software: Um estudo das técnicas e ferramentas de gerenciamento de escopo do projeto voltadas para área de desenvolvimento de software. 2018. XX f. Monografia (MBA em Gestão de Projetos com ênfase no PMI) – Secretaria de Pós-Graduação Lato Senso, Universidade São Judas Tadeu. São Paulo, 2018.

Este trabalho consiste no levantamento de metodologias e *frameworks* utilizados em projetos de desenvolvimento de software através da pesquisa e do aprofundamento na literatura especializada. O trabalho, além de explicar sucintamente as origens históricas, os requisitos, as restrições e os benefícios da utilização dessas metodologias e *frameworks*, visa discutir a aplicabilidade destas no contexto do gerenciamento de projetos. O trabalho ainda discute as categorizações dessas metodologias e *frameworks*, e debate como elas podem ou não ser combinadas entre si. O objetivo da apresentação desse panorama levantado foi instrumentalizar os gerentes de projetos e profissionais de Tecnologia da Informação que são responsáveis por liderar projetos com um vasto conhecimento sobre essas metodologias e *frameworks* para que considerem usar as técnicas e ferramentas relatadas nesse trabalho que sejam mais adequadas para cada situação. Objetivo em prol do profissional na posição de liderança de um projeto de desenvolvimento de software fazer sempre o melhor trabalho no gerenciamento de escopo, do projeto e do produto.

Palavras-chave: Gerenciamento de Projetos, Desenvolvimento de Software, Tecnologia da Informação, Metodologias de Desenvolvimento, Escopo.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CPM	*Counting Practices Manual*
EAP	Estrutura Analítica de Projeto
IBM	*International Business Machines*
IFPUG	*International Function Point Users Group*
JAD	*Joint application design*
LOC	*Lines of Code*
MBA	*Master in Business Administration*
MDA	*model-Driven architecture*
MDE	*model-based engineering*
OMG	*Object Management Group*
PF	Pontos de Função
PMBOK	*Project Management Body of Knowledge*
PMI	*Project Management Institute*
PO	*Product Owner*
RUP	*Rational Unified Process*
TI	Tecnologia da Informação
UI	*User Interface*
UX	*User Experience*
WBS	*Work Breakdown Structuren*
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Este trabalho de conclusão de curso abordará o assunto do gerenciamento de escopo no contexto de projetos de desenvolvimento de software buscando técnicas que trazem muitos benefícios nesse âmbito que não necessariamente estejam no PMBOK.

As ferramentas e técnicas usadas em um projeto podem variar, mediante o conhecimento de sues executores e as necessidades do projeto. O *Project Management Institute* (PMI, 2013, p. 2) diz que "'Boa prática' não significa que o conhecimento descrito deva ser sempre aplicado uniformemente a todos os projetos; a organização e/ou a equipe de gerenciamento do projeto é responsável por determinar o que é apropriado para um projeto específico". Por isso esse trabalho trata também de técnicas não incluídas no PBMOK que podem ser úteis nos processos relacionados ao Gerenciamento de Escopo do Projeto.

Escopo é de vital importância no gerenciamento de projetos. É no escopo que encontra-se os requisitos. E como descreve o PMI (PMI, 2013, p. 112):

> O sucesso do projeto é diretamente influenciado pelo envolvimento ativo das partes interessadas na descoberta e decomposição das necessidades em requisitos, e pelo cuidado tomado na determinação, documentação e gerenciamento dos requisitos do produto, serviço ou resultado do projeto.

Escopo é algo fundamental para o êxito de projeto tanto para o fornecedor tanto para o cliente. O trabalho aborda o assunto, como aborda o PMBOK, pela ótica do executor do projeto e não do contratante. Mesmo assim, este trabalho também é válido para o contratante, para saber como seu fornecedor pode estar conduzindo o escopo do projeto e quais as técnicas e ferramentas ele possivelmente utilizará pra isso.

Para o cliente este conhecimento pode estreitar o dialogo, aumentar o sentimento de segurança e dar mais eficácia ao discurso quando algo precisar ser debatido ou revisto no meio do desenvolvimento. Ao executor do projeto, as técnicas e ferramentas, assim como o gerenciamento do escopo, são vitais para que o trabalho seja controlado e medido, e com isso amplie as margens de sucesso e gere informações para a reflexão e aprimoramento da gestão.

Ferramentas e técnicas são maneiras como se externalizou o conhecimento e se utiliza isso na esperança de retirar o fardo do gestor de guardar e processar toda a informação. Colocar ferramentas promove um grande aumento na produtividade do gerenciamento.

Logo, quem quer fazer um gerenciamento com qualidade deve conhecer as técnicas e ferramentas de gerenciamento de escopo certas e apropriadas para cada caso e usá-las bem e é disso que trata este trabalho.

Este trabalho foca exclusivamente projetos de software e terá uma análise direcionada sobre as técnicas e ferramentas de gerenciamento de escopo em projetos de desenvolvimento de software e no trabalho do time ou equipe de desenvolvimento.

Quando profissionais de TI pensam na gestão de times de desenvolvimento de software nem sempre consideramos apenas no cargo de gerente de projetos nessa gestão, podem ser também *scrum masters* ou até líderes técnicos que assumem esse papel. Como os projetos de software são vistos como mais simples por envolver essencialmente programação, muitos desses profissionais que estão nessas posições, de guiar equipes, nem sempre têm todo ferramental adequado para a sua boa atuação.

Projetos de software podemos dividir entre projetos genéricos e projetos e produtos sob encomenda. Como explica o professor Sommerville (2011, p. 4) “em softwares genéricos, a organização que o desenvolve controla sua especificação. Para produtos sob encomenda, a especificação é normalmente desenvolvida pela empresa que está adquirindo o software”. Portando o foco deste trabalho é nos projetos sob encomenda. Softwares genéricos envolvem projetos e se beneficiam das técnicas que trataremos aqui, mas as variáveis envolvidas são mais controladas e previsíveis. Além disso, nos casos de desenvolvimento de software genérico, geralmente já existe um padrão de desenvolvimento estabelecido na organização que o desenvolve.

Este trabalho apresenta algumas das técnicas ferramentas para fazer o gerenciamento de escopo e traz uma discussão crítica sobre a importância dessas técnicas e ferramentas para o desenvolvimento de software em vista do benefício que essas técnicas e ferramentas proporcionam.

1.1 Objetivo Geral

Expor um panorama das técnicas e ferramentas utilizadas nos processos de Gerenciamento de Escopo do Projeto em projetos de desenvolvimento de software.

Com este estudo pretende-se organizar de forma fácil e explicativa essas técnicas e ferramentas de modo que o material possa ser utilizado como uma referência prática de estudo para gestores na área de desenvolvimento de software que procuram instrumentalizar melhor o seu dia a dia.

1.2 Objetivos Específicos

- Produzir uma análise de técnicas e ferramentas relacionadas com o Gerenciamento de Escopo no contexto do desenvolvimento de software.
- Apresentar essas técnicas e ferramentas descrevendo sucintamente como funcionam, como utilizá-las no dia a dia e os benefícios atingidos quando utilizadas.
- Explicar de forma concisa e clara o assunto para que promova uma leitura agradável, rápida e fácil, voltada à líderes e gestores que tem tempo limitado em seus cotidianos.
- Apresentar algumas técnicas e ferramentas não citadas no PMBOK relacionando com os processos de gerenciamento de escopo em projetos de desenvolvimento de software.

1.3 Justificativa

O trabalho é dirigido para Gestores, Líderes e estudantes na área de TI que buscam informações sobre técnicas e ferramentas do Gerenciamento de Escopo do Projeto.

Partindo do conhecimento geral sobre Gestão de Projeto, sabemos o quanto são fundamentais as técnicas e ferramentas. O PMBOK cita algumas ferramentas, mas não tem a pretensão de explicá-las, ele indica que outras ferramentas podem ser somadas à gestão, mas não lista essas ferramentas nem aponta caminhos onde descobri-las e quando utilizá-las.

Este trabalho será construído nesta lacuna, explicando pontualmente as técnicas e ferramentas, além de ampliar o espectro de instrumentos que podem ser usados na gestão de Escopo do Projeto, detalhando como e quando utilizar algumas técnicas voltadas para o gerenciamento de escopo de projetos de desenvolvimento de software.

Sem ter a pretensão de cobrir tudo que existe disponível, o trabalho facilitará o estudo do assunto organizando numa forma fácil e prática algumas das principais técnicas e

ferramentas do gerenciamento de escopo no âmbito do desenvolvimento de software que podem ser uma descoberta para o leitor e trazer mais produtividade na gestão.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 PMBOK

O principal material de consulta e estudo na área de gerenciamento de projetos é o PMBOK. Este guia é principal material teórico sobre gestão de projetos. Um dos objetivos dele é padronizar e unificar a linguagem entre gestores de projeto. Como o autor Xavier (2006, p. 2) explica “O novo ambiente de negócios, caracterizado por uma grande pulverização de equipes, cada vez mais distantes geograficamente, apresenta um desafio gerencial” e continua “A resposta a esse desafio passa pelo desenvolvimento de técnicas, metodologias, tecnologias e as melhores práticas de gerenciamento de software”. Todavia, Se existe preocupações sobre as quantidades, passos e demais detalhes descritos no PMBOK, o próprio guia do PMI explica, como na citação sobre “Boa prática” feita no capítulo Introdução deste trabalho, que o material deve ser utilizado de acordo com as necessidades do projeto, portanto algumas preocupações e enfoque correntes que se vêm por aí não são sempre necessárias (2013).

Com o desenvolvimento deste trabalho vamos facilitar o acesso a certas informações do PMBOK, detalhar algumas informações extraídas e simplificar outras, além de incluir mais informações de técnicas específicas para o âmbito de projetos de desenvolvimento de software que vão além do guia. Entende-se, como princípio, que o PMBOK é um ponto de partida para alguns conceitos gerais que relacionamos com as técnicas e ferramentas e não um ponto de chegada.

2.2 Definições do PMBOK.

2.2.1 Escopo

Entende-se pelo conjunto de todo o trabalho requerido (e apenas o trabalho requerido) para completar satisfatoriamente um projeto. E no contexto de projeto, o PMBOK classifica o escopo em Escopo do Produto e Escopo do Projeto.

2.2.2 Escopo do Produto

Corresponde as características e funções de um produto, serviço ou resultado. “As características ou funções que devem ser incluídas no produto ou serviço, representando através dos ‘requisitos’” (XAVIER, 2006, p. 35).

2.2.3 Escopo de Projeto

Especifica o trabalho realizado para entregar o escopo do produto e sendo assim, em algumas situações pode incluir a definição do escopo do produto. Segundo Xavier (2006, p. 35) o escopo do projeto é “O trabalho que deve ser executado, a fim de fornecer um produto com as características e funções especificadas, representado em uma WBS”. Em outras palavras, Escopo do Projeto é mais abrangente que Escopo do Produto.

2.2.4 Gerenciamento do Escopo do Projeto

Como define o PMI (2013, p. 105) “O gerenciamento do escopo do projeto está relacionado principalmente com a definição e controle do que está e do que não está incluso no projeto”.

2.2.5 Linha Base

As ferramentas e técnicas podem variar de projeto a projeto, mas como o PMI (2013) define uma linha base de um escopo é composta de declaração mais um WBS (com seu dicionário associado) aprovados, pelo solicitante.

2.2.6 Os macroprocessos

Os macroprocessos do gerenciamento de escopo pelo PMBOK são:

- Planejar o gerenciamento do escopo
- Coletar os requisitos
- Definir o escopo

- Criar a estrutura analítica do projeto (EAP)
- Validar o escopo
- Controlar o escopo

Dentro de cada macroprocesso existe uma lista de técnicas e ferramentas abordadas pelo PMBOK. Este trabalho dará mais destaque para técnicas e ferramentas que estão fora dessa lista e que têm uma relação mais forte com desenvolvimento de software.

A lista com mais itens está no macroprocesso Coletar requisitos, o trabalho vai abordar esse macroprocesso no próximo capítulo como também os macroprocessos de Controlar o escopo, Validar o escopo e Definir o escopo. Esses últimos processos têm uma lista menor de técnicas e ferramentas no PMBOK. Este trabalho anseia ampliar o repertório de técnicas e ferramentas para o gerenciamento de escopo em projetos de desenvolvimento de software nesses processos. A partir das descrições feitas a partir do próximo capítulo.

2.3 Engenharia de Software

Academicamente, as técnicas e ferramentas do desenvolvimento de software fazem parte do campo da Engenharia de software. “Engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios inicias da especificação do sistema até sua manutenção” (SOMMERVILLE, 2011, p. 5). A abrangência desta disciplina chega em todos os níveis de um projeto de desenvolvimento de software, “engenharia de software não se preocupa apenas com os processos técnicos do desenvolvimento de software. Ela também inclui atividades como gerenciamento de projetos de software e desenvolvimento de ferramentas, métodos e teorias” (SOMMERVILLE, 2011, p. 5).

2.3.1 Modelos de processos de software

Assim Sommerville (2011) chama os ‘paradigmas de processo’ no segundo capítulo do seu livro Engenharia de software. Esses modelos são abstrações de diferentes abordagens do desenvolvimento de software, e não são excludentes podendo coexistir em projetos complexos.

Sommerville aborda três modelos. Vamos comentar aqui dois desses modelos, que são os mais conhecidos, e porque eles aparecem explicitamente e implicitamente no restante do trabalho.

Castaca ou *waterfall*: Este foi o primeiro modelo de desenvolvimento formalmente definido. Sua execução é linear, contemplando as atividades fundamentais do desenvolvimento, como demostra a figura a seguir:

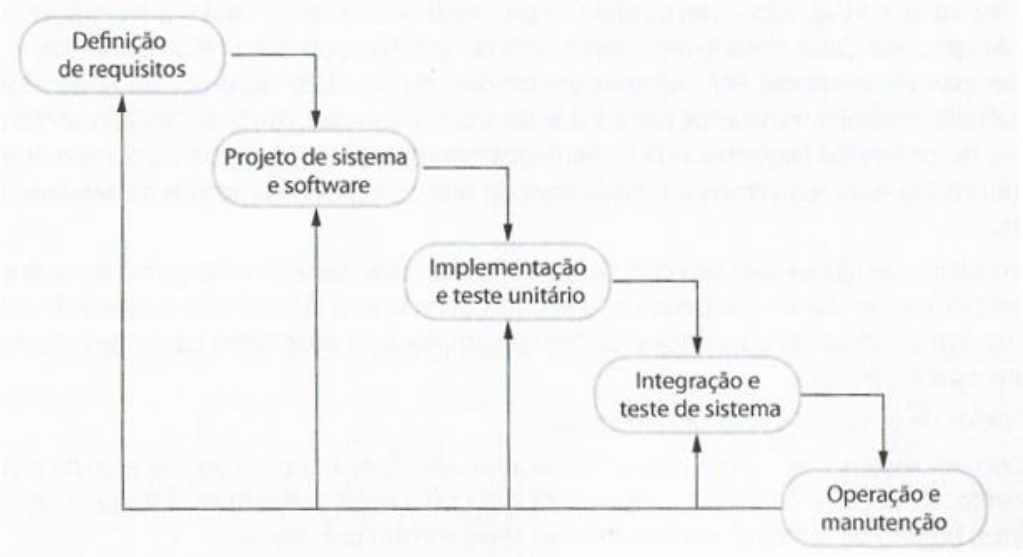

Figura 1 - Diagrama do Modelo Cascata
Fonte: Sommerville (2011).

Para seguir para a fase seguinte a fase anterior deve estar totalmente concluída e aprovada. Esse processo também é chamado de processo orientado a planos. Um grande problema nesse modelo é que se for identificada uma mudança de escopo após a fase de planejamento todo o projeto deve ser reiniciado, isso traz grandes riscos ao prazo do projeto.

"Em princípio, o modelo em cascata deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento do sistema" (SOMMERVILLE, 2011, p. 21). Por conta da uniformidade e da segurança propiciada por esse modelo ele ainda é utilizado em projetos com grande necessidade formal e determinação de resultado, como em projetos de software para aeronaves e jogos.

O desenvolvimento incremental "é baseado na ideia de desenvolver uma implementação inicial e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido" (SOMMERVILLE, 2011, p. 22). Este modelo é fundamental para as abordagens ágeis. Ele está presente em projetos para softwares de sistemas de

negócios, *e-commerce* e sistemas pessoais. Entre as vantagens está que ele é mais rápido e mais barato para fazer mudanças no software durante o desenvolvimento, mais fácil receber os *feedbacks* do cliente porque avaliarão o produto em si, o que foi implementado, e não uma documentação do projeto de sistema, um relatório de desenvolvimento, e o cliente pode utilizar o software, entregando valor mais rápido ao cliente, antes que esteja concluído.

Também existem desvantagens no modelo incremental: Como a entrega é mais rápida e dinâmica fica mais difícil elaborar uma documentação completa do sistema no tempo das entregas, como o sistema é frequentemente mexido, existe o risco de uma atualização prejudicar a utilização do que já foi entregue.

> Os problemas do desenvolvimento incremental são particularmente críticos para os sistemas de vida-longa, grandes e complexos […]. Sistemas de grande porte necessitam de um *framework* estável […]. Isso deve ser planejado com antecedência, e não desenvolvido de forma incremental.

O que já nos diz o conceito básico de modelos de processos de software é que não há resposta certa, ou solução mágica que sirva para todos os casos, o que existe são diversas opções de técnicas, ferramentas e modelos. Algumas opções são mais adequadas em determinados casos. O trabalho apresenta algumas dessas opções e faz essa discussão das suas aplicabilidades ao longo do texto, retomando essa questão das escolhas com mais detalhes nas Considerações Finais.

3 TÉCNICAS E FERRAMENTAS PARA O DESENVOLVIMENTO DE SOFTWARE

Nos subcapítulos seguintes serão abordadas as técnicas e ferramentas que não estão no PMBOK, mas são práticas consagradas de mercado que trazem benefícios para a gestão de projetos de gerenciamento de software além de comentar algumas técnicas e ferramentas listadas no PMBOK.

3.1 Levantamento de Requisitos

Este processo é crítico no gerenciamento de requisitos. É preciso ter conhecimento para diferenciar uma mudança de escopo de um trabalho de levantamento de requisitos mal feito, como salienta o trecho do livro Engenharia de Requisitos:

> Tenta-se justificar atrasos por motivos de mudança no escopo, quando trata-se de falha na Engenharia de Requisitos. O escopo em termos específicos é produto, não um insumo que limite a Engenharia de Requisitos. O escopo que limita a Engenharia de Requisitos são áreas funcionais, processos em mais alto nível e partes interessadas chave que afetam ou são afetados pelo desenvolvimento" (VAZQUEZ, 2016, p. X).

O PMI é generoso neste assunto trazendo algumas técnicas e ferramentas. E também nos ajuda na definição de requisito:

> Os requisitos incluem condições ou capacidades que devem ser atendidas pelo projeto ou estar presentes no produto, serviço ou resultado para cumprir um acordo ou outra especificação formalmente imposta. Os requisitos incluem as necessidades quantificadas e documentadas e as expectativas do patrocinador, cliente e outras partes interessadas. […] Requisitos se transformam na fundamentação da EAP (PMI, 2013, p. 112).

O autor Xavier, por sua vez, faz uma observação interessante sobre os requisitos e contextualiza o assunto para a área de desenvolvimento de software:

> Algumas vezes, no início do projeto não existe um detalhamento muito grande do produto, fazendo parte do escopo do projeto elaborar esse detalhamento. Isso é muito comum, por exemplo, em projetos de desenvolvimento de software, em que o levantamento dos requisitos do usuário é feito no decorrer do procedimento (XAVIER, 2006, p. 40).

O PMI divide tipos de requisito entre soluções de negócio e técnicos e explica de uma forma muito simples “a primeira se refere às necessidades das partes interessadas e a última a como essas necessidades serão implementadas” (PMI, 2013, p. 112). Em análise de sistemas existe a divisão entre Requisitos de usuário e de sistema que são equivalente a essa divisão do PMI.

Com detalhado a seguir:

> 1. Requisitos de usuário são declarações, em uma linguagem natural com diagrama, de quais serviços o sistema deverá fornecer a seus usuários e as restrições com as quais este deve operar.
> 2. Requisitos de sistema são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software. O documento de requisito de sistema (às vezes, chamado de especificação funcional) deve definir exatamente o que deve ser implementado (SOMMERVILLE, 2011, p. 58).

Uma outra divisão comum é entre requisitos funcionais e não funcionais. Um estudo mais a fundo faz-se necessário para internalizar esses conceito. A seguir algumas definições rápidas dessas classificações.

Requisitos funcionais “variam de requisitos gerais, que abrangem o que o sistema deve fazer, até requisitos muito específicos, que reflitam os sistemas e as formas de trabalho em uma organização” (SOMMERVILLE, 2011, p. 59).

Requisitos não funcionais refletem necessidades mais externas do sistema, que podem restringir, guiar ou até determinar requisitos funcionais. Como explica Sommerville (2011 p. 60 e 61):

> Eles podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, tempo de resposta e ocupação de área […] Os requisitos não funcionais surgem por meio das necessidades dos usuários, devido a restrições de orçamento, políticas organizacionais, necessidade de interoperabilidade com outros sistemas de software ou hardware, ou a partir de fatores externos, como regulamentos de segurança ou legislações de privacidade

Para finalizar a apresentação dessas classificações dos requisitos, está reproduzido a seguir os tipos de requisitos não-funcionais mostrados em um diagrama no livro Engenharia de software (SOMMERVILLE, 2011, p. 61).

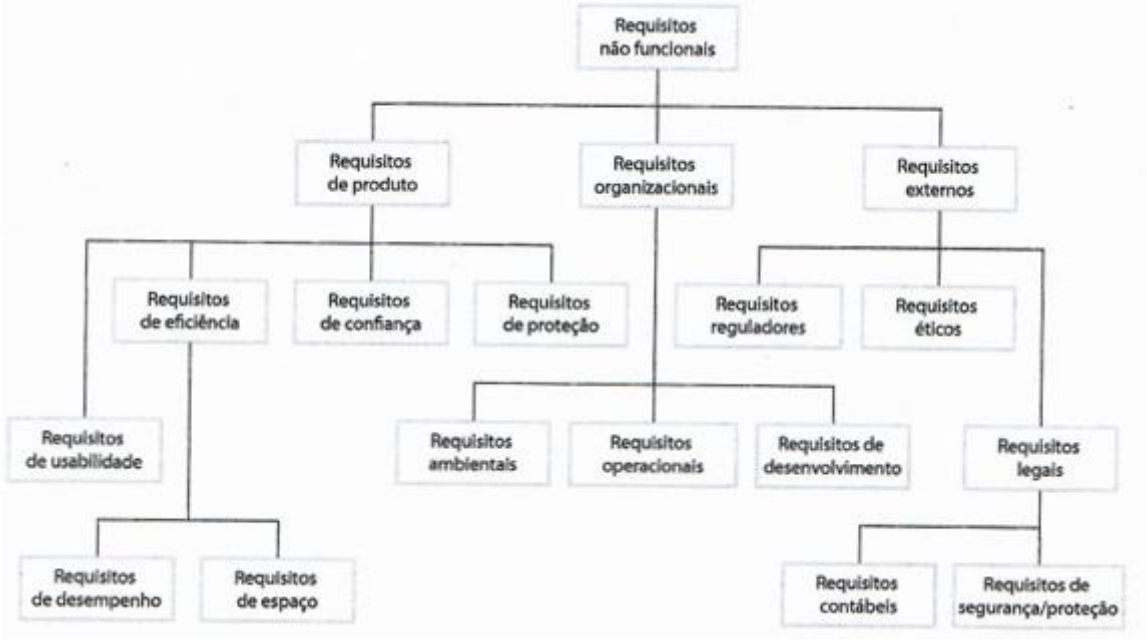

Figura 2 – Tipos de requisitos não funcionais
Fonte: Sommerville (2011).

Entender e usar essas classificações significa elevar a inteligência aplicada na gestão do desenvolvimento, através de ferramentas conceituais que estão a disposição, isso torna o sistema mais facilmente compreendido, mais organizado, e por ser algo comum, outras pessoas vão compreendê-lo utilizando essas mesmas classificações. A medida que esses conceitos tornam-se mais naturais a dificuldade de utilizá-los desaparece e fica apenas os benefícios de pensar de uma maneira que vai facilitar a compreensão de um sistema através de sua decomposição.

As primeiras técnicas que o PBMOK traz, nesse âmbito, são Entrevistas, Grupos de Discussão e Oficinas facilitadas, com destaque para a *Joint application design* (JAD) que é uma prática dos projetos de desenvolvimento de software. O que elas têm em comum é que são instrumentos de prospecção para utilizar no cliente para extrair os requisitos e podemos incluir aqui também Questionários e pesquisas.

O que o PMBOK não fala é que pessoas do cliente precisam ser eleitas para esses momentos. A pessoa que participa disso decide algo para a empresa toda, significa responsabilidade. E nem sempre essa pessoa estará comprometida com o sucesso do projeto. Isso deve passar pela percepção do gerente do projeto. A escolha dessa pessoa é dividida entre os dois lados, porque o gerente de projetos pode fazer recomendações a respeito.

Para concluir a reflexão sobre prospecção vamos adiantar um pouco na lista do PBMOK e comentar de uma técnica importante, a Observação, com as palavras do PMI:

> A observação é também conhecida como "*job shadowing*." (aprendizagem por observação). É normalmente feita externamente por um observador acompanhando um especialista de negócios na execução do seu trabalho. Também pode ser feita por um 'observador participante' que de fato realiza um processo ou procedimento para experimentar como o mesmo é feito e descobrir requisitos ocultos (PMI, 2013, p. 116).

Não é em todo projeto que essa tarefa terá relevância, mas isso aponta para algo importante neste momento que é ver e procurar além de apenas receber os requisitos.

O PBMOK ainda trata de outras técnicas ligadas aos requisitos. Um conjunto mais voltado para a equipe do projeto são as técnicas de criatividade em grupo e inclui-se também a prototipação, neste momento. Interessantes, mas não vamos comentá-las individualmente. Apenas quero salientar que esse tipo de técnica pode ser parte de algo maior que chamamos de metodologias e falaremos das metodologias a seguir, começando pelo chamado Análise dos Pontos de Função.

3.2 Pontos de Função

O objetivo desta técnica é medir um software, e para fazer isso temos que entender a motivação para medir. Para planejar um projeto a dimensão do produto é um elemento que precisa ser considerado, "é de suma importância que a definição de meios para a comparação do progresso real com o planejado seja parte do planejamento do projeto" (VAZQUEZ; SIMÕES; ALBERT, 2015, p. 19) portanto a Análise de Pontos de Função é uma opção à ser considerada, tanto para estimar como para controlar o projeto.

> A análise de pontos de função permite não só medir o tamanho do sistema em termos da funcionalidade fornecida ao usuário, mas também estimar seu tamanho em qualquer fase do seu ciclo de vida (mesmo que os requisitos ainda não tenham sido detalhados). […] durante a especificação de requisitos, projeto de alto nível, projeto detalhado e implementação, toda medição funcional realizada ainda é uma projeção do que de fato será entregue. […] pois normalmente entre o final do projeto detalhado e a conclusão do projeto ainda pode haver mudanças no escopo por mudanças nos requisitos e necessidades que originaram o projeto (VAZQUEZ; SIMÕES; ALBERT, 2015, p. 19).

Essa técnica surgiu na IBM no início dos anos 70, sendo uma alternativa a medição por linhas de código (LOC). A medição por linha de código tem uma série de furos, nem

sempre revelando dados importantes do sistema, podendo contar repetições de funções, comentários. Além disso é uma técnica que não dá informações novas para o início do projeto, apenas por comparação com projetos semelhantes.

Quem a desenvolveu foi Allan Albrecht, que recebeu a missão de avaliar a produtividade dos projetos de desenvolvimentos de software de uma unidade da IBM. Como esses softwares tinham sido escritos em linguagens de programação diversas a LOC não foi uma alternativa viável, cabendo ao Allan desenvolver uma técnica que permitisse avaliar a dimensão de diferentes software numa mesma escala. A técnica foi apresentada a comunidade em 1979 em um artigo. Capers Jones foi realizou estudos destacando a importância da APF. Em 1986 a *International Function Point Users Groups* (IFPUG) foi fundada.

O IFPUG é responsável por publicar seu Manual de Práticas de Contagem, que atualmente está na sua versão 4.3.1[1]. A fundação é uma organização sem fins lucrativos com o propósito de difundir boas práticas do gerenciamento de processos e desenvolvimento e manutenção de software envolvendo pontos de função e outras técnicas de medição, o seu manual é disponibilizado gratuitamente para seus afiliados pagantes e vendido para não afiliados. Com a popularização existem outras organizações que apresentam seus próprios manuais de utilização da técnica, que os oferecem gratuitamente, como COSMIC[2].

“Os métodos de medição permitem expressar quantitativamente qualidades de um objeto ou fenômeno. […] De maneira análoga, software também tem uma diversidade de dimensões” (VAZQUEZ; SIMÕES; ALBERT, 2015, p. 47). O manual que serve de referência aos comentários a seguir é o CPM da IFPUG explicado por Vazquez, Simões e Albert (2015). O que se mede é o requisito funcional do usuário que recebe um tamanho funcional. O primeiro passo é reunir todos os requisitos. Observando que para a técnica um usuário não representa apenas pessoas, “Um usuário é qualquer pessoa ou coisa que interaja (envia ou receba dados) com a aplicação” (VAZQUEZ; SIMÕES; ALBERT, 2015, p. 56)”

Os requisitos são decompostos em funções, divididas em dois tipos: De Dados e de Transição. “O método define referências para a identificação de cada componente funcional básico, sua classificação quanto ao tipo de função e à complexidade funcional e a determinação de sua contribuição individual” (VAZQUEZ; SIMÕES; ALBERT, 2015, p. 48). É da somatória desses componentes funcionais que se atinge o tamanho funcional e o último passo desse ciclo é documentar essa conta.

1 http://www.apfmetricas.com.br/ifpug.html
2 https://cosmic-sizing.org/

Essa conta se repete nas fases do projeto porque o nível de detalhamento dos requisitos vai aumentando a medida que o projeto vai avançando. Isso não desqualifica a técnica, é apenas algo natural do desenvolvimento. Apesar dela ser dependente dos requisitos, existem técnicas que permitem utilizá-la para estimar custo, tempo e esforço ainda nas fases principiais do projeto, como explicado a seguir.

> Embora o projeto ainda que esteja em um estágio de concepção ou análise de viabilidade […] alguns requisitos são conhecidos de maneira mais ou menos parcial. Com base nessas informações é possível estimar o tamanho em PF a partir de várias técnicas […] então é possível derivar as estimativas de esforço, custo, prazo (VAZQUEZ; SIMÕES; ALBERT, 2015, p. 52)

Algumas diretrizes para aplicação dessa técnica incluem que haja um tempo planejado no projeto para a medição e uma pessoa dedicada e treinada nessa tarefa. Caso haja interesse é recomendado que se dedique um tempo para uma capacitação adequada dos recursos, para que a técnica traga vantagens sendo aplicada corretamente e não prejuízos decorrentes a erros em sua utilização.

Outra diretriz importante é a qualidade de requisitos. A coleta de requisitos precisa ser uma atividade bem madura da equipe porque a qualidade dos requisitos impacta diretamente da qualidade da análise dos pontos de função.

Como comentado, essa técnica tem utilidade tando no planejamento quanto no controle do escopo, controlando andamento e mudanças. Ela possui poucas dependências, apenas requisitos bem descritos. Ela é altamente formal, mas conta com organizações dispostas a ajudar no entendimento e utilização. Ela é agnóstica em termos de tipos de software e linguagens de programação. Por todos esses motivos é uma ferramenta versátil que vai trazer benefícios nos projetos de desenvolvimento de software.

3.3 O RUP

Machado (2106) conta que O Processo Unificado da Rational (RUP, *Rational Unified Process*) é um *framework* que foi desenvolvido por uma empresa chamada *Rational Software Corporation*, que foi incorporada pela IBM gerando a IBM Rational. Este *framework*, que levou mais de uma década de elaboração, é composto por Papéis (Quem), Artefatos (O que), Atividades (Como) e Fluxo de Atividades (Quando).

Machado (2016, p. 60) explica que “o RUP define um catálogo de elementos de processos que deve ser instanciado. A instanciação é a escolha dos elementos que serão utilizados para compor o processo de desenvolvimento a ser utilizado em um projeto específico”.

Sobre o tamanho deste *framework*, Machado (2016, p. 71) descreve que “O RUP define quatro fases, nove disciplinas, mais de 40 papéis e mais de 100 artefatos; o produto comercial RUP possui mais de mil páginas de orientação”. Por essa razão ele é classificado por muitos como muito pesado (*heavyweigth*), no entanto na prática ele deve ser adaptado para ser utilizado. Machado (2016, p. 71) justifica que “Na própria definição do RUP (IBM, 2013), menciona-se que nenhum projeto pode, por si só, beneficiar-se da utilização do *framework* RUP completo”.

Machado (2016) detalha que o RUP pode ser adaptado em dois níveis, Nível de organização quando o RUP recebe um formato que será aproveitado em mais de um projeto da organização e Nível de projeto quando um formato é adaptado ou instanciado de acordo com as necessidades de um projeto específico. Essa adaptação ainda pode ser dividida em três tipos: extensão, configuração e instanciação. O último tipo é a seleção de um processo a partir dos elementos do *framework* do RUP. Configuração é modificação de processo para necessidades específicas e a extensão também, com a diferença que neste caso são considerados elementos que estão fora do universo RUP.

Algo que têm consonância com o assunto de escopo é uma parte do RUP chamada de práticas, que podem ser chamadas de boas práticas. Elas foram consolidadas através de um estudo, que é explicado por Machado (2016, p 61):

> […] foram identificados alguns sintomas comuns e recorrentes da falha de projetos, como entendimento impreciso das necessidades do usuário; falta de habilidade para lidar com mudanças nos requisitos; problemas de integração em módulos do sistema; dificuldade de manutenção e de evolução; descoberta tardia de sérias falhas do projeto; má qualidade do *software*; performance inaceitável do *software*; falta de rastreabilidade, no sentido de saber que membros da equipe alteraram que parte do sistema; processo de liberação de versões de *software* não confiável.

Em contrapartida, uma lista com boas práticas foi confrontada com os projetos estudados e “as práticas recomendadas pelo RUP foram identificadas em grande número de projetos de sucesso e sua falta foi verificada em um grande número de projetos fracassados” (MACHADO, 2016, p. 62). Essas práticas são:

- Desenvolver software iterativamente
- Gerenciar requisitos
- Usar arquitetura baseada em componentes
- Modelar *software* visualmente
- Verificar a qualidade do *software* continuamente
- Controlar as mudanças no *software*

Para o último ponto e o penúltimo há disponível hoje no mercado uma série de ferramentas (inclusive gratuitas) para ajudar nessas atividades, ferramentas Git[3] para controlar mudanças no software e ferramentas de testes automatizados[4] para tornar a verificação de qualidade do *software* mais rápida e mais barata.

Machado (2016, p. 64) expõe a sua visão dos benefícios da prática de modelar *software* visualmente, que está sintetizada a seguir:

> Modelos são usados para compreender melhor um problema ou uma solução, para comunicar uma ideia, para descrever uma abordagem ou alternativas de resolução de questões relacionadas ao sistema; para documentar essas alternativas ou a decisão tomada em prol de uma das alternativas.
> […] Assim, a modelagem melhora a habilidade da equipe de gerenciar a complexidade do *software*.
> Através da visualização dos modelos, comportamentos do sistema podem ser descritos de forma não ambígua, detalhes podem ser abstraídos quando necessários, inconsistências e problemas de arquitetura podem ser encontrados nas facilmente (em oposição à análise de código ou de descrições textuais).

No subcapítulo seguinte “Prototipação” a questão de modelagem do sistema durante o desenvolvimento continuará sendo abordada.

Para encerrar a sucinta descrição sobre o RUP, serão abordadas as suas chamadas fases do processo. São quatro fases Iniciação ou Concepção, Elaboração, Construção e Transição.

Em relação as etapas do gerenciamento do projeto do PMI, a Iniciação do RUP pode acontecer durante a etapa homônima do PMI (2013), como pode-se concluir nos comentários escritos por Machado (2016, p. 70):

> […] durante essa fase, é definida a visão geral do produto a ser desenvolvido; a principal preocupação é uma compreensão abrangente dos requisitos do sistema e a determinação do escopo do projeto. A iniciação termina quando o marco de Objetivo do Ciclo de Vida é alcançado: os

3 https://github.com
4 https://junit.org/junit5/

> *stakeholders* concordam com uma definição de escopo para o projeto, e com as estimativas de custo e prazo; os requisitos estão compreendidos e documentados; existe credibilidade quanto às estimativas de custo e prazo, e quanto a prioridades, riscos e o processo de desenvolvimento.

A fase seguinte é a Elaboração. Nota-se que pelo ponto de vista do PMI (2013) as atividades dessa fase podem ser consideradas na etapa de Planejamento, nela ainda são produzidas algumas definições importantes no projeto, como “planejamento das atividades e recursos necessários ao projeto, a especificação detalhada dos requisitos e o projeto da arquitetura do sistema” (MACHADO, 2016, p. 70). Pode existir desenvolvimento de código nesta fase, mas como explica Machado (2016, p. 70) “As atividades de geração de código desta fase são voltadas à criação de um protótipo da arquitetura, à mitigação de riscos técnicos através da experimentação de possíveis soluções e ao treinamento em ferramentas e técnicas específicas”. O que marca o fim desta fase é a concordância dos envolvidos com os planos de entrega do sistema.

A próxima fase, Construção, consiste no desenvolvimento total do sistema. O que marca seu final é a aceitação de uma versão estável pelos *stakeholders*. Feito isso caminha-se para a quarta e última fase, Transição.

Esta última fase tem este nome por que representa a passagem da posse do sistema no domínio do desenvolvimento (ou homologação) para o domínio de produção que envolve questões técnicas de empacotamento, instalação, e com essa passagem deve-se empoderar os usuários com documentações de administração do sistema, treinamentos, suporte e manutenção.

Esse foi um resumo de um *framework* muito extenso, o que já se pode notar é que muitos dos seus elementos podem trazer benefícios para o gerenciamento do escopo, afinal isso está no cerne do RUP.

Outras técnicas e ferramentas ainda são abordadas no trabalho e já sabemos que podemos misturar e combinar técnicas e ferramentas para montar o arsenal que será utilizado em um projeto de desenvolvimento de software.

3.4 Prototipação

Essa atividade geralmente acontece durante o detalhamento do requisito, nos diversos modelos apresentados neste trabalho existe a previsão deste detalhamento que é onde a prototipação pode acontecer. Neste subcapítulo vamos discutir quais os benefícios de utilizar

essa ferramenta na sua gestão de escopo e como geralmente esta atividade está relacionada com outras atividades do projeto.

Machado (2016, p. 261) apresenta protótipo como um aspecto opcional dentro do contexto da modelagem de caso de uso que ele sugere como a técnica a ser utilizada para detalhar as especificações de requisito:

> Protótipo de interface com o usuário: esta seção não aparece no modelo clássico de especificação de caso de uso. É opcional e relaciona-se à atividade de Modelar a Interface do sistema. O objetivo dessa seção é facilitar o entendimento do fluxo de eventos do caso de uso, fornecendo informações mais tangíveis do que o diálogo entre ator e sistema.

Como as atividades podem estar relacionadas, um comentário sobre modelagem de caso de uso faz-se necessário. Nestes modelos são descritos os eventos do sistema através de descrições que podem conter algumas representações gráficas dos atores (pessoas ou sistemas) mostrados suas conexões e o sentido das comunicações, Machado (2016, p. 260) faz um comentário sobre o fluxo de eventos, "Os eventos são descritos da forma mais independente possível da solução técnica, para evitar a tomada de alguma decisão que deveria ser feita durante as atividades de projeto (design) do sistema".

Sommervile (2011) apresenta um capítulo inteiro dedicado aos assuntos de Modelagem de sistemas no seu livro Engenharia de Software. Ele divide os modelos em 4 categorias, Modelos de contexto, Modelos de interação, Modelos estruturais e Modelos comportamentais. O modelo de caso de uso (onde pode ser adicionado um protótipo) é um dos tipos apresentados na categoria Modelos de integração.

Após abordado como e quando utilizar essa modelagem, apresenta-se os benefícios da prototipação, especialmente a prototipação de interface, utilizando as pontuações do autor Machado (2016, p. 262),

> A interface com o usuário é um dos recursos mais produtivos na obtenção de *feedback* do cliente sobre o sistema em desenvolvimento […] Além disso, sistemas de software precisam de interfaces planejadas e uniformes, para melhorar sua usabilidade, a facilidade de manutenção e até mesmo a produtividade, através do reúso de código padronizado.

Interface pode trazer dar a ideia que este protótipo é necessariamente visual, mas isso não é verdade, como podemos ver na explicação a seguir:

> O artefato gerado é um Protótipo de Interface, que pode ser um documento textual sobre o comportamento, layout e outras características comuns de interface; um protótipo gráfico; ou qualquer outro formato que oriente a equipe de desenvolvimento sobre a padronização de interface. (MACHADO, 2016, p. 262)

A utilidade de protótipo também pode ir além do uso interno pela equipe, e pode ser um instrumento de coleta de *feedback*, validação, como já apontado anteriormente e como expressa Sutherland (2014, p. 125),

> O importante não é ter um projeto totalmente estabelecido no início, mas construir um protótipo funcional e, depois, ver o que é possível aprimorar. E, então, depois de fazer tal aprimoramento, construir o protótipo seguinte e aprimorá-lo novamente. A ideia é que o quanto antes você tenha um *feedback* real, mais rápido você será capaz de construir um produto melhor.

O profissional responsável por produzir essas modelagens pode ser tanto um membro não especializado que detenha um mínimo de conhecimento, um membro especializado dentro da equipe, alguém externo à equipe ou até alguém externo à própria empresa da equipe de projeto, "O responsável por essa atividade é o analista de sistemas, que pode utilizar recursos como ferramentas de construção de protótipos, padrões de interface de mercado ou mesmo a ajuda de um *expert* em projeto gráfico" (MACHADO, 2016, p.262). Existem definições de cargos hoje em dia que se encaixam muito bem nesta atividade, como o caso dos UI *Designer* e UX *Designer*[5].

O protótipo é um artefato que já consta em diversos modelos de processo de desenvolvimento. Caso não seja integrante dentre os recursos escolhidos para tocar o desenvolvimento é uma prática que vale a pena ser considerada, pela sua simplicidade conceitual e pelo alto valor que traz ao entendimento do produto. As formas de utilizar são muitas. Por exemplo: Integrar um profissional especializado, usar como documento de validação do projeto pelo cliente. Casa forma de utilizar esse artefato vai ter seu impacto no custo e no tempo do projeto, por isso deve ser pensado quando e como utilizar.

3.5 Engenharia dirigida a modelos

Modelos não são meros artefatos para documentação ou validação, existe uma abordagem de desenvolvimento que considera os modelos como a peça-chave da

5 https://designculture.com.br/o-que-e-ui-design-e-ux-design

implementação, fazendo a implementação automática e parametrizada de sistemas através de ferramentas que permitem gerar sistemas através do desenho de modelos, "A ideia fundamental por trás da MDE[6] é que a transformação completamente automatizada de modelos para códigos deve ser possível. Para conseguir isso, é preciso ser capaz de construir modelos gráficos com semântica bem-definida" (SOMMERVILLE, 2011, p. 99).

Sommerville (2011, p. 96) destaca alguns argumentos a favor da MDE:

> A engenharia dirigida a modelos permite que os engenheiros pensem em sistemas em alto nível de abstração, sem preocupações com detalhes de implementação. Isso reduz a probabilidade de erros, acelera o processo de projeto e implementação e permite a criação de modelos de aplicação reusáveis, independentes de plataforma. Por meio de ferramentas poderosas, as implementações do sistema podem ser geradas para diferentes plataformas a partir do mesmo modelo, Portanto para adaptar o sistema a uma nova tecnologia de plataforma, é necessário apenas escrever um tradutor para essa plataforma. Quando este estiver disponível, todos os modelos independentes de plataforma podem ser rapidamente reimplementados na nova plataforma.

Há casos de sucesso da sua aplicação, "As técnicas têm sido utilizada com sucesso no desenvolvimento de grandes sistemas com ciclos duradouros, como sistema de tráfego aéreo" (SOMMERVILLE, 2011, p. 98) e pela internet podemos encontrar mais casos de sucesso nas páginas da OMG[7] (*Object Management Group*), onde podemos entender melhor em quais casos essa abordagem melhor se aplica, pois também devemos tomar cuidado com o seu revés, como comenta Sommerville (2011, p. 98): "acredito que a MDE seja uma evolução importante. No entanto, como é o caso com os métodos formais, não está claro se os custos e riscos das abordagens dirigidas a modelos ultrapassam os possíveis benefícios".

Um custo que deve ser observado é o custo de seu *set-up*, tanto em termos de ferramentas de softwares avançadas necessárias para esse desenvolvimento, quanto no desenvolvimento de um projeto "A noção de modelagem inicial extensiva contradiz as ideias fundamentais do manifesto ágil" (SOMMERVILLE, 2011, p. 98) e também não há indicativos que ela funcione bem com certos modelos de testes, "pelo que sei, não existem ferramentas de MDA[8] que deem apoio às práticas como testes de regressão e desenvolvimento dirigido a testes" (SOMMERVILLE, 2011, p. 98).

---

6 Do inglês *model-based engineering*
7 https://www.omg.org/mda/products_success.htm
8 Do inglês *model-driven architecture*

3.6 Desenvolvimento dirigido a testes

No livro Engenharia de software encontra-se a seguinte definição: “Essencialmente, você desenvolve um código de forma incremental, em conjunto com um teste para esse incremento. Você não caminha para o próximo incremento até que o código desenvolvido passe no teste”. O autor ainda comenta a origem desta metodologia: “O desenvolvimento dirigido a testes foi apresentado como parte dos métodos ágeis, como o Extreme Programming. No entanto, ele também pode ser usado em processos de desenvolvimento dirigido a planos” (SOMMERVILLE, 2011, p. 155).

No diagrama a seguir podemos ver uma esquematização feia por Sommerville:

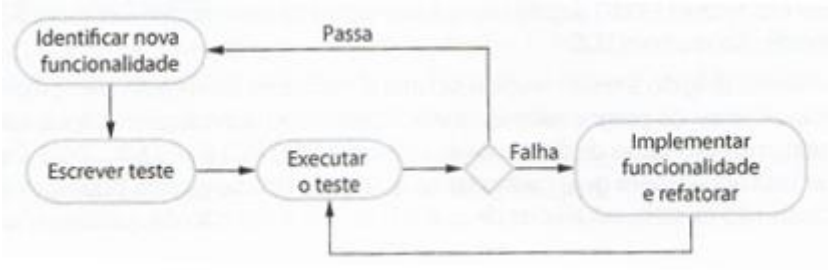

Figura 3 – Desenvolvimento dirigido a testes
Fonte: Sommerville (2011).

Como toda metodologia, traz benefícios e demanda alguns comportamentos. Um comportamento importante que o projeto precisa adquirir é a prática de testes automatizados.

> Como o código é desenvolvido em incrementos muito pequenos, você precisa ser capaz de executar todos os testes cada vez que adicionar funcionalidades ou refatorar o programa. Portanto os testes são embutidos em um programa separado que os executa e invoca o sistema testado. Usando essa abordagem, é possível rodar centenas de testes separados em poucos segundos (SOMMERVILLE, 2011, p. 156).

Essa metodologia, como demonstrado em sua definição e pela sua principal necessidade de realizar testes automatizados, é simples de ser praticada e alguns de seus benefícios, apontados por Sommerville (2011, p. 156), são:

- Cobertura de código: Como os testes (de regressão) são realizados a cada pequeno pedaço de código, temos uma garantia maior de que todo o programa foi testado.

- Testes de regressão: Os testes de regressão, por estarem automatizados reduzem os custos para esse tipo de teste, já que não precisam ser realizados manualmente.

- Depuração simplificada: Quando alguma falha acontece, a localização do problema deve ser óbvia já que esse rastro do sistema (*log* detalhado do teste) faz parte da automatização do teste.

- Documentação de sistema: Os testes funcionam como um tipo de documentação descrevendo o sistema deve fazer, tornando fácil o entendimento do comportamento do sistema.

Além desses citados, um ganho direto ao time, comentado por Summerville (2011, p. 156) é o incentivar um maior entendimento do desenvolvedor sobre o sistema, “Para escrever um teste, você precisa entender a que ele se destina, e como esse entendimento faz que seja mais fácil escrever o código necessário”.

Escolher uma boa ferramenta de automação de testes é fundamental. Uma observação óbvia e importante é que a ferramente de automatização de testes deve ser compatível com a linguagem de programação utilizada no projeto. Vale lembrar que essa metodologia não se basta sozinha, mesmo em relação aos testes, outros testes além do que ela prevê ainda são necessários, “ainda precisa de um processo de testes de sistema, isto é, para verificar se ele atende aos requisitos de todos os *stakeholders*” (SOMMERVILLE, 2011, p. 156) por exemplo.

Pela simplicidade desta metodologia, ela pode ser usada em conjunto com outras metodologias, aproveitando das vantagens desta. Uma última recomendação sobre essa metodologia: “O desenvolvimento dirigido a testes é de maior utilidade no desenvolvimento de softwares novos, em que a funcionalidade seja implementada no novo código ou usando bibliotecas-padrão já testadas” (SOMMERVILLE, 2011, p. 156).

O próximo subcapítulo vai abordar assuntos que estão mais na camada de apresentação do sistema que em seu interior, dirigido à interface e a experiência do usuário.

3.7 Manifesto ágil

Antes de comentar sobre SCRUM e XP, faz-se necessário escrever uma pequena introdução sobre metodologia ágil. O foco deste trabalho é o escopo e tratar deste assunto não é fugir do tempo. Como já explicado, há escopo de produto e de projeto e os princípios que tratam o manifesto ágil têm impacto sobre o escopo do projeto. Soomerville (2011, p. 40) explica o panorama histórico desse manifesto:

> A insatisfação com essas abordagens pesadas da engenharia de software levou um grande número de desenvolvedores de software a proporem, na década de 1990, novos 'métodos ágeis'. Estes permitiram que a equipe de desenvolvimento focasse no software em si, e não em sua concepção e documentação. […] A filosofia por trás dos métodos ágeis é refletida no manifesto ágil, que foi acordado por muitos dos principais desenvolvedores desses métodos.

O manifesto em si é uma lista de princípios, frases, pensamentos que guiaram a construção dos chamados métodos ágeis, e esses métodos têm suas particularidades na gestão do escopo, que serão tratadas nas apresentações que vamos fazer de alguns desses métodos. "Embora esses métodos ágeis sejam todos baseados na noção de desenvolvimento e entrega incremental, eles propõem diferentes processos para alcançar tal objetivo" (SOMMERVILLE, 2011, p. 40).

3.8 Extreme Programming (XP)

"Exterme Programming (XP) é talvez o mais conhecido e mais utilizado dos métodos ágeis" é como o introduz Sommerville em seu livro (SOMMERVILLE, 2011, p. 44). Ele ainda explica o seu funcionamento:

> os requisitos são expressos como cenários (chamados estórias do usuário), que são implementados diretamente como uma série de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes de escreverem o código. Quando um novo código é integrado ao sistema, todos os testes devem ser executados com sucesso. Há um curto intervalo de entre os *release* do sistema

A figura a seguir ilustra esse funcionamento:

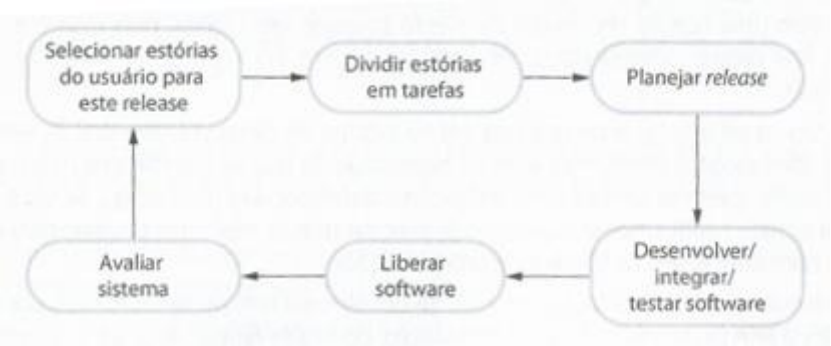

Figura 4 – O ciclo de um *release* em Extreme Programming
Fonte: Sommerville (2011).

Pela descrição assemelha-se com desenvolvimento dirigido a testes, porém destaco uma diferença, neste temos lançamentos do software para o cliente durante o desenvolvimento. Além disso XP traz consigo um conjunto de valores e princípios fortes que determinam comportamentos do projeto, cito aqui um valor relacionado como a gestão de escopo, explicado pelo autor Machado (2016, p. 77), "Assumir simplicidade: deve-se procurar pela solução mais simples possível para cada problema. Projetos simples são mais fáceis de implementar e podem ser alterados em caso de necessidade"

O próximo assunto que abordaremos, SCRUM, vai trazer semelhanças como XP, principalmente no funcionamento, porém têm suas diferenças, como a maneira como são tratados os requisitos (o *backlog* do SCRUM *versus* o jogo de planejamento do XP) e pelos novos papéis que serão descritos a seguir.

3.9 Contribuições do SCRUM

SCRUM não é uma metodologia, "SCRUM é um *framework* (diga-se um conjunto de ferramentas e técnicas) usado para o desenvolvimento ágil de software de forma iterativa e incremental" (MACHADO, 2016, p. 203).

Por ser incremental e cíclico ele se parece com o XP, além de disso ambos compartilham o conceito de estórias do usuário. Porém algumas ferramentas e papéis são próprios do SCRUM, serão descritos nesse subcapítulo destacando especialmente sua relação com o escopo.

Conceitos do SCRUM começaram ser utilizados antes que todas as suas características fossem modeladas. Como explica um dos formatadores, Jeff Sutherland, no seu livro: Scrum: A arte de fazer o dobro do trabalho na metade do tempo. Nesse livro Sutherland (2014, p. 54)

conta como foi o *background* da concepção do Scrum e como a prática foi ganhando forma, através de muito criação coletiva, “nosso objetivo não era apenas ser uma boa equipe, mas ser a melhor. Como conseguiríamos isso? Uma vez mais, a resposta foi algo simples que ‘roubamos’ de outra pessoa – uma reunião diária”. Machado (2016, p. 208) explica sucintamente o que é a reunião diária:

> No SCRUM há avaliações diárias do andamento do projeto. A cada dia de um Sprint, a equipe faz uma breve reunião, chamada Daily SCRUM Meeting.
> Essa reunião é utilizada para que a equipe relate o que foi feito desde a última reunião, o que será feito até a próxima reunião e para informar a causa para determinados atrasos.

Dois conceitos novos apareceram nessa explicação sobre reunião diária, Sprint e ScrumMaster. O primeiro é uma técnica para organizar o tempo, “Metodologias ágeis de desenvolvimento de software são interativas, ou seja, o trabalho é dividido em iterações, que são chamados de Sprints no caso do SCRUM (MACHADO, 2016, p. 208)”, o segundo é um papel, “O ScrumMaster é a pessoa responsável por garantir que o processo seja entendido e seguido. Ele deve verificar se a equipe dou time de SCRUM está aderindo aos valores, princípios, práticas e regras do SCRUM” (MACHADO, 2016, p. 207). Explicando o papel com mais detalhes:

> Apesar de o ScrumMaster não gerenciar a equipe ou time de SCRUM (visto não ser um gerente de projeto), pois este time deve ser auto-organizado, ele deve fazer com o time de SCRUM entenda e use autogerenciamento e interdisciplinariedade. No nosso mundinho real o ScrumMaster acaba sendo um papel exercido pelo gerente de projetos.
> A responsabilidade do ScrumMaster é manter o foco no processo, remover impedimentos da equipe e auxiliar na comunicação entre equipe e o Product Owner (MACHADO, 2016, p. 207).

Com o Product Owner mais o Time (ou Equipe), estes são os três papéis presentes no SCRUM. O Time são os desenvolvedores, as pessoas que vão escrever, testar, arquitetar o sistema: É a equipe de desenvolvimento. Product Owner, também chamado de Dono do Produto, é um papel próprio do SCRUM, “Ele possui a visão do retorno que o projeto trará para a Empresa e para os envolvidos” (MACHADO, 2016, p. 207). Ele está relacionado com uma ferramenta chamada no SCRUM de Product Backlog, “sua missão é cuidar do Product Backlog (lista de requisitos), planejar releases, priorizar requisitos e passar ao time uma visão

clara dos objetivos do projeto" (MACHADO, 2016, p. 207). O autor ainda faz um aconselhamento sobre a pessoa que deve assumir esse papel "É indicado então que o PO seja um profissional do lado do cliente, lembra vagamente o Patrocinador (Sponsor) de um projeto, porém sua interatividade com o projeto e participação constante fazem deste papel um diferencial muito importante (MACHADO, 2016, p. 207)."

Cohn (2011, p. 148) apresenta uma explicação do que é o Product Backlog:

> As descrições de alto nível dos requisitos podem ser coletadas no início […] Elas são documentadas em um backlog de produto […] Diferentemente de um documento de requisitos tradicional, o backlog do produto é altamente dinâmico, itens são adicionados, removidos e repriorizados a cada sprint à medida que conhecemos melhor do produto, os usuários, a equipe e assim por diante.

Machado (2016, p. 212) explica e importância das user stories, "Uma comunicação eficaz é a chave, e precisamos estabelecer uma linguagem comum. A user story fornece a linguagem comum para construir um entendimento entre o interessado e a equipe técnica". No SCRUM tradicionalmente cada uma dessas estórias é escrita num cartão (do tamanho de um cartão de fichário), isso para que se limite e se escreva de uma forma resumida no intuito de gerar uma visão geral do requisito. "Como as user stories devem ser escritas pelo cliente, como certeza encontraremos problemas para compreender o cliente" (MACHADO, 2016, p. 210), por esta razão, muitas vezes no SCRUM alguém da esquipe fica com esta incumbência, "Na prática, qualquer membro da equipe com conhecimento do domínio (Negócio) é suficientemente capaz de escrever user stories, mas cabe ao Product Owner aceitar e dar prioridade a essas histórias em potencial para o product backlog" (MACHADO, 2016, p. 211).

Cohn (2011, p. 148) fala sobre o papel o Product Owner em relação ao product backlog, "Para mim, não importa quem executa o ato físico de redigir o backlog do produto [..] Além de assegurar que exista o backlog do produto, o dono do produto adiciona detalhes à visão respondendo perguntas feitas pelos membros da equipe". Cohn (2011, p. 148) ainda explica essa responsabilidade através de um exemplo:

> Um dono de produto pode dizer, 'Converse com Nirav se tiver perguntas sobre como devem funcionar os requisitos de carrinho de compras e pagamento no caixa', mas se Nirav não responder ou não for útil, um bom dono de produto entrará no circuito e responderá as perguntas pessoalmente,

> descobrirá por que Nirav não pôde fazer isso, designará uma pessoa diferente ou encontrará alguma outra solução.

Outras técnicas podem ser mescladas para tirar ainda mais proveito dessas ferramentas, “Utilizando práticas de engenharia de requisitos, mantemos as user stories bastante simples e alinhadas ao negócio, melhorando a especificação do que deve ser feito e consequentemente, aumentando a velocidade e a confiança da equipe” (MACHADO, 2016, p. 211).

Apesar de muito se ver esses assuntos relacionados, não são dependentes “o SCRUM não prega como obrigação seguir a user story. Ela é bastante utilizada devido à sua grande facilidade neurolinguística de capturar necessidades e comprovar a sua real utilidade” (MACHADO, 2016, p. 211), afinal de contas “No desenvolvimento ágil, é o trabalho do desenvolvedor falar a língua do interessado, e não do interessado falar a língua do desenvolvedor” (MACHADO, 2016, p. 212).

Pode parecer complicado, mas se analisado friamente o SCRUM é composto por 3 papéis (Product Owner, ScrumMaster e Time), 2 conceitos (Product Backlog e Sprint) e alguns ritos (*daily*, reunião de planejamento de Sprint, reunião de revisão da Sprint e reunião de retrospectiva da Sprint). Ele pode ser facilmente aplicado e gera muita agilidade e produtividade em projetos de sistemas voltados para negócios.

Para ilustrar a aplicação do SCRUM no mundo real, quando estagiei na CI&T, em um projeto de sustentação da seguradora Berkley, em 2016, o papel do Product Owner era realizado pelo Gerente de Projetos e estávamos começando a usar pontos de função para medir os incrementos. Esse exemplo é para ilustrar duas coisas: SCRUM pode ser utilizado em conjunto com outras ferramentas e técnicas, não existe uma forma definida como você vai utilizar os papéis do SCRUM na sua organização.

Porém, deve-se tomar cuidado com uma questão, nem sempre as técnicas e ferramentas vão se misturar bem. O conceito do desenvolvimento ágil é produzir apenas aquilo que tem valor, portanto a documentação e o planejamento devem ser repensados, “Em métodos ágeis não há prescrição de documentação […] Em projetos ágeis você pode documentar sem problemas desde que a documentação seja necessária de fato. A ideia é que não devemos perder tempo com nada que não seja requerido de verdade para o projeto” (MACHADO, 2016, p. 219).

Atenção na última frase da citação anterior. Não é fácil definir o que é requerido de verdade para o projeto, se pensar pelo lado da formalização do que está e o que não está no

acordo com o cliente fica óbvio, mas e a definição do que está e o que não está no acordo? Quais os riscos de formalizar um projeto sem prever testes, etapa de levantamento de requisitos, validações de prototipações, implantação em produção? O comentário ao final deste subcapítulo é que um projeto de software vai além do puro desenvolvimento, o conjunto de técnicas e ferramentas escolhidos para um projeto deve ser suficientemente abrangente para cobrir todos os pontos críticos e racional o suficiente para não tornar o projeto caro e lento demais.

Para exemplificar essas escolhas está reproduzida abaixo uma história contada pelo Sutherland (2014, p. 27):

> Na Easel eu sabia que a metodologia em cascata atrasaria nosso prazo em messes, talvez até anos. Precisávamos descobrir uma maneira completamente diferente de fazer as coisas. Eu fui até o CEO e informei que íamos rasgar o diagrama de Gantt. Ele ficou chocado e exigiu saber o motivo.
> "Quantos diagramas de Gantt você já viu na sua carreira", perguntei.
> "Centenas", ele respondeu
> "Quantos estavam certos?"
> Ele fez uma pausa.
> "Nenhum"
> Então eu disse a ele que entregaria um software em perfeito funcionamento no final do mês, em vez de um diagrama de Gantt não cumprido. Ele mesmo poderia testá-lo para verificar se estávamos no caminho certo.

3.10 Desenvolvimento dirigido a planos

Este modelo não se constitui de muitas técnicas e ferramentas, é basicamente a maneira de encarar o projeto, "é uma abordagem da engenharia de software em que o processo de desenvolvimento é planejado em detalhes. Um plano de projeto é criado para registrar o trabalho a ser feito, quem o fará, o cronograma de desenvolvimento e os produtos do trabalho" (SOMMERVILLE, 2011, p. 434).

Este simples modo de fazer é citado no trabalho por conta da consonância que há com as práticas do PMI (2013) e também pelo comentário feito pelo Sommerville (2011, p. 434): "Em minha opinião, a melhor abordagem para o planejamento de projeto envolve uma mistura equilibrada entre o desenvolvimento baseado em planos e o ágil. O equilíbrio depende do tipo de projeto e das habilidades das pessoas que estão disponíveis". O desenvolvimento dirigido a planos é algo que deve ser considerado, principalmente em alguns tipos de software, "Em um extremo, sistemas críticos de proteção e segurança de grande porte

requerem extensa análise inicial e podem precisar ser certificados antes de serem colocados em uso. Esses, principalmente devem ser dirigido a planos" (SOMMERVILLE, 2011, p. 434) e também no caso de alguns projetos, "Quando várias empresas estão envolvidas em um projeto de desenvolvimento, uma abordagem dirigida a planos é normalmente usada para coordenar o trabalho de cada área de desenvolvimentos" (SOMMERVILLE, 2011, p. 434), em outros casos, essa abordagem não é tão importante, "No outro extremo, pequenos ou médios sistemas de informações, para serem usados em um ambiente competitivo que muda rapidamente, devem ser ágeis" (SOMMERVILLE, 2011, p. 434).

3.11 Gold platting

Este subcapítulo se diferencia dos demais pois não traz técnicas e ferramentas, porém traz uma recomendação de algo que não se deve fazer. Nas palavras do autor Xavier (2006, p. 136):

> A expressão *gold platting* é utilizada na literatura para o escopo não solicitado (adicional) que a equipe do projeto fornece ao cliente. Isso não significa que está oferecendo um produto melhor, mas sim fora (ou além) das especificações do cliente, sob riscos de aumento de custos, prazos etc. Pela ótica do PMI, o gerente do projeto deve estar atento para não permitir o trabalho da equipe em escopo não planejado.

Uma coisa é clara, *gold platting* é algo que está fora do escopo. Se está fora do escopo, está fora do gerenciamento. Se é algo que deveria estar no escopo, o planejamento deve ser revisto, o impacto deve ser medido, e as partes interessadas devem ser comunicadas. Isto é o certo a ser feito, para que o projeto não acumule prejuízo e para que o cliente tenha visibilidade do que está sendo feito.

A ocorrência do *gold platting* é algo que as técnicas e ferramentas abordadas nesse trabalho ajudam evitar visto que elas dão formas de descobrir, documentar, monitorar e controlar o escopo.

4 CONSIDERAÇÕES FINAIS

O trabalho apresentou um conjunto diversificado de constructos conceituais (modelos, técnicas e ferramentas) tem têm trazem benefícios no gerenciamento de escopo em projetos de desenvolvimento de software.

A ideia do panorama montado com os tópicos escolhidos não foi cobrir tudo que existe entre técnicas e ferramentas e sim mostrar ao leitor algumas abordagens radicalmente diferentes entre si, dês dos modelos de processo (cascata e incremental) abordagens (chamadas formais e as ágeis) e complexidades (leve ou simples e pesados), dentre as mais notórias e significativas.

Isso não tinha a intenção de confundir o leitor e o deixar perdido e sim mostrar que não existe um formato único de equacionar o gerenciamento do escopo em projetos de software. Cada software, cada projeto, cada tipo de equipe vai se favorecer mais por uma determinada abordagem. Essa é a ideia que o trabalho quer deixar na mente do leitor: Existe uma certa abordagem, ou um conjunto de abordagens, que vai trazer maiores benefícios dado uma determinada configuração.

Sendo particularidades de cada um, o debate sobre como e quando aplicar os modelos, técnicas e ferramentas foi discutido diretamente nos seus respectivos tópicos.

Uma prática é certa e foi recorrente em alguns dos tópicos: Os modelos, técnicas e ferramentas podem ser misturados e combinados para maximizar seus efeitos, tando Sommerville (2011) como Sutherland (2014) fizeram insinuações nesse sentido.

A prática de misturar diferentes duas ou mais metodologias (ou *frameworks*) é também é popularmente conhecida como metodologia híbrida[9], um nome único para dizer que está sendo usada duas ou mais metodologias (ou *frameworks*).

O que se pode concluir é que um gerente de projetos que vai trabalhar com desenvolvimento de software não deve se restringir à apenas uma determinada escola de pensamento. O software, o projeto e as pessoas que vão ditar quais modelos, técnicas e ferramentas devem ser utilizados. Não utiliza-las, como demonstrado pelas pesquisas das organizações como a *Rational Software Corporation* ou de experiência de autores como Sutherland (2014) e Cohn (2011), é o que determina o fracasso nos projetos.

O trabalho espera ter gerado no leitor que atua como gerente de projetos em projetos de software que as técnicas e ferramentas do desenvolvimento, que apoiam no gerenciamento

9 https://www.devmedia.com.br/modelo-hibrido-no-gerenciamento-de-projetos/37289

de escopo, são necessárias e que ele não deve escolher qualquer uma e sim ter uma ideia abrangente das opções e saber escolher as melhores opções de acordo com as situações.

5 BIBLIOGRAFIA

COHN, Mike. **Desenvolvimento de Software com Scrum:** Aplicando médotos ágeis com sucesso. São Paulo: Bookman, 2011.

MACHADO, Felipe Nery Rodrigues. **Análise e Gestão de Requisitos de Software:** Onde Nascem os Sistemas. 3. ed. São Paulo: Saraiva Educação Ltda., 2016.

PROJECT MANAGEMENT INSTITUTE (PMI) (Org.). **Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®).** 5. ed. Newtown Square: Globalstandard, 2013.

SOMMERVILLER, Ian. **Engenharia de software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SUTHERLAND, Jeff. **Scrum:** a arte de fazer o dobro do trabalho na metade do tempo. São Paulo: Leya, 2014.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **Análise de Pontos de Função:** Medição, Estimativa e Gerenciamento de Projetos de Software. 13. ed. São Paulo: Editora Saraiva, 2015.

VAZQUEZ, Carlos Eduardo. **Engenharia de Requisitos:** software orientado ao negócio. São Paulo: Brasport, 2016.

XAVIER, Carlos Magno da Silva. **Gerenciamento de Projetos:** Como definir e controlar o escopo do projeto. São Paulo: Editora Saraiva, 2006.