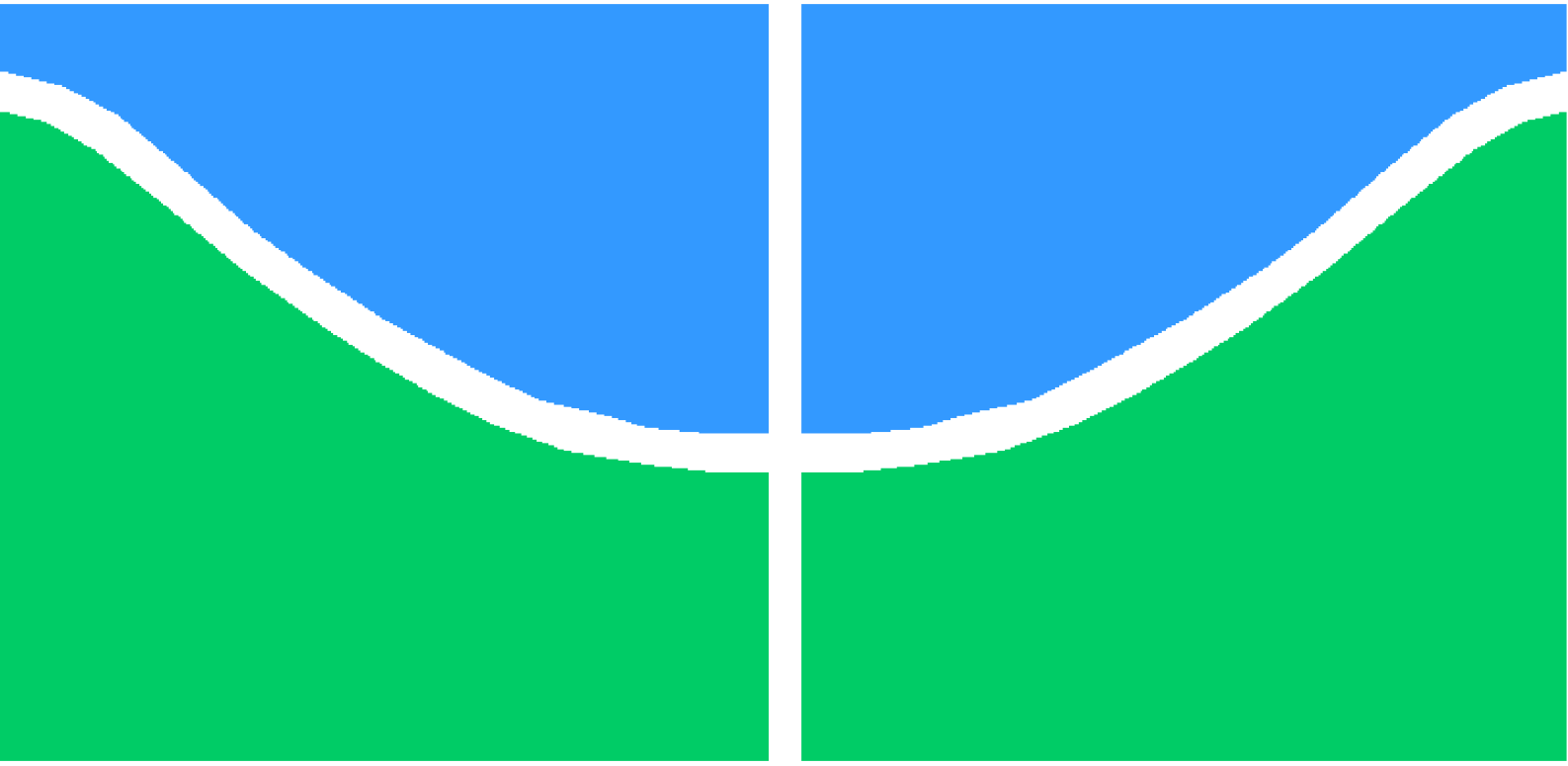

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Aplicação Web para Contagem de Tamanho Funcional de Software

Autor: Daniel Henrique Marinho Damasceno
Orientador: (Prof.Msc. Elaine Venson)

Brasília, DF
2016

Daniel Henrique Marinho Damasceno

Aplicação Web para Contagem de Tamanho Funcional de Software

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: (Prof.Msc. Elaine Venson)

Brasília, DF

2016

Daniel Henrique Marinho Damasceno
Aplicação Web para Contagem de Tamanho Funcional de Software/ Daniel Henrique Marinho Damasceno. – Brasília, DF, 2016-
63 p. : il. (algumas color.) ; 30 cm.

Orientador: (Prof.Msc. Elaine Venson)

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. medição de tamanho funcional. 2. automação. I. (Prof.Msc. Elaine Venson). II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aplicação Web para Contagem de Tamanho Funcional de Software

CDU 02:141:005.6

Daniel Henrique Marinho Damasceno

Aplicação Web para Contagem de Tamanho Funcional de Software

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 09 de Dezembro de 2016:

(Prof.Msc. Elaine Venson)
Orientador

Prof.Dr Paulo Roberto Miranda Meirelles
Convidado 1

Prof.Msc. Ricardo Ajax Dias Kosloski
Convidado 2

Brasília, DF
2016

A dedicatória é opcional. Caso não deseje uma, deixar todo este arquivo em branco.

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

A inclusão desta seção de agradecimentos é opcional, portanto, sua inclusão fica a critério do(s) autor(es), que caso deseje(em) fazê-lo deverá(ão) utilizar este espaço, seguindo a formatação de *espaço simples e fonte padrão do texto (arial ou times, tamanho 12 sem negritos, aspas ou itálico.*

Caso não deseje utilizar os agradecimentos, deixar toda este arquivo em branco.

A epígrafe é opcional. Caso não deseje uma, deixe todo este arquivo em branco.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

Estimar o tamanho funcional de software é fundamental para construção de modelos de estimativa e processos de desenvolvimento de software. Dentre suas várias utilidades, estimar o tamanho funcional de software auxilia na estimativa de esforço de desenvolvimento, na definição de prazos e no pagamento de desenvolvedores de software. A análise de pontos de função tem a proposta de medir o esforço necessário para a implementação de funcionalidades de acordo com a visão do usuário. Os órgão públicos brasileiros utilizam a contagem de pontos de função para realizar os pagamentos. Esse processo é feito por meio do preenchimento de planilhas sem um padrão definido. Em alguns casos a contagem de pontos de função de software pode ser exaustiva. Nesse contexto algumas ferramentas possibilitam a automatização e uma maior agilidade na execução desta tarefa. Quando falamos de ferramentas podemos notar um crescimento das iniciativas de desenvolvimento de softwares livres, aqueles aos quais temos acesso ao código e temos o direito de alterar, copiar e distribuir seu conteúdo. Neste contexto, o presente trabalho busca desenvolver, em parceria com a Secretaria de Tecnologia da Informação(STI), uma ferramenta para controle e gestão de tamanho funcional de software, adequando seus requisitos as necessidades dos órgãos públicos brasileiros e buscando uma forma de unificar e centralizar a contagem dos projetos desenvolvidos pelos órgãos.

Palavras-chaves: medição de tamanho funional. extreme programming. automação.

Abstract

Estimating the functional size of software is fundamental for building estimation models and software development processes. Among its various utilities, estimating the functional size of software assists in the estimation of development effort, in the definition of deadlines and in the payment of software developers. Function point analysis has the purpose of measuring the effort required to implement functionalities according to the user's vision. Brazilian public agencies use the function point count to make payments. This process is done by filling in worksheets without a defined pattern. In some cases the counting of software function points can be exhaustive. In this context some tools allow automation and greater agility in the execution of this task. When we talk about tools, we can see a growth in the initiatives of development of free software, those to which we have access to the code and we have the right to change, copy and distribute its content. In this context, the present work seeks to develop, in partnership with the Information Technology Secretariat (STI), a tool for control and management of software functional size, adapting its requirements to the needs of Brazilian public agencies and seeking a way to unify and Centralize the count of the projects developed by the organs.

Key-words: functional size measurement. extrem programming. automation.

Lista de ilustrações

Figura 1 – Representação Visual do XP (BECK, 2004) 32
Figura 2 – Práticas do XP (BECK, 2004) 33
Figura 3 – Processo de Desevolvimento XP (EXPERTIZA, 2016) 35
Figura 4 – Processo Adaptado XP (BERNABÉ; NAVIA, 2015) 37
Figura 5 – Processo de contagem de Pontos de Função (SISP, 2016) 42
Figura 6 – Macro-atividades do Processo de Desenvolvimento 47
Figura 7 – Atividade de Levantamento de Requisitos 48
Figura 8 – Atividade de Planejamento de Sprints 48
Figura 9 – Atividade de Execução da Sprint 49
Figura 10 – Atividade de Revisão da Sprint 50
Figura 11 – Ferramentas Escolhidas para o Desevolvimento 55

Lista de tabelas

Tabela 1 – Avaliação de Ferramentas 26
Tabela 2 – Análise de funcionalidades das ferramentas 27
Tabela 3 – Análise de Práticas no PXP (AGARWAL; UMPHRESS, 2008) 39

Lista de abreviaturas e siglas

XP	Extreme Programming
PXP	Personal Extreme Programming
IFPUG	International Function Point Users Group
STI	Secretaria de Tecnologia da Informação
API	Interface de Programação de Aplicação
JSON	JavaScript Object Notation
AJAX	Asynchronous Javascript and XML

Sumário

I INTRODUÇÃO 23

1 INTRODUÇÃO 25
1.1 Contexto 25
1.2 Descrição do Problema 26
1.3 Objetivos 27
1.4 Organização do Trabalho 28

II REFERENCIAL TEÓRICO 29

2 EXTREME PROGRAMMING 31
2.1 Metodologias ágeis 31
2.2 O que é o XP 31
2.3 Práticas 32
2.4 Visão de Projeto no XP 35
2.5 Um desenvolvedor no papel de time 36
2.6 Práticas para apenas um desenvolvedor 38

3 TAMANHO FUNCIONAL DE SOFTWARE 41
3.1 Tamanho Funcional de Software 41
3.2 Contagem de Pontos de Função 41
3.3 Problemas com o Ponto de Função 42
3.4 Utilização de Pontos de Função 43

III METODOLOGIA 45

4 METODOLOGIA 47
4.1 Definição do Processo 47
4.2 Levantar Requisitos 47
4.3 Planejar Sprints 48
4.4 Executar Sprint 49
4.5 Revisar Sprint 50

IV PROPOSTA 51

5 PROPOSTA 53

5.1 **Requisitos** **53**
5.1.1 Requisitos Extraídos a Partir do Estudo das Ferramentas 53
5.1.2 Requisitos Extraídos a Partir de Reunião com a STI 54
5.2 **Definição de Ferramentas** **54**
5.3 **Repositório** **56**

REFERÊNCIAS **57**

APÊNDICES **59**

APÊNDICE A – PRIMEIRO APÊNDICE **61**

APÊNDICE B – SEGUNDO APÊNDICE **63**

Parte I

Introdução

1 Introdução

1.1 Contexto

Estimar o tamanho funcional de um software é a chave para a construção de modelos e processos de desenvolvimento de software. Diferente das estimativas diretas, a estimativa de tamanho de software resulta em uma medida do próprio produto de software e pode ser usada para construir modelos de estimativa mais objetivos, para predizer o esforço e a duração de um projeto, estimar defeitos e medir a qualidade do produto (EBERT; SOUBRA, 2014).

A Análise de Pontos de Função, proposta para medir as funcionalidades de um software, também pode ser utilizada para medir o esforço necessário para o desenvolvimento de software. Pontos de Função são uma medida de tamanho funcional que utiliza termos lógicos para o maior entendimento de clientes e usuários do produto. (ALBRECHT, 1994)

Como a medição é feita sobre as funcionalidades, seu tamanho permanece o mesmo independente de tecnologias ou linguagens de programação. A medição pode ser feita no início do projeto, fazendo seu uso oportuno para o planejamento do design e do desenvolvimento do projeto. (KUSUMOTO et al., 2002)

A aplicação manual da medição de tamanho funcional é exaustiva e pode consumir muito tempo dar organizações que possuem um grande número de projetos. Em geral as organizações têm de lidar com vários projetos em um curto espaço de tempo, o que dificulta a estimativa de esforço e acaba gerando a perda de um dos benefícios gerados por tais estimativas. (EBERT; SOUBRA, 2014)

Em casos onde os um grande número de requisitos é analisado, e muitos desses são considerados complexos, uma análise especializada é necessária. Recentemente tecnologias e ferramentas têm emergido para automatizar e facilitar a medição de tamanho funcional. (EBERT; SOUBRA, 2014)

Neste contexto de ferramentas podemos destacar o de software livre, este que tem gerado um engajamento de comunidades e uma maior participação popular no últimos anos. (JOHNSON-EILOLA, 2002) "Software livre"refere-se à liberdade que os usuários têm de executar, copiar, distribuir, estudar e modificar o software de acordo com suas necessidades. Isso significa que o usuário tem acesso ao código fonte do software, além de cópias de suas versões e variações da mesma. (STALLMAN, 2003) *"Assim sendo, Software livre é uma questão de liberdade, não de preço. Para entender o conceito, pense em "liberdade de expressão", não em "cerveja grátis". "* (STALLMAN, 2003)

O desenvolvimento de software, neste caso de uma ferramenta livre, segue metodologias e processos estruturados para a organização de um time de desenvolvedores. O *Extreme Programming(XP)* oferece uma série de práticas e princípios para guiar o desenvolvedor durante os ciclos de desenvolvimento.

Metodologias ágeis, como o XP, buscam o aumento da organização das empresas enquanto diminuem os problemas com o desenvolvimento de software. Tem o foco na entrega de código executável e vê nas pessoas o fator principal no desenvolvimento de um produto. (MAURER; MARTEL, 2002)

Visando o desenvolvimento de um software livre, que atenda as necessidades dos órgãos públicos brasileiros no contexto de medição de tamanho funcional de software, uma parceria com a STI (Secretaria de Tecnologia da Informação) foi estabelecida por meio deste trabalho.

1.2 Descrição do Problema

Várias ferramentas disponíveis no mercado se propõe e tratar da medição de tamanho funcional de software. Entretanto, os órgãos do governo federal tem utilizado basicamente planilhas para armazenar e controlar os dados das medições, informação confirmada em reunião com a Secretaria de Tecnologia da Informação(STI), a qual prepara uma portaria recomendando aos órgãos a adoção de ferramentas.

O problema do uso de planilhas para gerenciar as medições é a dificuldade no controle das informações, o que impossibilita a criação de baselines. Ainda de acordo com o STI, um dos principais fatores que levam os órgãos a não adotarem ferramentas é a usabilidade ruim, sendo mais fácil para os servidores digitar os dados em planilhas.

A tabela 1 representa os atributos avaliados durante um estudo de ferramentas realizado durante o presente trabalho.

Tabela 1 – Avaliação de Ferramentas

Nome	Windows/Linux	Paga	Versão Demo ou Free	Continua a ser evoluída	Software Livre
SCOPE		X		X	
FUNCTION POINT WORKBENCH		X	X	X	
Sfera		X			
FPModeler	X	X		X	
Metric Studio			X	X	
WINFPA			X		
Sizify	X	X	X	X	

Uma análise de ferramentas presentes no mercado, mostrou a escassez de ferramentas livres. Outra observação a ser feita é a de que nenhuma ferramenta atende as necessidades atuais dos órgãos públicos brasileiros, seja por lacunas em seus requisitos funcionais, ou em seus requisitos não funcionais.

Tabela 2 – Análise de funcionalidades das ferramentas

Nome	Criação de Baselines	Gerência de Múltiplos Projetos	Medição de Esforço	Geração de Gráficos
SCOPE	X	X	X	X
FUNCTION POINT WORKBENCH	X	X	X	X
Sfera	X	X		
FPModeler	X	X		X
Metric Studio	X		X	X
WINFPA		X	X	
Sizify	X	X		

O apêndice deste trabalho traz informações mais detalhadas acerca da análise das ferramentas e fatores observados para a caracterização das mesmas.

Portanto, este trabalho tem como objetivo o desenvolvimento de uma ferramenta livre, para realizar a contagem de pontos de função e o armazenamento de um histórico das mesma, com a STI como pilar de apoio para a obtenção e validação dos requisitos da presente ferramenta.

1.3 Objetivos

O objetivo deste trabalho é desenvolver uma ferramenta para contagem e armazenamento de pontos de função, em parceria com a STI, que se adeque ao contexto dos órgãos públicos brasileiros e suas necessidades.

Também foram definidos os seguintes objetivos específicos:

- Levantar requisitos para o sistema;
- Desenvolver uma solução livre;
- Selecionar um framework para o front-end que permita atender a usabilidade requerida pelos usuários;
- Definir questões de segurança de dados;
- Selecionar linguagem e framework para desenvolvimento back-end;
- Definir um processo de desenvolvimento de software para o contexto de desenvolvedor único;
- Disponibilizar ferramenta no Portal do Software Público

1.4 Organização do Trabalho

Este trabalho está organizado em cinco capítulos, incluindo este capítulo, o introdutório, que disserta sobre o contexto, a descrição do problema, os objetivos do trabalho, e a organização do mesmo.

O capítulo 2 apresenta os conceitos relativos ao Extreme Programming, suas práticas e as diferenças de sua aplicação em um contexto onde apenas um programador desenvolve um projeto inteiro.

O capítulo 3 aborda os conceitos relativos ao Ponto de Função, além de sua história, método de contagem, desvantagens em sua utilização e por fim a importância do mesmo na indústria de desenvolvimento de software.

O capítulo 4 descreve a metodologia empregada neste trabalho, ou seja, o processo de desenvolvimento de software adaptado ao contexto de um desenvolvedor, baseando-se nas práticas e conceitos do Extreme Programming.

O último capítulo apresenta os resultados obtidos até o momento, escolha de ferramentas e opções tecnológicas para o desenvolvimento da segunda parte deste trabalho.

Parte II

Referencial Teórico

2 Extreme Programming

Este capítulo fará uma abordagem a metodologia chamada *Extreme Programming*. Serão tratadas suas características, benefícios de sua utilização além dos pilares que definem a metodologia: seus valores, princípios e práticas. Também abordará um contexto diferente ao desenvolvimento de software por meio de um time, quando o desenvolvedor trabalha sozinho em um projeto.

2.1 Metodologias ágeis

As abordagem de desenvolvimento de software vem mudando drasticamente na última década. Existem várias desvantagens acerca das metodologias tradicionais e bem documentadas, consideradas muito "pesadas" para algumas abordagens. Em resposta a essas abordagens, um novo grupo de metodologias têm aparecido nos últimos anos. Estas metodologias são as conhecidas como metodologias ágeis. (AGARWAL; UMPHRESS, 2008)

As metodologias ágeis envolvem menos documentação e são mais focadas no código fonte do produto. Elas são mais adaptativas do que as consideradas "tradicionais". Muitas metodologias hoje caminham para o lado ágil. Todas possuem características similares, mas também diferenças significativas. Entre as várias metodologias ágeis dos dias de hoje, podemos destacar o XP(*Extreme Programming*), *Scrum* e o *Crystal.* (AGARWAL; UMPHRESS, 2008)

Neste trabalho apenas nos aprofundaremos na metodologia XP. Seus fundamentos e princípios serão a base para a construção da metodologia e da definição do processo de desenvolvimento de software descritos no próximo capítulo deste trabalho.

2.2 O que é o XP

Segundo Kent Beck, (BECK, 2004) *Extreme Programming* é sobre mudança social. é sobre deixar ir embora nossos velhos hábitos e manias que antes faziam parte de uma adaptação forçada, mas que hoje em dia deixam de ser adaptação, para ser aquilo que define como fazer o nosso trabalho da melhor forma possível. É sobre abrir mão das defesas que nos protegem, mas interferem em nossa produtividade.

Extreme Programming é um estilo de desenvolvimento de software focado na excelência na aplicação de técnicas de programação, comunicação clara e trabalho em equipe. O XP também pode ser descrito como uma filosofia de desenvolvimento de software ba-

seada em valores, sendo eles: comunicação, feedback, simplicidade, coragem e respeito. (BECK, 1999)

Além dos valores, a filosofia de desenvolvimento do XP também possui princípios e suas práticas. Os princípios fundamentais são o foco no aspecto humano e social do desenvolvimento de software, na qualidade do software e na busca do benefício mútuo de desenvolvedores e clientes de um projeto.(BECK, 2004)

O *Extreme Programming* nada mais é do que uma junção harmoniosa destas três vertentes. Kent Beck descreve em seu livro uma ponte, que representa os três elos presentes no XP. A figura abaixo representa a ponte descrita por Kent, nela podemos ver o hiato entre os valores e as práticas presentes no XP, estes são os princípios presentes na metodologia.(BECK, 2004)

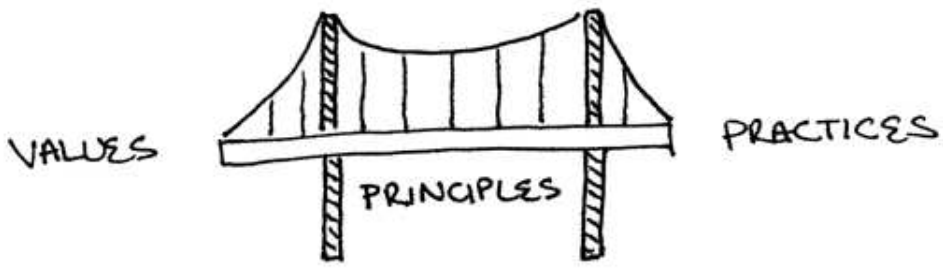

Figura 1 – Representação Visual do XP (BECK, 2004)

A seguir serão descritas características gerais das práticas propostas pela metodologia, fundamentais para o entendimento do ciclo de desenvolvimento do XP.

2.3 Práticas

O conjunto de práticas do XP tem a finalidade de aumentar a produtividade enquanto mantém a qualidade do produto. (MAURER; MARTEL, 2002) As práticas são o último elo que compõe a ponte descrita por Kent Beck. A sustentação e travessia da mesma depende de todos os elementos que a compõe. A figura 2 mostra um detalhamento das práticas presentes no XP.

Podemos descrever um número enorme de práticas neste trabalho, entretanto o foco será apenas descrever as práticas mais utilizadas pelo mercado e aquelas que, de alguma forma, agreguem valor à construção de uma metodologia de desenvolvimento baseada no XP.

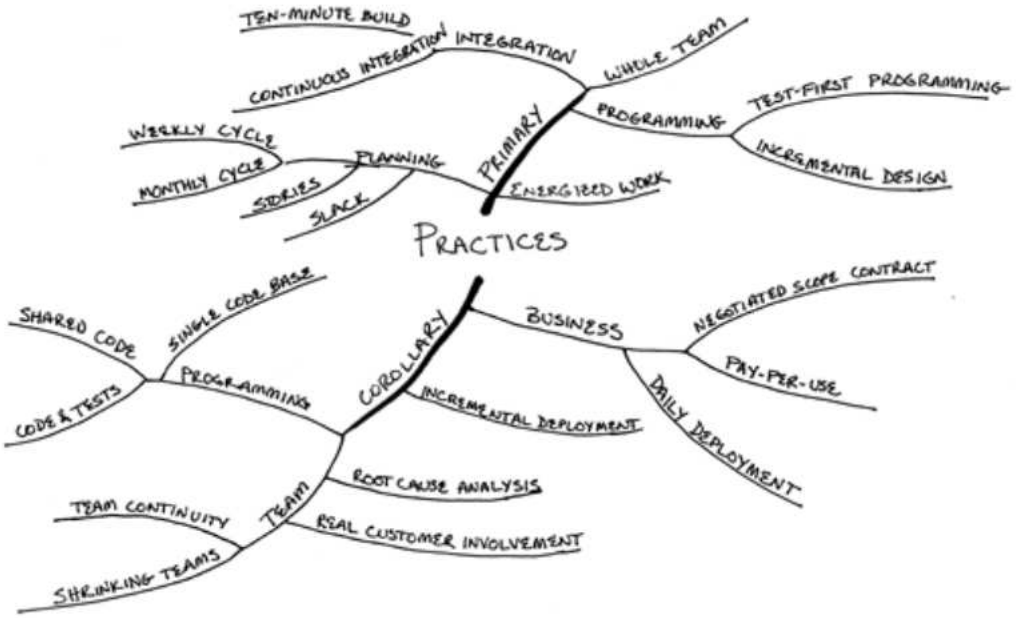

Figura 2 – Práticas do XP (BECK, 2004)

(MAURER; MARTEL, 2002) descrevem de forma sucinta as práticas mais utilizadas pelo mercado e mais importantes dentro de um ciclo de desenvolvimento de software:

- **Envolvimento dos *Stakeholders*:** Determinar e priorizar os requisitos é uma prática fundamental para o sucesso de um projeto. Esta prática envolve agregar valor ao desenvolvimento de software. Conforme as mudanças emergem, os desenvolvedores podem consultar o cliente para entender suas necessidades e prioridades ao invés de especular através de uma documentação robusta o que seria a prioridade.

- **Pequenas Versões:** Devido a mudança constante de requisitos, o XP mantém ciclos com pequenas versões para garantir que cada das mesmas produza um software com valor ao cliente. Ciclos pequenos com lançamento de pequenas versões reduz os riscos e permite a adaptação dos desenvolvedores as mudanças dos requisitos.

- **Metáfora:** Uma metáfora representa uma visão do sistema que pode ser compreendida por ambas as partes, tanto técnica quanto de negócios. Geralmente é uma ideia que representa as funcionalidades básicas do sistema a ser desenvolvido.

- **Testes:** Teste são uma parte fundamental do XP. Sejam testes de aceitação desenvolvidos pelos clientes, ou testes unitários construídos pelos desenvolvedores do software. Todos eles têm a função de prevenir erros e guiar os desenvolvedores na validação da funcionalidade após o código escrito.

- **Projeto Simples:** O XP não prevê esforço no desenvolvimento de uma solução robusta para um problema. Resolver os problemas de hoje, evitando redundâncias,

com o menor número de classes e métodos possíveis é o objetivo do desenvolvedor. Códigos com um design simples facilitam a manutenção sem a necessidade de documentação extensa. Isso evita predições erradas sobre o produto, além do gasto de recursos para refatoração.

- **Refatoração:** A refatoração pode ser considerada como qualquer mudança que melhore a manutenibilidade e o entendimento do código fonte, mas não alterando a funcionalidade do mesmo. (FOWLER, 1999) A refatoração é um fator importante dentro do projeto devido ao fato da pouca documentação. O código deve ser de simples entendimento e compreensão, assim como a forma mais simples de código para que os testes escritos rodem.

- **Programação em pares:** No XP a produção do código é feita por duas pessoas trabalhando em uma única máquina. Uma pessoa controla teclado e mouse, pensando na resolução do problema e se aquela abordagem funcionará para determinada situação. A segunda pessoa pensa de forma mais estratégica, avaliando se a solução aplicada é a melhor possível para desenvolver a funcionalidade proposta. As posições são trocadas constantemente durante o dia.

- **Jogo de Planejamento:** O objetivo desta prática é planejar o escopo do próximo ciclo de desenvolvimento. São avaliadas as prioridades dos desenvolvedores e do negócio. São definidas quais funcionalidades mais importantes, quais podem ser postergadas e quais são essenciais para o desenvolvimento do sistema. Ao final do planejamento são definidas as funcionalidades a serem desenvolvidas, de acordo com a experiência do desenvolvedores e com o espaço de tempo para o próximo ciclo.

- **40 horas por semana:** De acordo com a metodologia, nenhuma pessoa pode trabalhar mais de 40 horas por semana sem que sua produtividade seja afetada. Mesmo que os desenvolvedores sejam pressionados a trabalhar além deste período na semana, o resultado, na maioria das vezes é queda de produtividade e qualidade do produto.

- **Propriedade Coletiva de Código:** Todos podem modificar qualquer parte do código a todo o momento. Não existe propriedade individual de nenhum trecho de código. A propriedade coletiva auxilia na troca de conhecimento entre os membros e evita problemas quando algum desenvolvedor deixa a equipe, já que todos os integrantes tem conhecimento de todas as partes do código.

- **Padrões de Codificação:** Padrões de código são uma obrigação em um projeto onde integrantes modificam constantemente partes do sistema. Padrões facilitam o entendimento e a manutenção além de tornar o código consistente.

- **Integração Contínua:** Desenvolvedores devem integrar o código sempre que possível. Integrando o código uma vez ao dia garante que sempre haverá um novo executável com novas funcionalidades para a validação do cliente.

Após uma breve explicação das características da metodologia e um aprofundamento em suas práticas, a próxima seção descreve-rá o ciclo de trabalho proposto pelo Extreme Programming.

2.4 Visão de Projeto no XP

O XP deixa o processo de desenvolvimento tradicional de lado. Em vez de analisar, planejar e desenvolver um design visando um futuro muito distante, o XP explora a redução de custos na mudança do software, desenvolvendo um pouco de cada uma dessas atividades durante o ciclo de desenvolvimento de software. (BECK, 1999) A figura 3 ilustra o ciclo de desenvolvimento de um projeto na visão do XP.

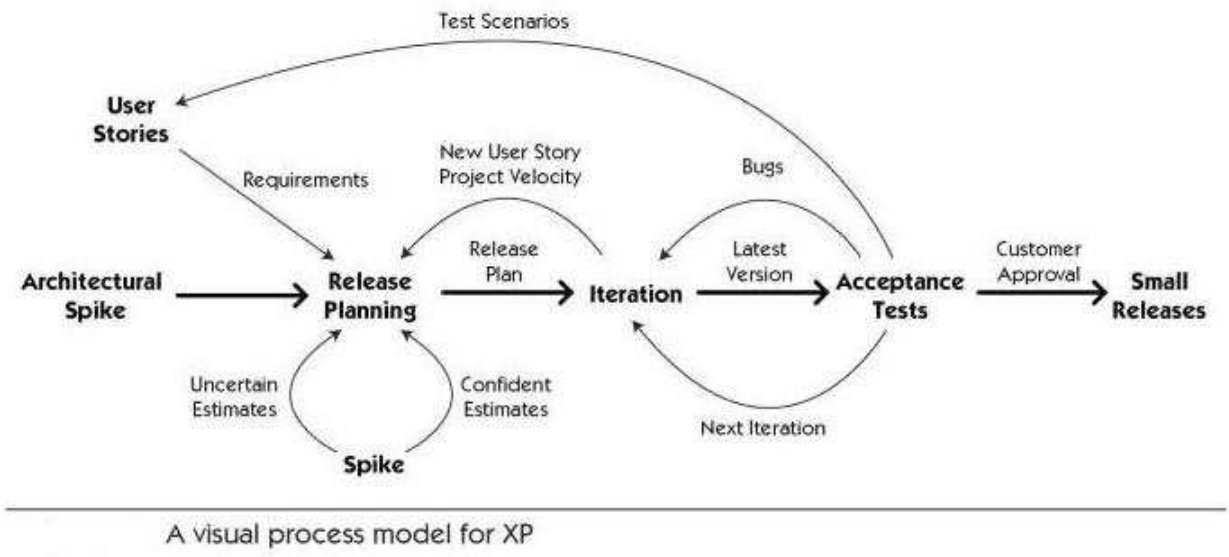

Figura 3 – Processo de Desevolvimento XP (EXPERTIZA, 2016)

O processo se inicia com o cliente priorizando as features mais significativas dentro de todas levantadas para a definição de uma release. Uma release é uma versão estável do produto com um grupo de funcionalidades implementadas. Após uma seleção das mesmas, os desenvolvedores o orientam acerca do tempo para a implementação e seus custos. (BECK, 1999) Essas estimativas são obtidas a partir das experiências e entendimento dos riscos pelos desenvolvedores. (BERNABÉ; NAVIA, 2015)

Uma iteração se inicia com a escolha das features (chamadas de user stories no XP) mais significativas dentre as previamente selecionadas para a release, Novamente os

desenvolvedores informam o custo e o tempo para o desenvolvimento e de acordo com o tempo definido para a iteração um escopo de desenvolvimento é definido. (BECK, 1999)

Os desenvolvedores transformam as user Stories em tarefas, as quais cada um tomará responsabilidade durante o ciclo. Destas tarefas são derivados casos de testes e a aceitação destes irão demonstrar a conclusão das tarefas ao final da iteração. (BECK, 1999)

As iterações acontecem até que uma pequena release seja construída. Durante este período os desenvolvedores devem fazer refatorações no código, práticas o pareamento e desenvolvedor a arquitetura da forma mais simples e manutenível possível. (BECK, 2004)

O ciclo de desenvolvimento do XP é planejado para que um time possa desenvolver suas atividades de uma maneira eficiente e com foco na qualidade do produto. Entretanto, em alguns casos o desenvolvedor é obrigado a desenvolver um projeto sozinho, seja pelo tamanho do mesmo ou por limitações de uma equipe. Nesses casos devemos fazer adaptações a esta metodologia .

A próxima seção terá um foco nas diferenças entre um ciclo de desenvolvimento para um time e na situação onde apenas um desenvolvedor é responsável por todas as etapas do processo.

2.5 Um desenvolvedor no papel de time

O XP foca em situações que envolvem times pequenos, onde o desenvolvimento é feito em pares. Muitas das práticas do XP necessitam de trabalho em equipe, algum desenvolvedor revisando o código de outro. Todo o desenvolvimento no XP, feito em pares, necessita da revisão constante de código. (JEFFRIES, 2000)

Existem casos onde a programação em pares é uma prática intangível, onde um único programador trabalha no projeto. Como um consultor independente, o desenvolvedor não terá um parceiro para a prática da técnica de pareamento. Em um projeto muito pequeno ou algo experimental, o pareamento pode ser um problema devido ao tamanho das tasks. (AGARWAL; UMPHRESS, 2008)

Nestes casos algumas adaptações ao processo e as práticas descritas pelo XP são necessárias. As práticas devem ser adaptadas ao desenvolvedor para que não sejam perdidos seus benefícios. O processo não deve ser tornar oneroso, a adaptação do mesmo e de suas fases se faz necessária para diminuir custos e tempo de desenvolvimento. (BERNABÉ; NAVIA, 2015)

Em seu artigo, (BERNABÉ; NAVIA, 2015) propõe a adaptação do processo de desenvolvimento proposto pelo XP. A figura 4 representa visualmente o processo descrito pelo autor:

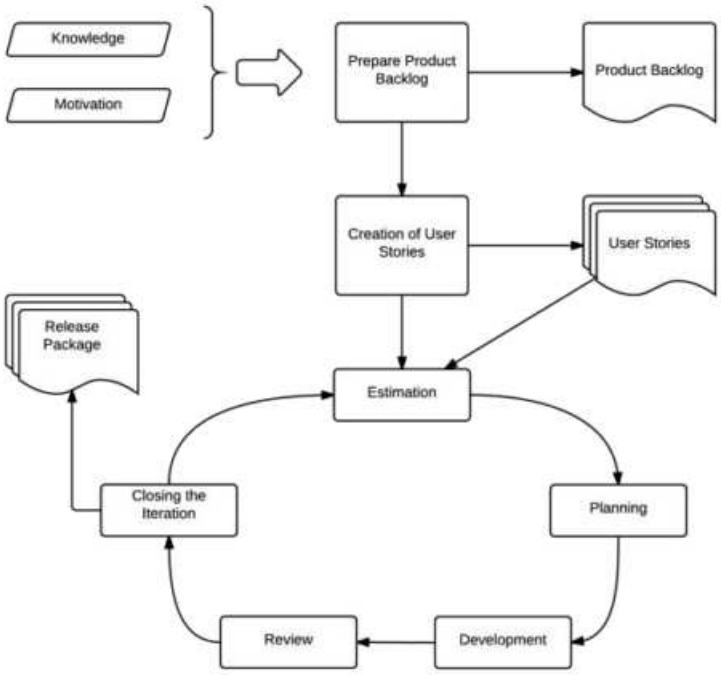

Figura 4 – Processo Adaptado XP (BERNABÉ; NAVIA, 2015)

A primeira fase do processo se resume ao conhecimento e a motivação de desenvolver o produto. O conhecimento sobre tecnologias e avaliação de riscos futuros pode ser muito complexo para apenas um desenvolvedor. A motivação para o desenvolvimento é fundamental, já que o esforço necessário e a gestão pessoal serão grandes inicialmente. (BERNABÉ; NAVIA, 2015)

A definição do *backlog* do produto, ou um conjunto de features que compõe todo o projeto, se mantém da mesma forma. A única diferença é o local do armazenamento. O autor recomenda fortemente o uso de ferramentas ou "quadros" digitais. Sua portabilidade e flexibilidade se fazem mais úteis ao desenvolvedor do que quadros físicos em uma parede que muitas vezes não trarão visibilidade.

A criação das *user stories* se torna mais simples. O entendimento das mesmas será feito apenas por um desenvolvedor, o que evita lacunas na comunicação de como uma *feature* deve ser implementada.

Um dos maiores desafios em um cenário com um único desenvolvedor, é a estimativa das *user stories*. Não apenas pela falta de proximidade de um cliente, mas por riscos não identificados ou deficiências técnicas que não poderão ser supridas por outros membros da equipe. Nestes casos a única recomendação é comparar as estimativas de cada tarefa a cada iteração para se obter um esforço médio na estimativa das próximas. (BERNABÉ; NAVIA, 2015)

O planejamento das iterações devem ser feitos priorizando aspectos como duração

e quantidade de tarefas a serem feitas. Para um desenvolvedor uma iteração deve durar entre uma ou duas semanas, respeitando um limite de 30 horas semanais. O XP propõe 40 horas semanais, mas desenvolvedores *"freelancers"* muitas vezes trabalham em mais de um projeto semanalmente. Desenvolver uma metáfora para explicar o objetivo da iteração pode ajudar o desenvolvedor a cumprir seus objetivos de forma mais rápida. Vale ressaltar a necessidade de avaliar a experiência do desenvolvedor na hora de selecionar a quantidade de *tasks* a serem executadas durante um ciclo de desenvolvimento. (BERNABÉ; NAVIA, 2015)

Na fase de desenvolvimento o desenvolvedor deve seguir as mesmas práticas e observações propostas pelo XP. A diferença desta fase é a ausência do pareamento para garantir uma maior qualidade de código. Uma solução para esse problema pode ser a técnica conhecida como *Rubber Duck Debugging*. (ERRINGTON, 2002) A técnica consiste em colocar um pato de borracha ao seu lado para simular um pareamento. A ideia é que explicando o problema e seu objetivo ao pato, o desenvolvedor consiga identificar o que o código não está fazendo, mas deveria.

As metodologias ágeis incluem uma melhora contínua a cada final de release ou iteração. A vantagem para um único desenvolvedor é que a reavaliação em busca de melhorias pode ser feita várias vezes e sobre todos os aspectos do desenvolvimento. (BERNABÉ; NAVIA, 2015) A fase de revisão visa a melhoria destes aspectos e a preparação para um novo ciclo de desenvolvimento.

Após a revisão das *user stories* e a avaliação de sua completude, o desenvolvedor deve disponibilizar uma versão com as *features* desenvolvidas naquela iteração. *Pequenas releases* são práticas do XP e devem ser mantidas para o aumento de qualidade do código e valor para o cliente.

Esta seção abordou a visão do XP para um desenvolvedor e algumas adaptações a serem feitas no processo de desenvolvimento de software. A próxima seção tratará especificamente das diferenças entre as práticas do XP no contexto de time e no contexto onde apenas um desenvolvedor atua.

2.6 Práticas para apenas um desenvolvedor

Além das adaptações ao processo de desenvolvimento mostradas na seção anterior, o contexto onde apenas um desenvolvedor desenvolve um projeto também necessita de adaptações às práticas descritas pelo XP. (AGARWAL; UMPHRESS, 2008) buscou estabelecer uma nova metodologia denominada *PXP(Personal Extreme Programming)*. Em seu artigo, o autor busca comparar as práticas em contextos onde um time atua e onde um desenvolvedor sozinho assume a responsabilidade de todo o projeto. A tabela abaixo busca mostrar uma síntese de suas observações:

Tabela 3 – Análise de Práticas no PXP (AGARWAL; UMPHRESS, 2008)

Práticas	PXP
Envolvimento dos Stakeholders	Se inicialmente você for seu próprio cliente, definir você mesmo as necessidades do projeto é compreensível. Caso existam clientes, o contato deve ser mantido constantemente com os interessados no projeto.
Pequenas Versões	Versões podem ser lançadas pelo desenvolvedor mais facilmente e de forma mais rápida, já que conflitos no código ocorrem com uma frequência muito baixa.
Metáfora	Um único desenvolvedor pode refatorar a metáfora que define o sistema diversas vezes. É mais fácil deixar clara a visão de uma parte do sistema para uma pessoa do que para um time inteiro.
Testes	Testar o código feito por apenas uma pessoa é simples. O uso de ferramentas de automação de testes também é facilitada devido ao baixo número de conflitos ou quebras de código fonte.
Projeto Simples	Manter o projeto simples com um desenvolvedor é fácil. Em paralelo pode ser desenvolvida uma solução mais robusta, visando uma refatoração futura.
Refatoração	A refatoração deve ser praticada ao extremo. Todo o código refatorado deve ser testado e integrado imediatamente, já que o mesmo não precisa de permissão do time para ser integrado.
Programação em Pares	Quando se trabalha sozinho os benefícios do pareamento são perdidos. Uma solução seria pedir a um amigo para revisar seu código, ou técnicas como o *Rubber Duck Debugging*, citado anteriormente.
Jogo de Planejamento	Se o desenvolvedor conhece o suficiente para se passar pelo cliente, basta que ele inverta os papéis por um tempo para escrever as user stories. É uma boa prática estimar, pontuar e priorizar as features mesmo quando está se desenvolvendo sozinho.
40 horas por semana	A prática se mantém igual. Não devemos trabalhar mais que 40 horas, pois a produtividade e qualidade do código é afetada pelo stress.
Propriedade Coletiva de Código	Você é dono de todo o código produzido, então não existem problemas com a propriedade do código. Entretanto, você não será beneficiado pelo código de outros desenvolvedores, que em muitos casos pode ser de grande valia. Técnicas para controle de versão de código são mantidas neste caso.
Padrões de Codificação	O desenvolvedor define o padrão de codificação pela forma que desenvolve. Vale ressaltar a importância de se seguir um padrão mesmo em situações como essa.
Integração Contínua	Se um desenvolvedor trabalha sozinho o código base não sofrerá conflitos vindos de outros desenvolvedores. Mesmo assim a integração contínua ainda é necessária para a integração mais rápida de código novo ao código base.

Este capítulo abordou a metodologia conhecida como XP, suas características e práticas. Além de fazer uma comparação com uma abordagem da metodologia quando utilizada por apenas um desenvolvedor.

O próximo capítulo trará uma visão geral sobre os Pontos de Função e sua utilização para medição de tamanho funcional de software.

3 Tamanho Funcional de Software

Este capítulo descreve o surgimento da contagem de tamanho funcional, a técnica de contagem de pontos de função (IFPUG) pelo manual do CPM.

3.1 Tamanho Funcional de Software

A contagem de tamanho funcional de software surgiu no ano de 1979, com Albrecht. A proposta era estimar o tamanho de um software a partir das funcionalidades entregues ao usuário. Estas primeiras métricas ficaram conhecidas como *Function Points(FP)* e *Function Points Analysis (FPA)*. Eram uma alternativa para suprir o problema da indústria em estimar prazos e o esforço necessário para a o desenvolvimento de software. (GENCEL; DEMIRORS, 2008)

Nos anos seguintes surgiram várias propostas de melhoria e variações do modelo apresentado no ano de 1979. No ano de 1986 a International Function Point User Group (IFPUG) foi criada como uma organização sem fins lucrativos. Sua função era, entre algumas outras, a de promover e disseminar o gerenciamento de projetos através do uso de FPA. Em 1996 a International Organization for Standardization (ISO), estabeleceu princípios de comum entendimento e uma interpretação consistente das definições que permeiam a medição de tamanho funcional. (GENCEL; DEMIRORS, 2008)

Atualmente a ISO/IEC 20926:2010 regulamenta a análise de pontos de função. Ela define regras e etapas para aplicação da mesma em projetos de desenvolvimento e manutenção de software. (JUNIOR; FANTINATO; SUN, 2015)

3.2 Contagem de Pontos de Função

O manual de contagem do IFPUG descreve o procedimento de contagem de pontos de função a partir de alguns passos: (IFPUG, 2010)

- Definir as fronteiras de medição, o escopo e o propósito da mesma. A fronteira de um software pode ser definida estabelecendo uma fronteira lógica entre o software a ser medido, seus usuários e a interação com outros softwares. Essa fronteira pode ser subjetiva, consequentemente tornando difícil a delimitação do início de um software e do término de outro. (JUNIOR; FANTINATO; SUN, 2015)

- Medir as funções de dados. Uma função de dados refere-se aos requisitos na visão do usuário em relação ao armazenamento e a referência de dados da aplicação. Podem

ser classificadas em arquivos lógicos interno e externos.

- Medir as funções de transação. Estas funções são caracterizadas pelo processamento de dados por uma funcionalidade provida ao usuário. Podemos classificá-las em três tipos: saída externa, entrada externa e consulta externa e definir a complexidade de cada uma como baixa, média e alta.
- Calcular o tamanho funcional. A partir da soma da complexidade das funções da dos e das funções de transação é possível calcular um tamanho total em pontos de função de um software. Vale ressaltar que existem aspectos para ajustar os pontos de função de acordo com a necessidade do projeto.
- Reportar os resultados. O último passo listado pelo manual corresponde ao relato dos resultados obtidos pela contagem, além de interpretações e possíveis tomadas de decisões a partir dos valores obtidos.

A figura a seguir representa os passos descritos anteriormente para a realização do procedimento de contagem de pontos de função:

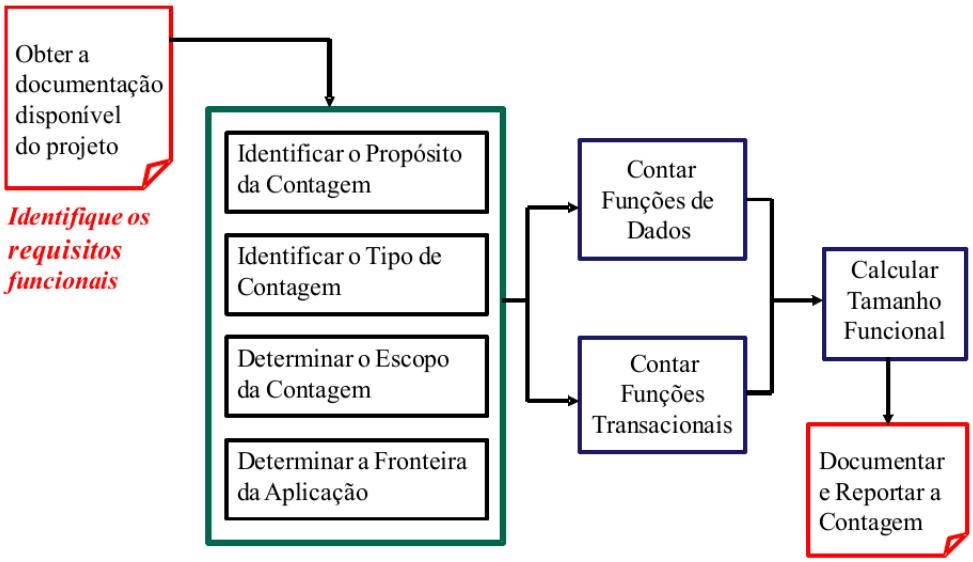

Figura 5 – Processo de contagem de Pontos de Função (SISP, 2016)

As tabelas abaixo descrevem a pontuação de funções de dados e funções de transação de acordo com a complexidades medidas durante a contagem:

3.3 Problemas com o Ponto de Função

Apesar da eficiência da contagem de pontos de função, ao longo dos anos estudos foram feitos a fim de descobrir problemas, lacunas do processo e possíveis melhorias para o mesmo. Buscando saber quais seriam esses problemas e alternativas para solucionar essas

lacunas, em 2015 três autores brasileiros realizaram um processo de revisão sistemática na literatura para agrupar as observações feitas por outros autores na área.

Os autores classificaram os problemas encontrados em três tipos: Peso e complexidade, independência de tecnologia e ajuste da pontuação. As principais críticas a parte de peso e complexidade da contagem de pontos de função diz respeito a mesma classificação em complexidade de duas funções de transação ou de dados com diferentes quantidades de campos e referências a arquivos. Outra crítica seria o espaçamento entre as complexidades. Alguns autores acreditam que definir apenas como baixa, média e alta pode ser muito e alto e acabar não detalhando de forma mais aproximada a real complexidade de desenvolvimento de uma funcionalidade. (JUNIOR; FANTINATO; SUN, 2015)

Problemas relacionados a independência de tecnologia também são citados na revisão sistemática feita pelos autores. O principal questionamento dos pesquisadores é a falta de adaptação do processo de contagem às novas linguagens de programação. A abordagem não reflete o atual desenvolvimento de software, principalmente o advento e crescimento da orientação a objetos. A independência de tecnologias também diz respeito ao hardware, muito diferente hoje do que a existente a 20 anos atrás. (JUNIOR; FANTINATO; SUN, 2015)

Por fim, o estudo relata problemas como ajuste do ponto de função. Alguns autores afirmam que o ajuste não considera importantes requisitos não funcionais, dentre eles: usabilidade, manutenibilidade, eficiência, e portabilidade. (JUNIOR; FANTINATO; SUN, 2015)

3.4 Utilização de Pontos de Função

Mesmo com os problemas e lacunas descobertos ao longo dos anos, a indústria de desenvolvimento de software continuam usando a contagem de pontos de função para a estimativa de tamanho de software. Na visão de Ebert (EBERT; SOUBRA, 2014), as companhias passaram a adotar esse modelo de estimativa com algumas finalidades. A primeira delas seria alocar melhor os recursos a partir de uma primeira estimativa de prazo e esforço necessário para a produção. Outro fator seria a estimativa de esforço quando alguma mudança nos requisitos do projeto ocorresse. Além disso também seria possível medir a produtividade e taxas de entrega de software, possibilitando benchmarks e análises de pontos de melhoria no processo de desenvolvimento.

A utilização dos pontos de função para estimativas de projeto passou a ser usada também por órgãos de governo e não só por companhias ou fábricas de desenvolvimento de software. O Roteiro de Métricas de Software(2016) do SISP cita essa utilização do processo em órgãos e em empresas privadas, além dos benefícios obtidos por ambos pela utilização dos mesmo: “Diversas instituições públicas e privadas têm utilizado a métrica Ponto de

Função(PF) nas estimativas e dimensionamento de tamanho funcional de projetos de software devido aos diversos benefícios de utilização desta métrica, destacando-se: regras de contagem objetivas, independência da solução tecnológica utilizada e facilidade de estimativa nas fases iniciais do ciclo de vida do software." (SISP, 2016)

Parte III

Metodologia

4 Metodologia

Este capítulo trata da definição do processo de desenvolvimento de software utilizado no presente trabalho, além de um detalhamento acerca de suas atividades e práticas.

4.1 Definição do Processo

O processo de desenvolvimento de software foi construído com base nas práticas e valores do XP, porém com adaptações feitas para um único desenvolvedor. As macro-atividades tentam passar por todas as disciplinas importantes dentro da engenharia de software, além das práticas presentes e cotidianas vindas do XP.

A figura 6 representa na forma de fluxograma as macro-atividades definidas neste processo:

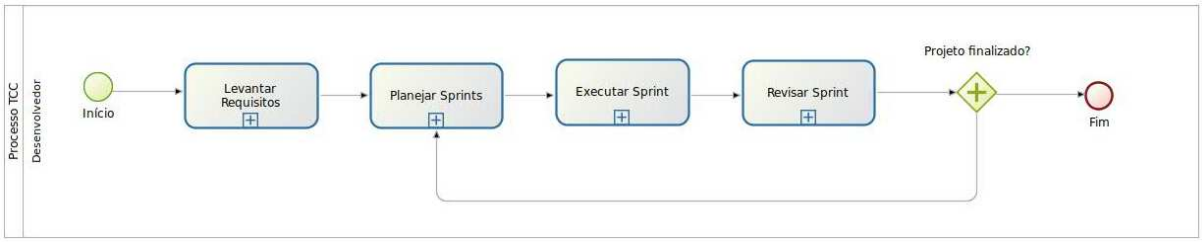

Figura 6 – Macro-atividades do Processo de Desenvolvimento

As macro-atividades tem como propósito o levantamento de requisitos, o desenvolvimento de um produto de qualidade e a entrega contínua de software. As próximas seções farão o detalhamento destas atividades.

4.2 Levantar Requisitos

Esta macro-atividade tem como propósito o entendimento das necessidades dos envolvidos no projeto, o levantamento das features a serem desenvolvidas e a criação de um backlog do produto com as features levantadas. A figura 7 representa as atividades do processo e seu fluxo.

- **Entender Necessidades:** Esta atividade tem o objetivo de entender as necessidades dos envolvidos. Neste caso a STI, como parceira de desenvolvimento, opinará acerca dos requisitos funcionais da aplicação.

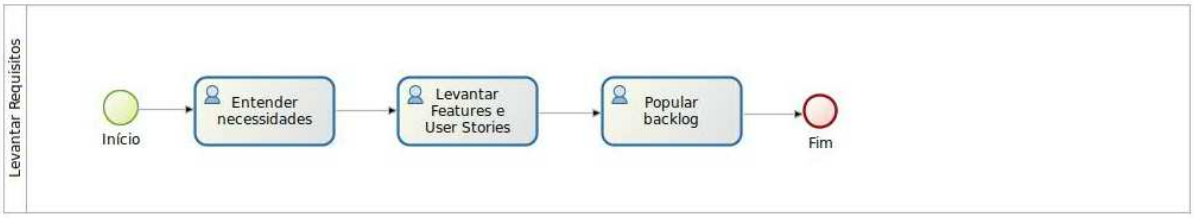

Figura 7 – Atividade de Levantamento de Requisitos

- **Levantar Features e User Stories:** Após o levantamento de requisitos, é necessária a definição das *User Stories* e *features*. Ambas estarão definidas e descritas como *issues* no repositório do projeto.

- **Popular Backlog:** Um *backlog* será criado no repositório do projeto, contendo as *User Stories* definidas no levantamento inicial do projeto. Durante o andamento do mesmo, novas *features* poderão ser acrescentadas.

4.3 Planejar Sprints

O foco desta atividade é priorizar as *issues* a serem desenvolvidas durante uma *sprint*, ou um ciclo de desenvolvimento em metodologias ágeis, tendo a duração de 15 dias. Além de adequar o escopo de cada ciclo à capacidade do desenvolvedor.A figura 8 representa as atividades do processo e seu fluxo.

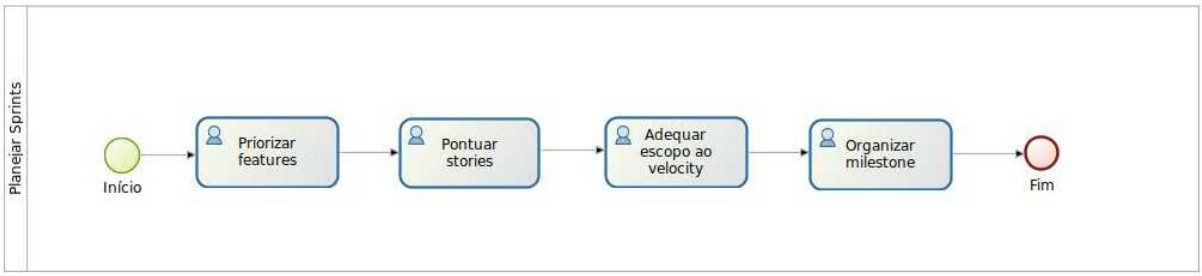

Figura 8 – Atividade de Planejamento de Sprints

- **Priorizar Features:** Esta atividade tem o objetivo de priorizar as *features* mais importantes, ou que trarão maior valor ao cliente dentro do projeto. Esta atividade será feita por meio de reuniões com a STI.

- **Pontuar Stories:** Definir um número que caracteriza a dificuldade do desenvolvimento de uma *user story*. Este número pode ser definido através de experiências dos desenvolvedores ou análise de riscos. É recomendado que essa numeração siga a sequência de Fibonacci.

- **Adquar Escopo ao Velocity:** Esta atividade tem a finalidade de avaliar a pontuação definida na priorização das *issues* a serem desenvolvidas na *sprint*. O desenvolvedor pode ter uma ideia de sua capacidade de produção comparando os valores da *sprint* atual com os anteriores. O número de pontos de uma sprint nunca deve ser muito maior que o da anterior ou da média das anteriores. Chamamos essa média de *velocity*.

- **Organizar Milestone:** Esta atividade se restringe ao repositório do projeto. Nele serão assinaladas as issues priorizadas para aquela sprint, além de uma descrição de sua finalidade e status.

4.4 Executar Sprint

Esta macro-atividade tem o objetivo de desenvolver as *features* priorizadas na atividade anterior, utilizando práticas presentes no XP, como integração contínua, testes automatizados e *deploy* de pequenos incrementos de software ao final de cada ciclo de desenvolvimento. A figura 9 representa em forma de fluxograma as atividades do processo.

Figura 9 – Atividade de Execução da Sprint

- **Escolher issue:** Definir uma *issue* a ser desenvolvida durante a sprint.

- **Criar Branch:** Cria uma nova *branch* para o desenvolvimento da *issue*. Ao final do desenvolvimento da mesma, essa *branch* será integrada a *branch* principal do projeto (Master).

- **Codificar:** Atividade padrão de codificação, onde práticas como testes unitários, refatoração, código simples e padrões de código serão aplicadas.

- **Testes de Integração:** Testes automatizados feitos por ferramentas. Testes como esse impedem a integração de *commits* a uma *branch* caso existam testes quebrados, ou a qualidade do código diminua.

- **Coletar Métricas:** Ferramentas de análise estática de código irão coletar métricas a fim de orientar o desenvolvedor acerca da qualidade de código.

- **Deploy de Incremento:** Após o término do desenvolvimento da *issue*, a *branch* será integrada a master e uma nova versão do produto será disponibilizada no servidor de produção.

Com a *issue* finalizada, caso houverem outras definidas para a *sprint*, o desenvolvedor repetirá o processo até que todas sejam finalizadas. A última atividade do processo encerra a *sprint*, fechando todas as *issues* no repositório.

4.5 Revisar Sprint

Esta atividade tem como objetivo revisar métricas de código a fim de que refatorações possam ocorrer nos próximos ciclos de desenvolvimento. Também tem como objetivo separar *issues* não finalizadas para que sejam realocadas na próxima *sprint*. A figura 10 representa em forma de fluxograma as atividades do processo.

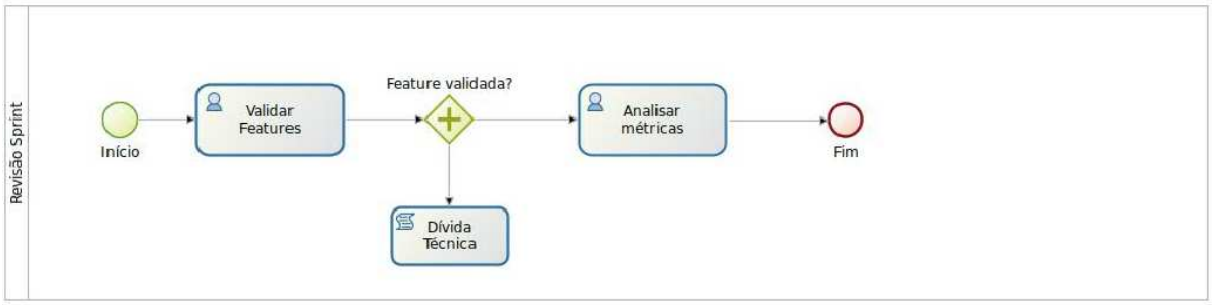

Figura 10 – Atividade de Revisão da Sprint

- **Validar Features:** Nesta atividade o desenvolvedor valida as *features* desenvolvidas durante a *sprint*. Esta validação pode ser feita através de testes de aceitação ou feita pela STI por comentários nas *issues* do repositório. Caso as *features* não sejam validadas ou não tiverem sido concluídas durante a *sprint*, são consideradas dívidas técnicas e separadas para serem novamente desenvolvidas na próxima *sprint*.
- **Analisar Métricas:** Ao fazer uma análise das métricas é possível aferir a qualidade do código e definir pontos para refatoração na *sprint* seguinte.

Este capítulo dissertou sobre a metodologia definida para o desenvolvimento do presente trabalho, além de detalhar suas atividades. O próximo capítulo fará um relato dos resultados obtidos até o momento.

Parte IV

Proposta

5 Proposta

Este capítulo tem como finalidade abordar os resultados obtidos até o momento. Serão discutidos tópicos como local do repositório, ferramentas escolhidas para o desenvolvimento e justificativas para a escolha das mesmas. Também serão abordados os requisitos obtidos para o desenvolvimento da aplicação.

5.1 Requisitos

Esta seção descreve os requisitos obtidos para o desenvolvimento da ferramenta. Alguns desses requisitos foram obtidos a partir do estudo realizado acerca das ferramentas disponíveis no mercado, enquanto o restante foi obtido a partir de reunião com a STI.

5.1.1 Requisitos Extraídos a Partir do Estudo das Ferramentas

A análise das ferramentas presentes no mercado possibilitou com que uma interseção entre os requisitos elicitados fosse feita. Seguem abaixo as *features* levantadas a partir desta interseção:

- Inserção, exclusão, edição e visualização de perfil de usuário;
- Inserção, exclusão, edição e visualização de organizações detentoras de
- projetos de desenvolvimento de software;
- Inserção, exclusão, edição e visualização de projetos de desenvolvimento de software;
- Gerenciamento de múltiplos projetos por organização;
- Criação de contagem de pontos de função para os softwares registrados;
- Geração de relatórios em pdf;
- Medição de esforço necessário para desenvolvimento de features;
- Visualização gráfica de atributos como tamanho total ao longo do tempo e esforço;
- Criação e gerenciamento de Baselines;
- Trankig acerca de mudanças nas contagens, como o autor da modificação e data da mesma

5.1.2 Requisitos Extraídos a Partir de Reunião com a STI

Além dos requisitos elicitados a partir do estudo das ferramentas, outros requisitos foram elicitados por meio de reunião com a STI. Nesta reunião foram abordadas necessidades específicas dos órgãos públicos brasileiros, e alguns esclarecimento acerca dos requisitos não funcionais da ferramenta.

Os requisitos funcionais obtidos foram:

- Gerenciamento e controle de acesso de usuários,
- Busca por projetos, organizações dados específicos de preenchimento como funções de transação ou de dados;

Requisitos não-funcionais também foram obtidos a partir desta reunião:

- Tratamento de fluxo de dados, devido sua grande quantidade;
- Questões de segurança como encriptação de dados, autenticação de usuários e bloqueio de acesso externo à aplicação;
- Desenvolvimento de formulários dinâmicos para preenchimento das informações acerca das contagens, facilitando a aceitação dos usuários ao sistema;
- Disponibilidade da aplicação para múltiplos sistemas operacionais

5.2 Definição de Ferramentas

Esta seção apresenta uma explicação sucinta de cada ferramenta utilizada para o desenvolvimento do projeto, assim como, quando necessário, o motivo de suas escolhas. A figura 11 ilustra todas as ferramentas a serem utilizadas no presente trabalho.

- **Ruby on Rails:** *Framework* de desenvolvimento de aplicações web por meio da linguagem de programação Ruby. Sua arquitetura se baseia em MVC (*Model, View e Controller*) e será utilizada em modo API(*Application Programming Interface*), ou seja, será construída para que forneça um serviço a outra aplicação. Os dados serão disponibilizados via JSON(*JavaScript Object Notation*). A criação de uma API se deve ao fato de que um dos requisitos não funcionais da aplicação é o grande fluxo de dados. Separar o *back-end* da aplicação e o *front-end* melhora o desempenho da aplicação, além de facilitar a manutenção da mesma.
- **React:** Segundo sua própria documentação (REACTJS, 2014), o React é um *framework* Javascript desenvolvido para criação de interfaces interativas. A utilização

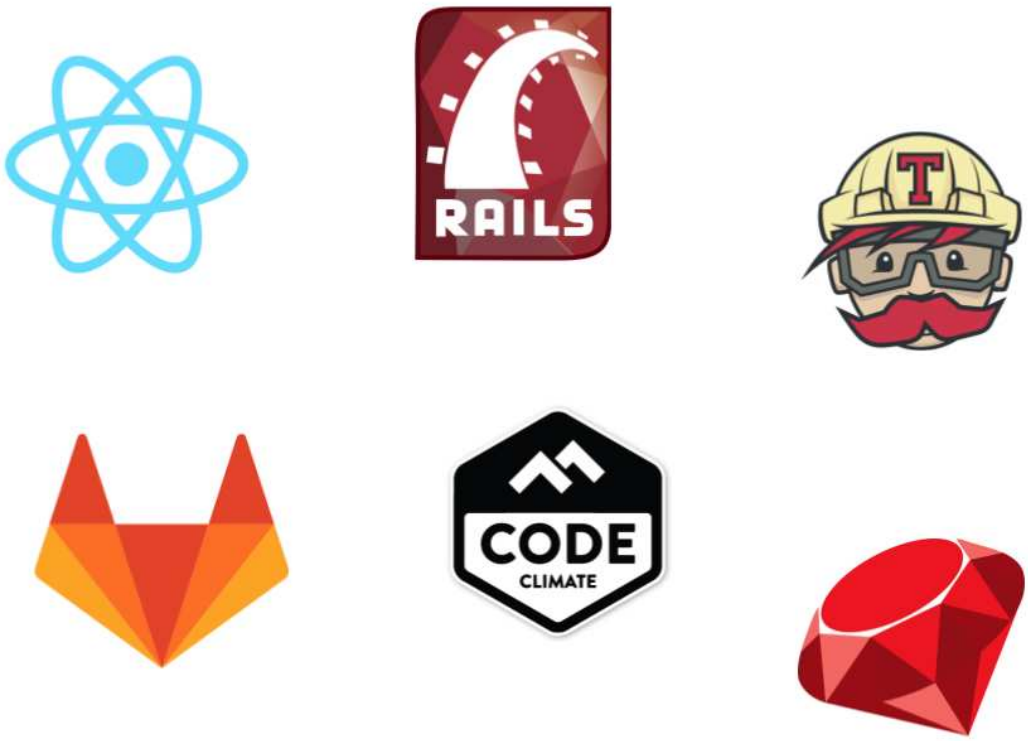

Figura 11 – Ferramentas Escolhidas para o Desevolvimento

de Javascript se deve ao fato da renderização de componentes ser feita a partir da diferenciação entre estados, ou seja, só serão renderizados novamente componentes cujo seus dados foram alterados. Esta aplicação consumirá os dados providos pela API através de requisições AJAX(*Asynchronous Javascript and XML*), sendo responsável também pela geração de *views* para interação do cliente com o servidor.

- **Travis:** Ferramenta para integração contínua de código. Quando um *commit* for integrado ao repositório da aplicação, a ferramenta criará uma máquina virtual, com configurações pré determinadas pelo desenvolvedor, para execução de testes unitários e de integração. Caso a *build* falhe, o Travis impede que o *commit* ou *merge* seja integrado a *branch*. Caso a *build* passe, é possível configurar a ferramenta para *deploy* automático em sites como Heroku.

- **Gitlab:** Provê controle de versão, revisão de código, *tracking* de *issues* além de espaço para a criação de uma wiki do projeto.

- **Code Climate:** Ferramenta para análise estática de código fonte, além de métricas como duplicação de código e complexidade ciclomática. Possui integração com o

repositório do aplicação. Caso algum merge entre branchs diminua a qualidade do código, o Code Climate previne o merge entre as duas branchs.

5.3 Repositório

No desenvolvimento deste trabalho serão utilizados dois repositórios. Um repositório fará o controle de versão da API desenvolvida através do *framework* 'Ruby on Rails', enquanto o outro repositório fará o controle de versão do 'React', framework Javascript para a construção de interfaces a partir de componentes.

Os repositórios podem ser encontrados nos seguintes links:

- **Ruby on Rails:** https://github.com/danielhmarinho/tcc1
- **React:** https://github.com/danielhmarinho/tcc1

Referências

AGARWAL, R.; UMPHRESS, D. Extreme programming for a single person team. In: *Proceedings of the 46th Annual Southeast Regional Conference on XX*. New York, NY, USA: ACM, 2008. (ACM-SE 46), p. 82–87. ISBN 978-1-60558-105-7. Disponível em: <http://doi.acm.org/10.1145/1593105.1593127>. Citado 5 vezes nas páginas 17, 31, 36, 38 e 39.

ALBRECHT, A. J. *Function point analysis, Encyclopedia of Software Engineering*. 1. ed. [S.l.]: John Wiley & Sons, 1994. Citado na página 25.

ANOTA, M. M. M. et al. Using free software tools: Middle education case in xalapa, veracruz. In: *2016 IEEE International Engineering Summit, II Cumbre Internacional de las Ingenierias (IE-Summit)*. [S.l.: s.n.], 2016. p. 1–5. Nenhuma citação no texto.

BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, Oct 1999. ISSN 0018-9162. Citado 3 vezes nas páginas 31, 35 e 36.

BECK, K. *Extreme Programming Explained: Embrace Change*. 2. ed. [S.l.]: Addison Wesley Longman, 2004. Citado 5 vezes nas páginas 15, 31, 32, 33 e 36.

BERNABÉ, R. B.; NAVIA. Faat: Freelance as a team. In: *Proceedings of the 3rd International Conference on Technological Ecosystems for Enhancing Multiculturality*. New York, NY, USA: ACM, 2015. (TEEM '15), p. 687–694. ISBN 978-1-4503-3442-6. Disponível em: <http://doi.acm.org/10.1145/2808580.2808685>. Citado 5 vezes nas páginas 15, 35, 36, 37 e 38.

EBERT, C.; SOUBRA, H. Functional size estimation technologies for software maintenance. *IEEE Software*, v. 31, n. 6, p. 24–29, Nov 2014. ISSN 0740-7459. Citado 2 vezes nas páginas 25 e 43.

ERRINGTON, A. *Rubber duck debugging*. 2002. Disponível em: <http://rubberduckdebugging.com/>. Citado na página 38.

EXPERTIZA. *XP Workflow*. 2016. Disponível em: <http://wiki.expertiza.ncsu.edu/images/5/5b/Extreme.jpg>. Citado 2 vezes nas páginas 15 e 35.

FOWLER, M. *Refactoring: Improving the Design of Existing Code*. Reading, Mass: Addison Wesley Longman, 1999. Citado na página 34.

GENCEL, C.; DEMIRORS, O. Functional size measurement revisited. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 17, n. 3, p. 15:1–15:36, jun. 2008. ISSN 1049-331X. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/1363102.1363106>. Citado na página 41.

IFPUG. *Function point counting practices manual, release 4.3.1*. Westerville, OH, USA: International Function Point Users Group, 2010. Citado na página 41.

JEFFRIES, R. *What is eXtreme Programming?* 2000. Disponível em: <http://www.xprogramming.com/what_is_xp.html>. Citado na página 36.

JOHNSON-EILOLA, J. Open source basics: Definitions, models, and questions. In: *Proceedings of the 20th Annual International Conference on Computer Documentation*. New York, NY, USA: ACM, 2002. (SIGDOC '02), p. 79–83. ISBN 1-58113-543-2. Disponível em: <http://doi.acm.org/10.1145/584955.584967>. Citado na página 25.

JUNIOR, M. de F.; FANTINATO, M.; SUN, V. Improvements to the function point analysis method: A systematic literature review. *IEEE Transactions on Engineering Management*, v. 62, n. 4, p. 495–506, Nov 2015. ISSN 0018-9391. Citado 2 vezes nas páginas 41 e 43.

KUSUMOTO, S. et al. Function point measurement from java programs. In: *Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002. (ICSE '02), p. 576–582. ISBN 1-58113-472-X. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/581339.581412>. Citado na página 25.

MAURER, F.; MARTEL, S. Extreme programming. rapid development for web-based applications. *IEEE Internet Computing*, v. 6, n. 1, p. 86–90, Jan 2002. ISSN 1089-7801. Citado 2 vezes nas páginas 26 e 32.

REACTJS. *What is React?* 2014. Disponível em: <https://facebook.github.io/react>. Citado na página 54.

SISP. *Roteiro de Métricas de Software do SISP: versão 2.2*. Brasília, DF, BR: Ministério do Planejamento, Desenvolvimento e Gestão. Secretaria de Tecnologia da Informação, 2016. Citado 3 vezes nas páginas 15, 42 e 44.

STALLMAN, R. *Free Software Foundation*. 2003. Disponível em: <http://www.gnu.org/philosophy/free-sw.html>. Citado na página 25.

Apêndices

APÊNDICE A – Primeiro Apêndice

Texto do primeiro apêndice.

APÊNDICE B – Segundo Apêndice

Texto do segundo apêndice.