

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

**PROTÓTIPO DE FERRAMENTA PARA AUXÍLIO NA
MEDIÇÃO E CONTROLE ESTATÍSTICO DE PROCESSOS**

PEDRO HUMBERTO MATTIOLLO

BLUMENAU
2015

2015/1-24

PEDRO HUMBERTO MATTIOLLO

**PROTÓTIPO DE FERRAMENTA PARA AUXÍLIO NA
MEDIÇÃO E CONTROLE ESTATÍSTICO DE PROCESSOS**

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Jacques Robert Heckmann , Mestre - Orientador

**BLUMENAU
2015**

2015/1-24

PROTÓTIPO DE FERRAMENTA PARA AUXÍLIO NA MEDICÃO E CONTROLE ESTATÍSTICO DE PROCESSOS

Por

PEDRO HUMBERTO MATTIOLLO

Trabalho de Conclusão de Curso aprovado
para obtenção dos créditos na disciplina de
Trabalho de Conclusão de Curso II pela banca
examinadora formada por:

Presidente: _____
Prof. Jacques Robert Heckmann, Mestre – Orientador, FURB

Membro: _____
Prof. Cláudio Ratke, Mestre – FURB

Membro: _____
Prof. Everaldo Artur Grahl, Mestre – FURB

Blumenau, 06 de Julho de 2015

Dedico este trabalho a minha família, principalmente ao meu pai Celso e minha Mãe Marizete. A minha namorada Dayanne pela paciência e compreensão em todas as horas gastas para a elaboração desse trabalho. Aos amigos, especialmente àqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pela fé e pelo amor.

À minha família, sem a qual não teria conseguido atingir meus objetivos na vida, inclusive o da conclusão deste.

À minha namorada, pelo incentivo, cobrança e auxílio na revisão do texto escrito neste.

Aos meus amigos, pelo conhecimento e incentivo fornecido.

Ao meu orientador, Jacques Robert Heckmann, pelo apoio e por ter acreditado na conclusão deste trabalho.

Nós somos aquilo que fazemos repetidamente.
Excelência, portanto, não é um ato, mas um
hábito.

Aristóteles

RESUMO

Este trabalho apresenta a implementação de um protótipo de ferramenta para auxiliar a análise e medição de software e, a partir disso, fornecer uma classificação mediante a capacidade e estabilidade do processo. As informações podem ser cadastradas manualmente ou podem ser capturadas de uma base de dados da ferramenta SonarQube. Elas podem ser resumidas em três unidades básicas: os softwares, as métricas ou itens a serem avaliados e as coletas dos valores. Depois disso, tomando como base cálculos específicos do Controle Estatístico de Processos (CEP), os dados podem ser submetidos a estes cálculos e serem apresentados em gráficos de controle para melhor visualização dos valores coletados. Por último, após a realização dos cálculos, os itens são classificados de acordo com sua estabilidade e capacidade.

Palavras-chave: Processos de software. CEP. Métricas.

ABSTRACT

This paper presents the implementation of a prototype tool to assist the software analysis and measurement and, from there, provides a classification by process capacity and stability. Information can be registered manually or it can be captured in a SonarQube tool database. They can be summarized in three basic units: software, metrics or items to be evaluated and collections of values. After that, taking specific of Statistical Process Control (SPC) calculations as a basis, the data may be submitted to these calculations and be presented in control charts for better collected values viewing. Finally, after performing the calculations, the items are classified according to their stability and capacity.

Key-words: Software processes. SPC. Metrics.

LISTA DE FIGURAS

Figura 1 - <i>DashBoard</i> inicial da ferramenta SonarQube.....	19
Figura 2 - Visão das atividades para aplicação de CEP	20
Figura 3 - Fluxograma do programa de medição proposto	21
Figura 4 - Arquitetura de <i>Data Warehousing</i> para PDS em evolução	22
Figura 5 - Diagrama de caso de uso	25
Figura 6 - Diagrama de classes de negócio	27
Figura 7 - Diagrama de classes de funcionalidades.....	29
Figura 8 - Modelo entidade relacionamento.....	30
Figura 9 - Diagrama de atividades.....	31
Figura 10 – Tecnologias utilizadas.....	32
Figura 11 - <i>Templates</i> Demoiselle.....	34
Figura 12 - Cadastro de fase do processo	44
Figura 13 - Cadastro de item	45
Figura 14 - Cadastro de software.....	45
Figura 15 - Cadastro de coleta.....	46
Figura 16 - Listagem dos itens	47
Figura 17 - Listagem de métricas da ferramenta SonarQube	48
Figura 18 - Confirmação da integração com a ferramenta SonarQube	48
Figura 19 - Gráfico de controle X-Barra Individual.....	49
Figura 20 - Fórmula do gráfico de controle X-Barra Individual	49
Figura 21 - Gráfico de controle EWMA.....	51
Figura 22 – Cálculo do gráfico EWMA	51
Figura 23 - Gráfico de Não conformidades/Defeitos	53
Figura 24 - Fórmula do gráfico de controle de Não conformidades/Defeitos.....	53
Figura 25 - Testes de estabilidade sobre um gráfico de controle	55
Figura 26 - Cálculo para o teste de capacidade do processo	55
Figura 27 - Classificação dos itens	56
Figura 28 - Arquivo exportado da listagem dos softwares.....	57

LISTA DE QUADROS

Quadro 1 - Arquivo Apache Maven pom.xml	35
Quadro 2 - Arquivo message.properties	36
Quadro 3 - Arquivo web.xml.....	37
Quadro 4 - Arquivo persistence.xml	38
Quadro 5 - Arquivo beans.xml.....	38
Quadro 6 - Arquivo xhtml gerado.....	39
Quadro 7 - Arquivo do pacote view	40
Quadro 8 - Arquivo do pacote controller	41
Quadro 9 - Arquivo do pacote converter	41
Quadro 10 - Arquivo do pacote persistence.....	42
Quadro 11 - Trecho de código para a geração do gráfico X-Barra Individual.....	50
Quadro 12 - Cálculo da amplitude móvel.....	50
Quadro 13 - Trecho de código para a geração do gráfico EWMA.....	52
Quadro 14 - Trecho de código para a geração do gráfico de Não conformidades/Defeitos.....	54
Quadro 15 - Comparativo entre os trabalhos.....	58
Quadro 16 - Descrição do UC04	62
Quadro 17 - Descrição do UC13	63
Quadro 18 - Descrição do UC14	63
Quadro 19 - Dicionário de dados.....	64

LISTA DE ABREVIATURAS E SIGLAS

CEP - Controle Estatístico de Processos

CMMI - *Capability Maturity Model Integration*

CRUD - *Create, Read, Update and Delete*

EWMA - *Exponentially Weighted Moving Average*

JPA - Java Persistence API

JSF - Java Server Faces

LC - Linha Central

LIC - Limite Inferior de Controle

LSC - Limite Superior de Controle

MR-MPS - Modelo de Referência para Melhoria do Processo de Software

MVC - *Model View Controller*

PDS - Processo de Desenvolvimento de Software

RF - Requisito Funcional

SEI - *Software Engineering Institute*

XML - *eXtensible Markup Language*

SUMÁRIO

1	INTRODUÇÃO	13
1.1	OBJETIVOS DO TRABALHO	13
1.2	ESTRUTURA DO TRABALHO	14
2	FUNDAMENTAÇÃO TEÓRICA.....	15
2.1	MEDIÇÃO DE SOFTWARE.....	15
2.1.1	Medição e Melhoria de Processos de Software.....	16
2.1.2	Controle Estatístico de Processos	17
2.2	MODELOS DE MATURIDADE CMMI E MR-MPS	17
2.3	FERRAMENTA SONARQUBE.....	19
2.3.1	SonarQube.....	19
2.4	TRABALHOS CORRELATOS	19
2.4.1	Modelo para Controle Estatístico de Processos de Desenvolvimento de Software (CEP-S).....	20
2.4.2	Programa de Medição para Organizações de Alta Maturidade.....	20
2.4.3	Um Método Evolutivo para Aplicação de Programas de Métricas em Processos de Desenvolvimento de Software	21
3	DESENVOLVIMENTO DO PROTÓTIPO.....	24
3.1	REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	24
3.2	ESPECIFICAÇÃO	24
3.2.1	Diagrama de casos de uso	25
3.2.2	Diagrama de classes	27
3.2.3	Diagrama de entidade relacionamento	29
3.2.4	Diagrama de atividades	30
3.3	IMPLEMENTAÇÃO	31
3.3.1	Técnicas e ferramentas utilizadas.....	32
3.3.2	Implementação do protótipo	33
3.3.3	Operacionalidade da implementação	42
3.4	RESULTADOS E DISCUSSÕES.....	57
4	CONCLUSÕES.....	59
4.1	EXTENSÕES	59

REFERÊNCIAS	61
APÊNDICE A – Principais casos de uso	62
APÊNDICE B – Dicionário de dados do MER.....	64

1 INTRODUÇÃO

Não é fato atual que a função de medir algo já faz parte do cotidiano das pessoas. Assim como um agrônomo mede o crescimento de uma planta, um nutricionista pode acompanhar e mensurar o peso do seu paciente. A necessidade do ser humano de se obter valores e, posteriormente, informações úteis para a tomada de decisões, pode revelar o uso das medidas. Na Engenharia de Software o cenário se repete:

Medir, também, é essencial na Engenharia de Software. É importante medir para entender e controlar processos, produtos e projetos. Medidas fornecem informações sobre objetos (processos, produtos e projetos) e eventos (por exemplo, a fase de testes de um projeto), tornando-os compreensíveis e controláveis. (BARCELLOS; ROCHA; SOUZA, 2012, p.14).

A medição de software, entretanto, não é uma atividade barata, principalmente quanto ao uso de recursos humanos e, não só isso, às vezes é utilizada somente para atender os requisitos de um modelo de maturidade de processos, como o Modelo de Maturidade em Capacitação e Integração (CMMI) ou o Modelo de Referência para Melhoria do Processo de Software (MR-MPS). Vários estudos têm mostrado que a maioria das organizações definem programas de medição precários e incapazes de produzir medidas que atendam às suas necessidades (BARCELLOS; ROCHA; SOUZA, 2012, p.14). Portanto, para serem efetivas, as medições de software precisam estar acopladas às regras de negócio da organização e aos seus objetivos de mercado e estarem disponíveis sempre que necessárias.

A partir deste cenário, propõe-se um protótipo de ferramenta para auxiliar na medição de software. A partir disso, fornecer-se-á uma classificação mediante a capacidade e estabilidade das métricas avaliadas, bem como índices e informações úteis para melhorar os processos da empresa, tomando como base o Controle Estatístico de Processos (CEP). O problema consiste em obter informações confiáveis e úteis e fornecer evidências que comprovem a classificação obtida, uma vez que os resultados podem ser utilizados para alterar o fluxo dos processos de toda uma empresa.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi o desenvolvimento de um protótipo de ferramenta capaz de auxiliar na medição de software aplicando os conceitos propostos pelo CEP, fornecendo o maior número de informações úteis.

Os objetivos específicos do trabalho são:

- a) prover uma entrada de dados manual e uma integração com a ferramenta SonarQube para recuperar as informações para análise;

- b) realizar cálculos e fornecer gráficos e planilhas sobre os dados coletados, com base nos índices do CEP;
- c) fornecer uma classificação da aplicação cadastrada de acordo com a estabilidade e capacidade dos processos por ela adotados.

1.2 ESTRUTURA DO TRABALHO

O trabalho foi construído e estruturado em quatro capítulos. O segundo capítulo abrange a fundamentação teórica necessária para fornecer um melhor embasamento e entendimento do trabalho desenvolvido e do tema abordado.

O capítulo três apresenta o desenvolvimento do protótipo, informando a especificação e os principais requisitos a serem cumpridos no trabalho. Para isso, diagramas de casos de uso, classes e atividades e o modelo de entidade relacionamento foram criados. Ainda nesse capítulo são apresentadas as tecnologias utilizadas na implementação, a descrição da operacionalidade do protótipo desenvolvido e, por fim, os resultados alcançados.

O quarto e último capítulo trata das conclusões obtidas após o desenvolvimento do trabalho e as sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 apresenta a área de avaliação de software, detalhando alguns conceitos, propriedades e métricas utilizadas, enfatizando as áreas abordadas no CEP. Na seção 2.2 é dada uma explanação sobre os modelos de maturidade CMMI e MR-MPS. Na seção seguinte, 2.3, são detalhadas algumas informações sobre a ferramenta SonarQube. Por final, na seção 2.4, são apresentados os trabalhos correlatos sobre o tema abordado.

2.1 MEDIÇÃO DE SOFTWARE

Medição de Software é uma avaliação quantitativa dos objetos e eventos da Engenharia de Software. O elemento básico da medição, são as medidas. Elas caracterizam, em termos quantitativos, os atributos dos produtos, processos e projetos de software. Como exemplo, podem-se utilizar medidas para analisar a produtividade de uma equipe, o esforço gasto numa tarefa e a confiabilidade do sistema desenvolvido (GRESSE VON WANGENHEIM; VON WANGENHEIM; LINO, 2012, p. 15).

Trata-se de uma prática fundamental para o gerenciamento de projetos e para a melhoria de processos e está presente nos principais modelos e padrões de melhoria de processos de software, como o CMMI e o MR-MPS. Nesses modelos, que tratam a melhoria de processo em níveis, a medição é introduzida nos níveis iniciais (nível 2 do CMMI e nível F do MR-MPS) e evolui à medida que a maturidade dos processos aumenta. Na alta maturidade (níveis 4 e 5 do CMMI e B e A do MR-MPS) a medição evolui a ponto de ser necessário a introdução do Controle Estatístico de Processos (MARETTO, 2013, p.6).

Barcellos (2014, p.8) define como principais conceitos básicos da medição de software, os seguintes:

- a) medida: trata-se de contar os atributos das entidades. Pode ser tratada como medida base ou medida derivada;
- b) elemento: propriedade de uma entidade que pode ser medida e transformada em algo quantificável;
- c) entidade: objeto que pode ser classificado e caracterizado pela quantificação dos seus elementos;
- d) unidade de medida: maneira pela qual uma medida pode ser apresentada;
- e) escala: formaliza os valores que uma medida pode assumir;
- f) procedimento de medição: descrição de como a coleta das medidas deve ser realizada;
- g) procedimento de análise de medição: descrição de como os dados coletados devem

- ser representados e analisados;
- h) definição operacional de medida: informações sobre os procedimentos de análise e medição, os responsáveis pela medição e pela análise da medida e os momentos e a periodicidade em que a medição e análise devem ser realizadas;
 - i) medição: resultado da execução do procedimento de medição sobre os dados coletados;
 - j) análise de medição: resultado da execução do procedimento de análise sobre os dados coletados;
 - k) indicador: tipo de medida utilizada para visualizar os objetivos e metas estipulados.

2.1.1 Medição e Melhoria de Processos de Software

Os processos devem ser capazes de atender às necessidades de uma empresa e devem ser objeto de melhorias contínuas. Evidenciar problemas nos processos e apoiar melhorias é fator de extrema importância nesse contexto.

Medições são essenciais para a realização de melhorias em processos de software porque fornecem dados objetivos que permitem conhecer o seu desempenho. Os dados coletados para as medidas são a base para a detecção de problemas no desempenho e de inadequações nos processos, bem como para a identificação de oportunidades de melhoria e tomada de decisão. A avaliação do alcance dos objetivos de melhoria nos processos, também, depende de medições. Cada objetivo definido deve ser associado a indicadores e medidas capazes de apoiar a organização na avaliação da adequação e eficácia das ações de melhoria implementadas e do alcance dos objetivos (BARCELLOS; ROCHA; SOUZA, 2012, p.21).

Identificam-se três objetivos para a medição de processos de software, são eles (FLORAC; CARLETON, 1999, p.37):

- a) coletar dados para medir o desempenho do processo;
- b) analisar o desempenho do processo;
- c) armazenar e utilizar os dados para interpretar os resultados de observações e análises, prever custos e desempenho futuros, fornecer *baselines* e *benchmarks*, identificar tendências e avaliar a estabilidade e capacidade do processo.

Para a maioria das organizações, responder a essas questões de forma objetiva e precisa não é uma tarefa trivial, pois nem sempre os processos da organização são corretamente gerenciados e, com isso, seu desempenho real não é conhecido (FLORAC; CARLETON, 1999, p.69).

2.1.2 Controle Estatístico de Processos

O CEP ou Controle Estatístico de Processos não é novidade para a indústria em geral. Foi inicialmente proposto para a melhoria de processos na área de manufatura. Pode ser definido com uma técnica estatística aplicada à produção que permite a redução da variabilidade nas características significativas, contribuindo para a melhoria da qualidade, da produtividade, da confiabilidade e do custo do que está sendo produzido. Seu objetivo principal está em verificar a presença de causas especiais, ou seja, causas que não são naturais ao processo e que podem prejudicar a qualidade do produto (RIBEIRO; TEN CATEN, 2012, p.5).

Dois fatores relevantes sobre o comportamento de processos são:

- a) estabilidade: o processo possui variações dentro de limites esperados e é previsível;
- b) capacidade: o processo é estável e atende aos objetivos estabelecidos.

Embora utilizado em conjunto com a chamada estatística clássica, existem grandes diferenças entre ela e o Controle Estatístico de Processos. De acordo com Barcellos, Rocha e Souza (2012, p. 60) a principal diferença entre eles é que a estatística clássica “[...] utiliza métodos de análise baseados em dados estatísticos no tempo. Com isso, na maioria das vezes, independente da ordem em que os dados forem analisados, o resultado da análise será o mesmo”.

Outra diferença importante, agora entre a medição tradicional e o Controle Estatístico de Processos, está no acompanhamento dos valores coletados para as medidas. Enquanto que na medição tradicional o acompanhamento é periódico (mensal, quinzenal, etc.) no Controle Estatístico de Processos o acompanhamento é bem mais frequente.

No Controle Estatístico de Processos, as medidas devem ser analisadas frequentemente. [...] Assim, caso a aderência aos prazos não fosse satisfatória, o gerente poderia tomar ações para melhorá-la ainda no contexto do projeto corrente (BARCELLOS; ROCHA; SOUZA, 2012, p.61).

2.2 MODELOS DE MATURIDADE CMMI E MR-MPS

O CMMI é um modelo de referência desenvolvido pela SEI¹ da Universidade Carnegie Mellon. Foi baseado no Modelo de Maturidade em Capacitação (CMM) com algumas ideias mais importantes da qualidade na indústria. Nesse processo, a organização pode optar por

¹ *Software Engineering Intitute* (SEI). É um centro de pesquisa e desenvolvimento patrocinado pelo Departamento de Defesa dos Estados Unidos da América que provê uma prática avançada de engenharia de software qualificando graus de qualidade de software.

duas representações: por estágios (apresenta um caminho pré-definido por meio de implementação sequencial) ou contínua.

Este modelo de qualidade é reconhecido internacionalmente e se tornou uma referência no mercado. Empresas como a Microsoft já adotam o modelo como estratégia para exportação da mão-de-obra brasileira, buscando obter um diferencial competitivo. O conjunto de práticas do CMMI contribui para o aprimoramento dos processos de uma organização tornando-a mais madura e eficiente. O CMMI ajuda a organização a conhecer os seus processos e o seu desempenho, melhorando a precisão do planejamento. Permite um melhor monitoramento dos processos, possibilitando que o gerente de projetos saiba se o projeto dará certo ou não. Com o tempo, adquirindo maturidade, a empresa vai identificando o que realmente tem valor, sendo este o foco, otimizando cada vez mais os processos, o que justifica o CMMI, que é um modelo que possibilita uma melhoria contínua nos processos, amadurecendo as organizações e tornando-as mais competitivas (FRANCISCANI; PESTILI, 2012, p.5).

Já o MR-MPS ou Melhoria de Processo do Software Brasileiro (MPS-BR) foi criado pelo Softex e patrocinado pelo Ministério de Ciência e Tecnologia (MCT). Consiste em uma adaptação do modelo CMMI com níveis de amadurecimento com saltos mais gradativos (7 níveis, de A a G).

Este outro modelo foi criado com o objetivo de ser um modelo de processo em que as empresas conseguem atingir os níveis de maturidade mais rápidos, sendo mais adequado à realidade brasileira, além de ser mais acessível do que o modelo de projeto CMMI. Além dessas vantagens, pode-se citar outras: maior número de níveis, compatibilidade com CMMI, avaliação periódica, integração universidade-empresa e aceite em licitações (FRANCISCANI; PESTILI, 2012, p.7).

Esses modelos de maturidade são cada vez mais utilizados por organizações de software que têm por objetivo investir em qualidade de software. Eles apresentam um conjunto de boas práticas de Engenharia de Software organizadas em diferentes níveis de maturidade. Níveis de maturidade caracterizam estágios de melhoria da implementação de processos na organização e permite prever o seu desempenho futuro ao executar um ou mais processos (SEI, 2010 apud BARCELLOS; ROCHA; SOUZA, 2012, p.154).

Vale ressaltar as diferenças nas formas de medição nos diferentes níveis de maturidade. Nos níveis iniciais é realizada a medição tradicional cujo propósito é apoiar a monitoração e controle tradicionais dos projetos sob um acompanhamento mais periódico, menos frequente. Em contrapartida, na alta maturidade, a medição busca apoiar a análise de desempenho dos processos sob um acompanhamento mais frequente e mais rigoroso. Com a

aplicação destes modelos acredita-se que as organizações possam produzir softwares melhores e menos custosos, estimativas mais precisas e, conseqüentemente, clientes mais satisfeitos. Desde os níveis F do MR-MPS e 2 do CMMI a área de medição já está presente.

2.3 FERRAMENTA SONARQUBE

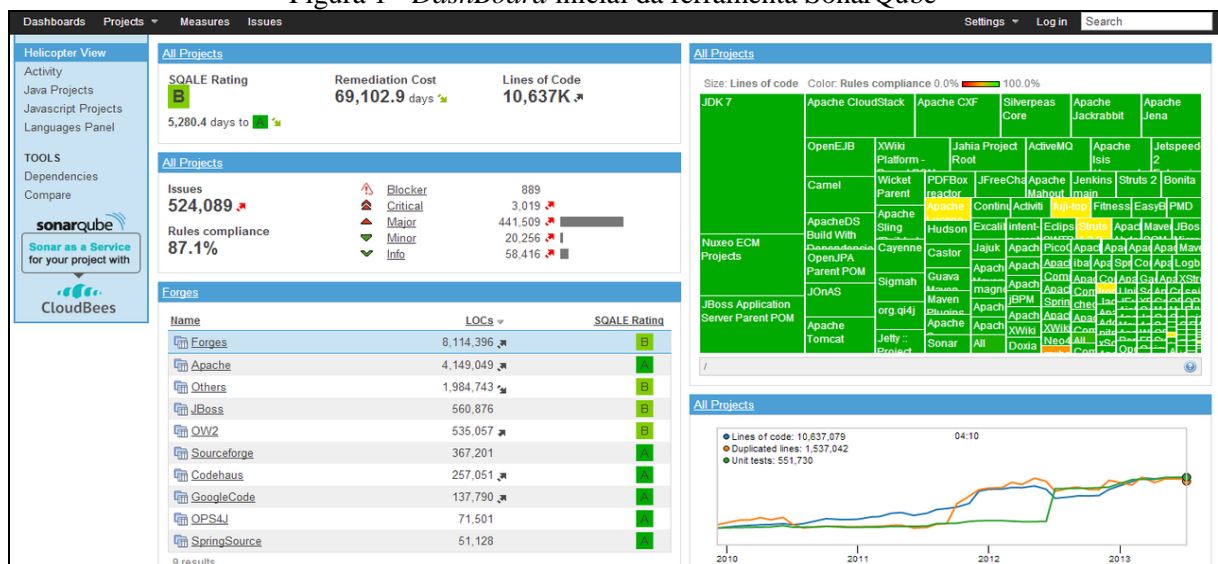
Neste item é apresentada a ferramenta utilizada para integração com o protótipo proposto neste trabalho. O seu uso tem por objetivo reduzir o tempo gasto no registro das informações sobre os projetos a serem analisados e, também, reaproveitar os dados já levantados (principalmente em projetos mais antigos).

2.3.1 SonarQube

Trata-se de um software *open-source* que se propõe a ser um repositório de qualidade do código-fonte, possibilitando o controle sobre um grande número de métricas de software e o apontamento de uma série de possíveis *bugs*. Os resultados obtidos são mostrados através de uma interface web em forma de *dashboards* e gráficos, conforme pode ser visto na Figura 1.

A forma de integração com esta ferramenta se dá através da utilização de *WebServices*. Estes serviços, por sua vez, permitem recuperar várias informações sobre o projeto, são elas: lista de versões, lista de usuários e suas propriedades, medidas e métricas, propriedades e, até mesmo, trechos de código fonte que compõem o projeto.

Figura 1 - *DashBoard* inicial da ferramenta SonarQube



Fonte: SonarQube Group (2014).

2.4 TRABALHOS CORRELATOS

A seguir são apresentados três trabalhos correlatos desenvolvidos no meio acadêmico: um modelo para Controle Estatístico de Processos de desenvolvimento de software proposto

por Fonseca (2010), um programa de medição para organizações de alta maturidade proposto por Batista (2005) e, por fim, um método evolutivo para aplicação de programas de métricas em processos de desenvolvimento de software proposto por Bogoni (2007). Os trabalhos serão detalhados a seguir.

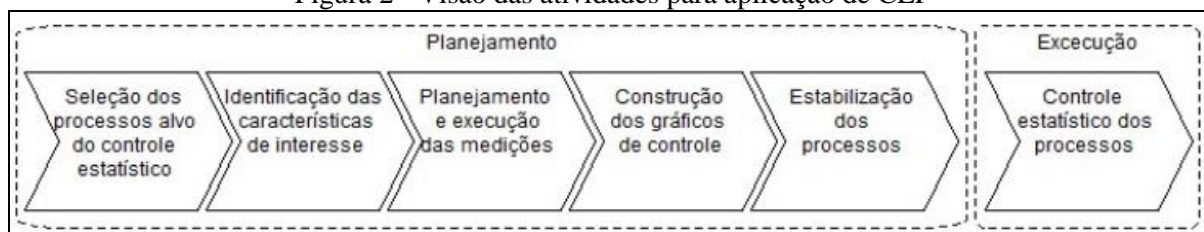
2.4.1 Modelo para Controle Estatístico de Processos de Desenvolvimento de Software (CEP-S).

Fonseca (2010) elaborou um modelo para pequenas e médias empresas aplicarem o Controle Estatístico de Processos (CEP) na prática de desenvolvimento de software (CEP-S). O seu objetivo principal é permitir diagnosticar se o processo está sob influência de causas atribuíveis que precisam ser investigadas e eliminadas.

O modelo CEP-S propõe um conjunto de características de qualidade a serem monitoradas para determinados processos, os gráficos de controle mais adequados para cada processo e sugestões para a escolha dos parâmetros de controle. Esses elementos são organizados em um método proposto no modelo, fornecendo um arcabouço com escolhas pré-definidas, porém flexíveis, que contribui para tornar mais simples a aplicação de CEP para pequenas e médias empresas, favorecendo as investidas iniciais nas atividades de controle estatístico.

As atividades de preparação para a execução do CEP estão agrupadas na etapa de planejamento, como apresentada na Figura 2. O processo estável permite o cálculo dos parâmetros de controle que conduzirão o controle estatístico. No momento em que os processos se tornam estáveis é possível iniciar a execução do CEP, que consiste nas seguintes atividades: coleta e plotagem dos dados, testes de controle, investigação de causas atribuíveis, aplicação de melhorias e exclusão de dados afetados por causas atribuíveis.

Figura 2 - Visão das atividades para aplicação de CEP



Fonte: Fonseca (2010).

2.4.2 Programa de Medição para Organizações de Alta Maturidade

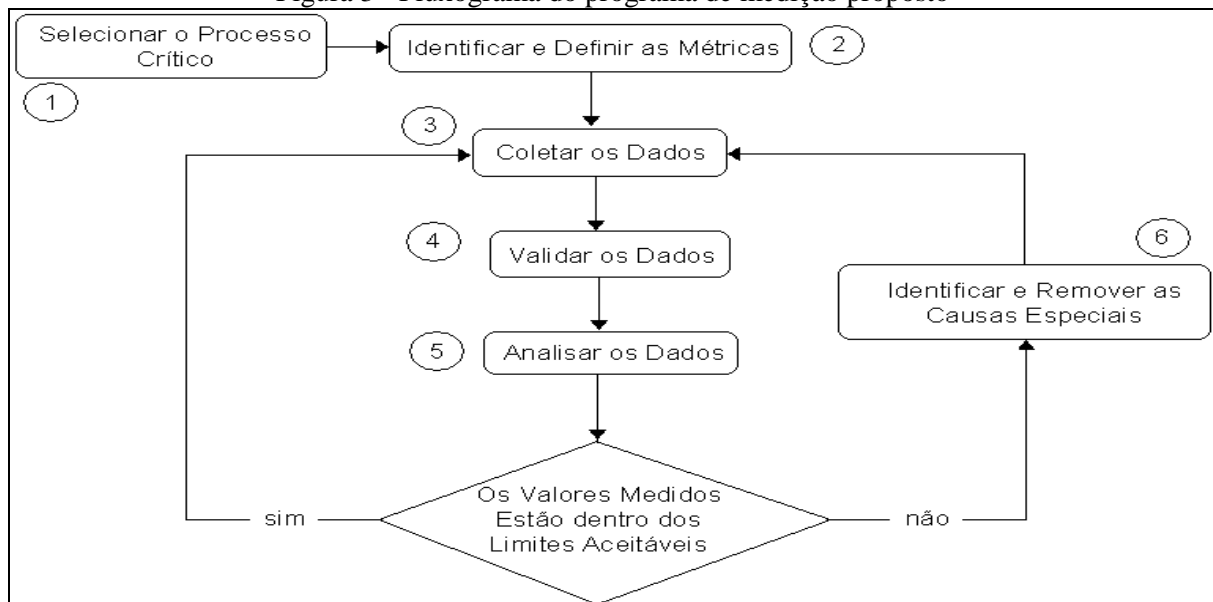
Batista (2005) apresentou o desenvolvimento de um programa de medição voltado para organizações de alto nível de maturidade, visando facilitar e auxiliar o processo de melhoria

contínua de seus processos. Uma vez que a avaliação quantitativa da capacidade do processo de software e suas variações permite planejar e gerenciar melhor os projetos, a proposta de programa de medição torna-se muito útil.

O programa de medição apresenta métricas alinhadas às metas organizacionais e exige que, após a coleta dos dados e sua análise, os envolvidos nessas métricas (um gerente administrativo, um gerente funcional, um líder de projeto ou um desenvolvedor) comprometam-se a usar os resultados da análise para identificar os desvios de processo e aplicar as ações corretivas necessárias. Desta forma, pode-se controlar o desempenho do processo de desenvolvimento de software dentro dos limites aceitáveis (BATISTA, 2005, p.3).

Para apoiar o processo de implantação e aplicação de métricas, uma ferramenta de coleta, validação e análise dos dados, baseada em controle estatístico de processo, denominada Vigia, foi desenvolvida. Esta ferramenta, por sua vez, poderia ser utilizada para controlar o desempenho do processo de software definido para o projeto, assegurando que o processo não comprometa as metas de qualidade da organização nem as metas de negócio, por meio de ações corretivas em tempo real e, conseqüentemente, de ajustes no processo de software. A Figura 3 descreve o processo adotado no programa.

Figura 3 - Fluxograma do programa de medição proposto



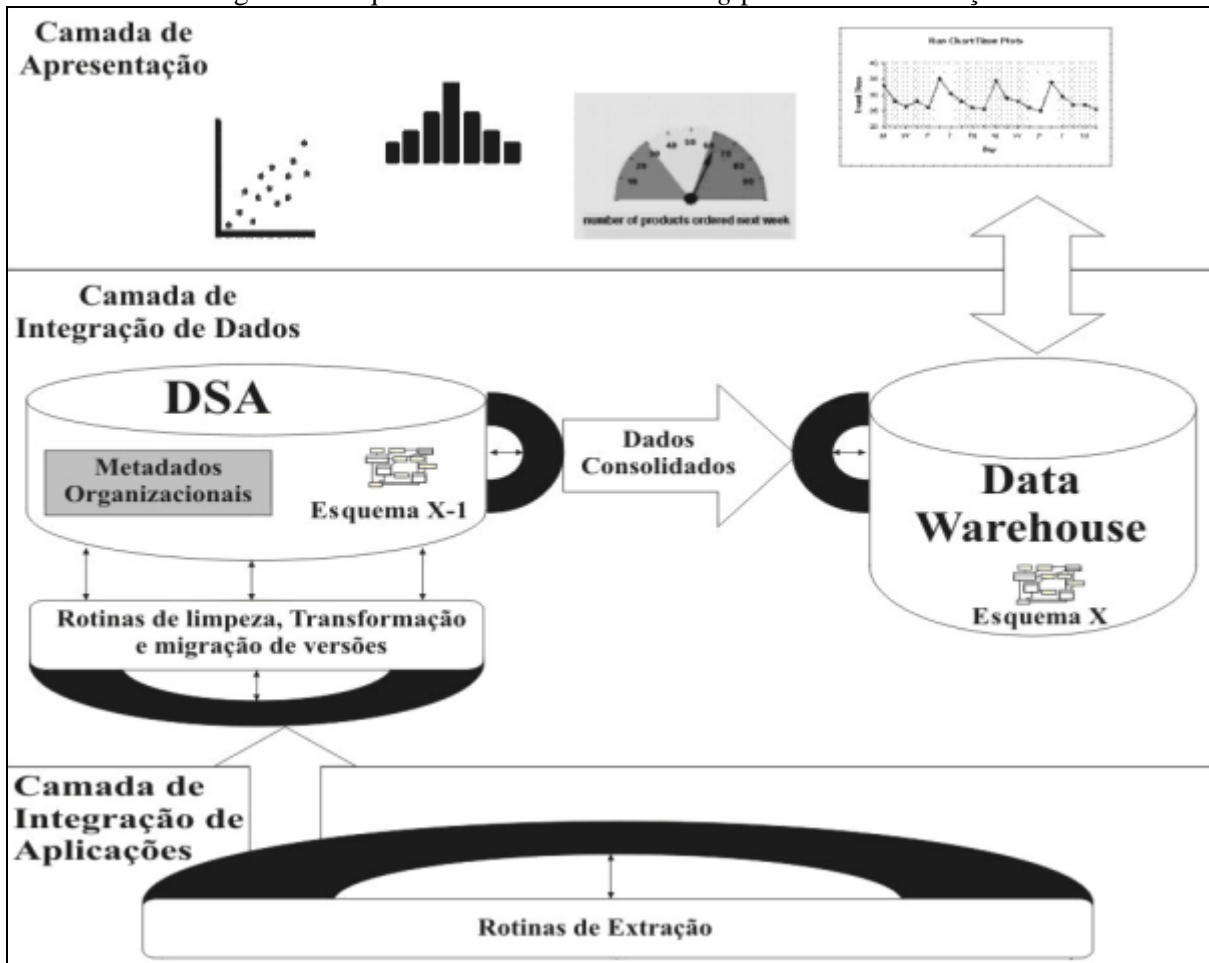
Fonte: Batista (2005).

2.4.3 Um Método Evolutivo para Aplicação de Programas de Métricas em Processos de Desenvolvimento de Software

Bogoni (2007) apresenta um método para extração, organização e apresentação de métricas para Processo de Desenvolvimento de Software (PDS), levando em consideração a

evolução do próprio PDS e do conjunto de métricas correspondente. A solução é baseada em um ambiente de *Data Warehousing* (conforme pode ser observado na Figura 4) e tem a finalidade de resgatar medições feitas em projetos passados, sob diferentes modelos de PDS e programas de métricas, e formar uma base sólida de informações desses projetos. Para viabilizar o resgate destas medições, são propostos procedimentos para tratar adequadamente a criação, alteração e exclusão de métricas.

Figura 4 - Arquitetura de *Data Warehousing* para PDS em evolução



Fonte: Bogoni (2007).

A principal contribuição do trabalho proposto é a de permitir que medições presentes e passadas possam ser mantidas em um repositório único de métricas da organização e que as mesmas sejam comparáveis, viabilizando um melhor controle dos projetos de software e qualidade de seus produtos. A arquitetura proposta, composta de três camadas é orientada a serviços, utilizando a facilidade desta abordagem para o acesso aos dados de projetos nas ferramentas de apoio. Quando ocorre uma evolução do PDS, e, conseqüentemente do conjunto de métricas, um conjunto de metadados, os quais servem de configuração para cada métrica, devem ser alterados para dar suporte à coleta das novas métricas, ou a alteração das métricas existentes.

Através de uma experimentação da solução em uma empresa desenvolvedora de software com certificação CMMI nível 2, Bogoni (2007) demonstrou que é possível manter um repositório organizacional de métricas utilizando-se dados de projetos, mesmo estes estando em diferentes versões de PDS. Conforme o PDS vai se desenvolvendo, o *Data Warehouse* é novamente reorganizado. Da mesma forma, as métricas que não sofreram modificações são migradas diretamente para o novo modelo, enquanto que as métricas que foram criadas ou modificadas devem ser tratadas e organizadas corretamente no novo modelo de dados.

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são apresentadas as etapas do desenvolvimento do protótipo de ferramenta web para suporte a avaliação de software de acordo com o CEP. Na seção 3.1 são enumerados os principais requisitos da ferramenta a ser desenvolvida. A seção 3.2 apresenta sua especificação. A seção 3.3 mostra os detalhes da implementação e, por fim, a seção 3.4 exhibe os resultados obtidos no protótipo.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Baseando-se nos trabalhos correlatos pesquisados e na descrição dos modelos do CEP, foram identificados os seguintes requisitos:

- a) permitir o cadastro de softwares (Requisito Funcional - RF);
- b) permitir o cadastro de itens (RF);
- c) permitir o cadastro de coletas (RF);
- d) permitir o cadastro de fases do processo de desenvolvimento de software (RF);
- e) permitir o cadastro de usuários (RF);
- f) realizar os cálculos estatísticos de avaliação, sempre que requisitado (RF);
- g) apresentar uma interface com gráficos informativos sobre os cálculos realizados (RF);
- h) classificar os itens cadastrados de acordo com a estabilidade e capacidade do processo (RF);
- i) permitir a geração de planilhas com dados sobre os itens e os softwares cadastrados (RF);
- j) utilizar a linguagem de programação Java 1.7 para a implementação (Requisito Não-Funcional - RNF);
- k) utilizar o ambiente de desenvolvimento Eclipse 4.4 (RNF);
- l) utilizar JSF 2.0 com o *framework* PrimeFaces para que a ferramenta ofereça suporte web (RNF);
- m) utilizar o Sistema Gerenciador de Banco de Dados MySQL (RNF);
- n) utilizar o *framework* Demoiselle para o desenvolvimento (RNF);
- o) realizar a integração com o SonarQube via banco de dados (RNF).

3.2 ESPECIFICAÇÃO

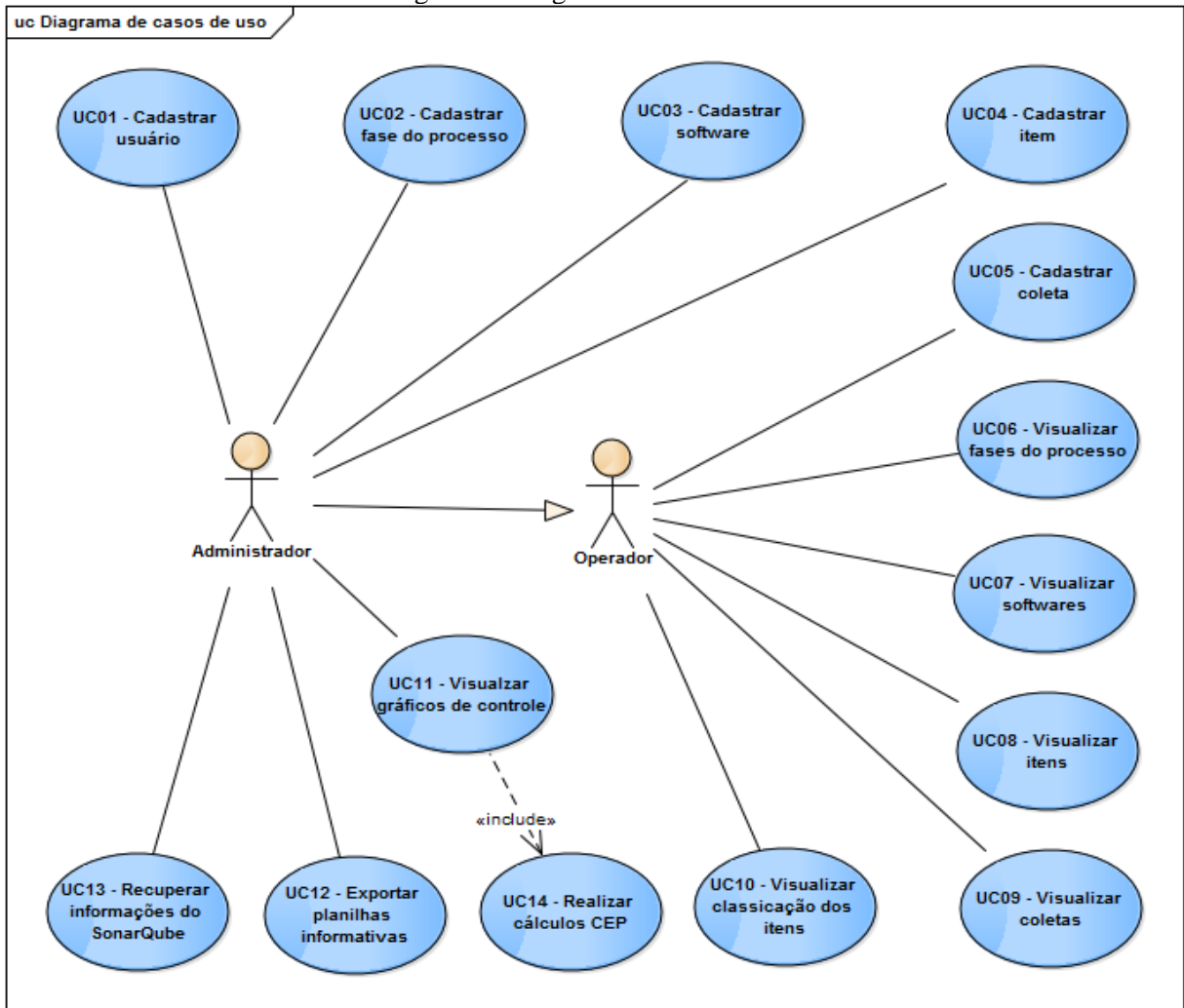
A seguir é apresentada a especificação do protótipo, representado por diagramas da *Unified Modeling Language* (UML) e modelada utilizando-se a ferramenta Enterprise

Architect 11. Os diagramas utilizados foram o de casos de uso, classes e atividades. Também foi criada a modelagem Modelo Entidade Relacionamento (MER).

3.2.1 Diagrama de casos de uso

Nesta seção são descritos os casos de uso com as funcionalidades do protótipo. Foram identificados dois atores: o administrador e o operador. De um modo geral, o administrador é responsável por acompanhar o processo de coleta das informações através do acesso aos gráficos de controle e às planilhas informativas, além de ter o acesso ao cadastro e a alteração de todas as informações do sistema. O operador será o usuário responsável por cadastrar e vincular as coletas. Na Figura 5 é apresentado o diagrama de casos de uso e o detalhamento dos três principais casos de uso encontra-se no Apêndice A.

Figura 5 - Diagrama de caso de uso



A seguir são descritos os casos de uso exibidos na Figura 5:

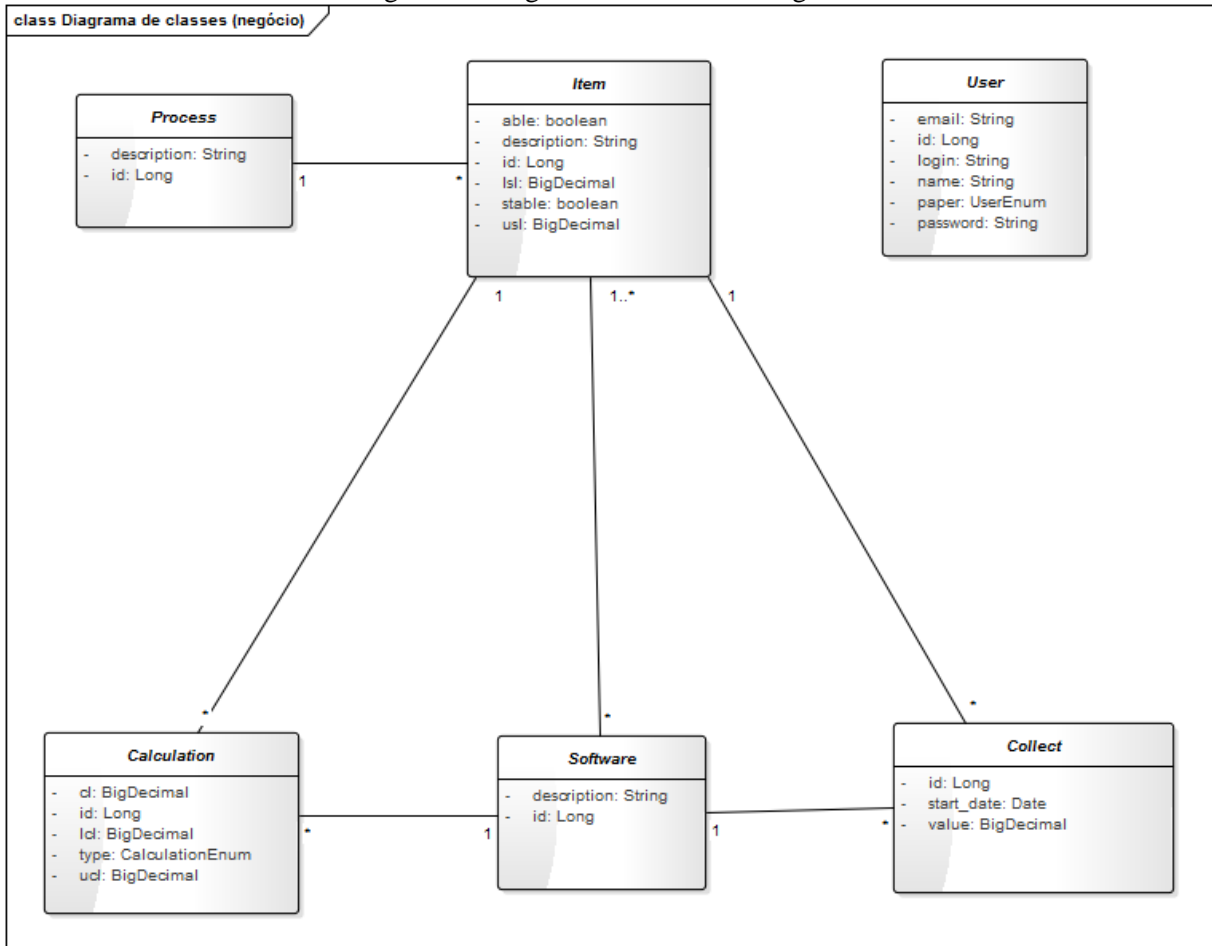
- UC01 - Cadastrar usuário: permite cadastrar os usuários que vão ter acesso ao protótipo;

- b) UC02 - Cadastrar fase do processo: permite cadastrar as fases do processo de desenvolvimento de software;
- c) UC03 - Cadastrar software: permite cadastrar os softwares ou projetos a serem avaliados com base no CEP;
- d) UC04 - Cadastrar item: permite cadastrar os itens (ou seja, as métricas) a serem avaliadas com base no CEP e vinculá-los aos softwares e as fases dos processos cadastrados;
- e) UC05 - Cadastrar coleta: permite cadastrar o valor de uma coleta e vinculá-la a um item e um software;
- f) UC06 - Visualizar fases do processo: permite visualizar as fases dos processos já cadastradas;
- g) UC07 - Visualizar softwares: permite visualizar os softwares ou projetos já cadastrados;
- h) UC08 - Visualizar itens/métricas: permite visualizar os itens ou métricas já cadastrados;
- i) UC09 - Visualizar coletas: permite visualizar as coletas já cadastradas durante a fase de análise;
- j) UC10 - Visualizar classificação dos itens: permite ao administrador visualizar a classificação dos itens obtida após a geração dos gráficos de controle e, conseqüentemente, dos cálculos estatísticos;
- k) UC11 - Visualizar gráficos de controle: permite ao administrador visualizar os gráficos de controle definidos no CEP;
- l) UC12 - Exportar planilhas informativas: permite ao administrador exportar as planilhas com a listagem dos softwares, itens e da classificação dos itens;
- m) UC13 - Recuperar informações do SonarQube: permite ao administrador realizar a integração do protótipo com o SonarQube, recuperando as informações úteis para a geração dos gráficos de controle do CEP;
- n) UC14 - Realizar cálculos CEP: permite ao administrador aplicar os cálculos estatísticos do CEP sobre os valores coletados a partir da geração dos gráficos de controle disponibilizados.

3.2.2 Diagrama de classes

Na Figura 6, através de um diagrama de classes, são apresentadas as classes pertencentes à camada de negócio do protótipo. Esta camada tem como responsabilidade transcrever o modelo de dados persistido no banco de dados.

Figura 6 - Diagrama de classes de negócio



As classes exibidas na Figura 6 são descritas a seguir:

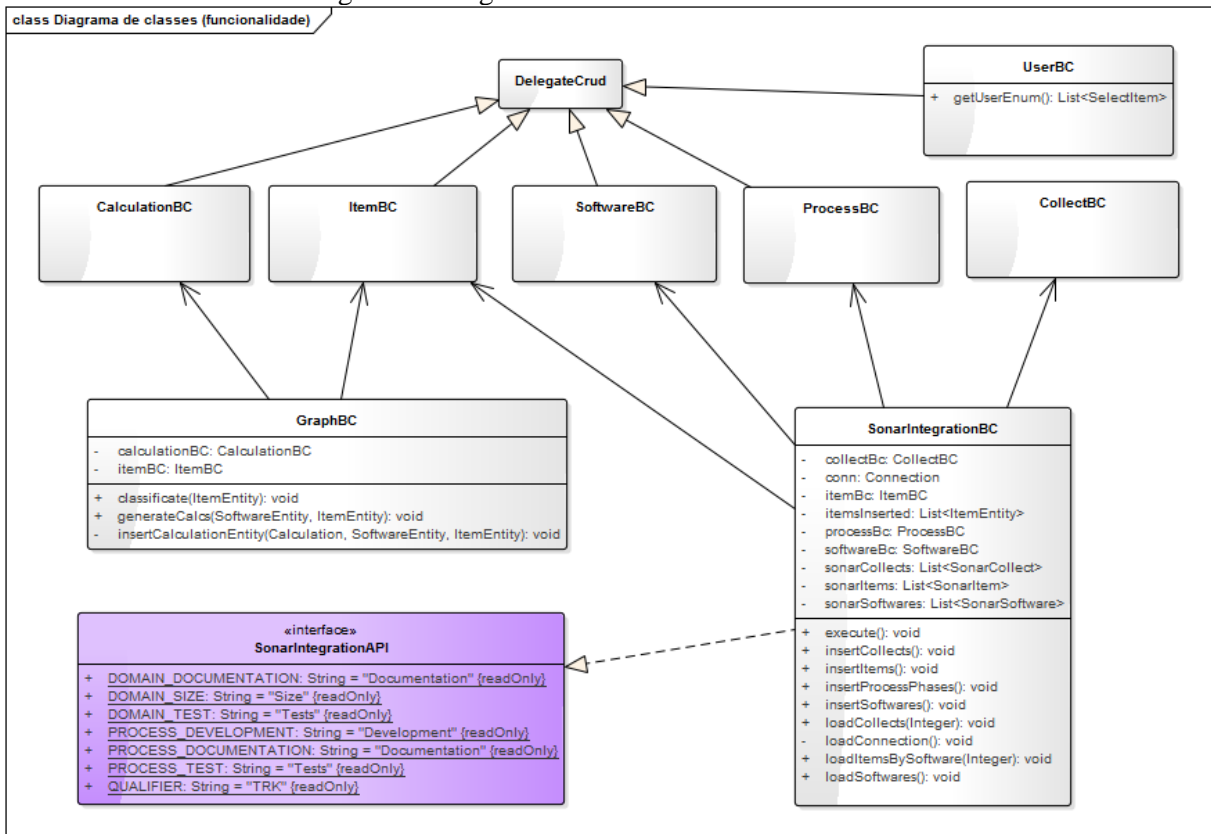
- Process**: armazena as informações sobre os processos de desenvolvimento de software que no qual se encontram os itens a serem avaliados;
- Item**: armazena as informações das métricas a serem avaliadas com base nos cálculos definidos no CEP;
- Software**: armazena as informações dos softwares a serem avaliados;
- Collect**: armazena as informações referentes aos valores das coletas realizadas ao longo do processo de análise;
- Calculation**: armazena os resultados dos cálculos realizados com base nos gráficos de controle estatísticos;

- f) `User`: armazena as informações referentes aos usuários que possuem acesso ao protótipo;

Na Figura 7 é exibido um novo diagrama de classes, porém indicando agora as classes que são responsáveis por realizar as funções disponibilizadas pelo protótipo, ou seja, que fazem o intercâmbio entre as requisições do usuário com algum recurso do sistema. As classes são:

- a) `DelegateCrud`: define uma implementação básica fornecida pelo *framework* `Demoiselle` para as quatro operações básicas de manipulação de dados em um banco de dados: *create*, *read*, *update* e *delete*;
- b) `UserBC`: define as funcionalidades que podem ser realizadas sobre os usuários do sistema;
- c) `CalculationBC`: define as funcionalidades que podem ser realizadas sobre os cálculos estatísticos no protótipo. Os cálculos são gerados e inseridos automaticamente após a geração dos gráficos de controle;
- d) `ItemBC`: define as funcionalidades que podem ser realizadas sobre os itens ou as métricas no protótipo;
- e) `SoftwareBC`: define as funcionalidades que podem ser realizadas sobre os softwares ou projetos no protótipo;
- f) `ProcessBC`: define as funcionalidades que podem ser realizadas sobre as fases dos processos no protótipo;
- g) `CollectBC`: define as funcionalidades que podem ser realizadas sobre os valores das coletas obtidas até o momento no protótipo;
- h) `GraphBC`: define as funcionalidades que podem ser realizadas na visualização dos gráficos de controle. Antes da geração destes gráficos, são realizados novos cálculos sobre o item e o software a ser selecionado pelo usuário;
- i) `SonarIntegrationBC`: define a implementação sobre a integração com a ferramenta `SonarQube`. Neste processo, são importados pelo protótipo as fases dos processos, as métricas (ou seja, os itens) e as coletas;
- j) `SonarIntegrationAPI`: define a interface de implementação para a integração com a ferramenta `SonarQube`.

Figura 7 - Diagrama de classes de funcionalidades

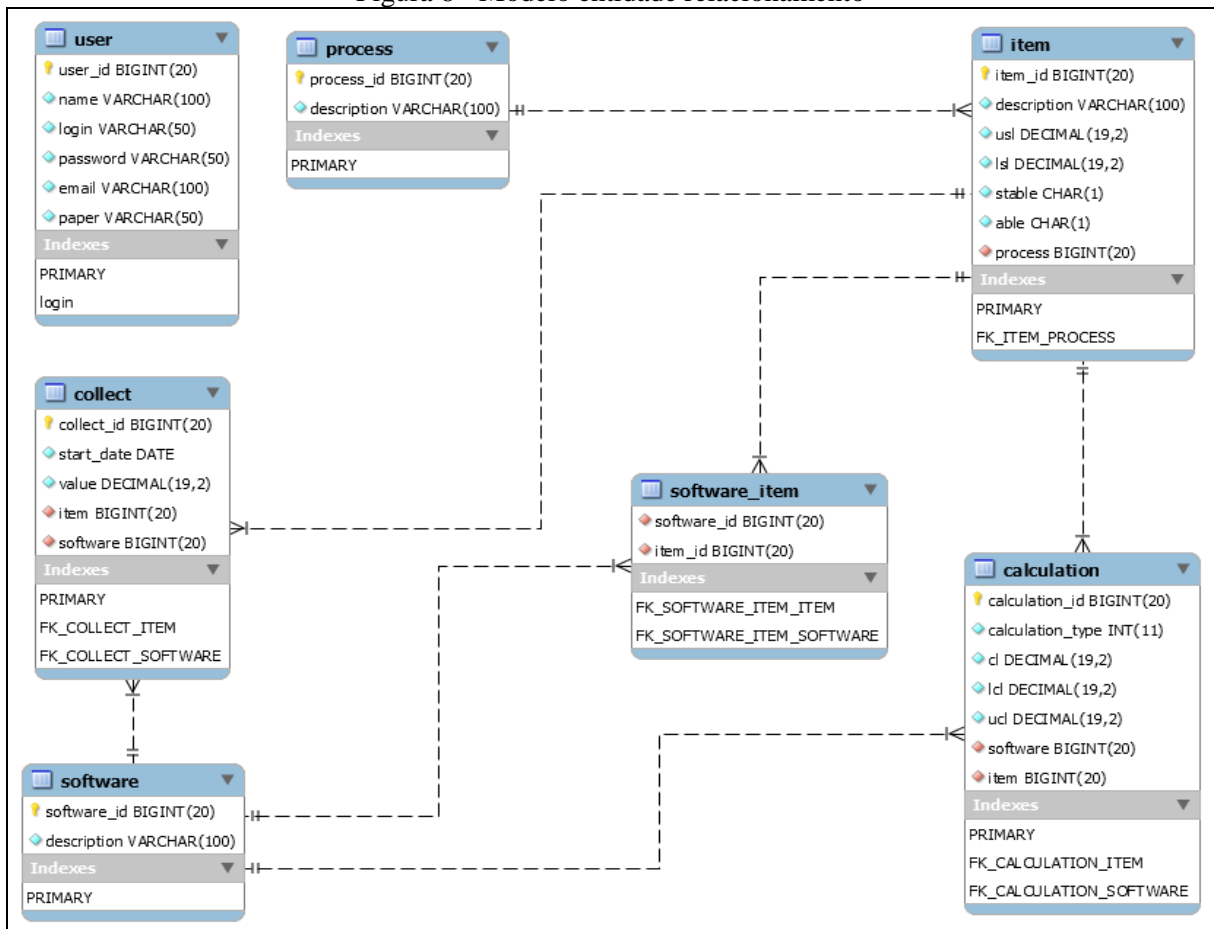


Na Figura 7, pode ser verificado que as classes **GraphBC** e **SonarIntegrationBC** possuem várias dependências de outras classes para que se sejam capazes de realizar a recuperação e validação de um grande número de informações. Também pode ser identificado que as classes que estendem a classe **DelegateCrud** podem fazer uso das operações de CRUD já implementadas pelo **Demoiselle** sem a necessidade de uma implementação própria.

3.2.3 Diagrama de entidade relacionamento

Na Figura 8 é mostrado o modelo da base de dados do protótipo utilizando-se como base a camada de entidades. No diagrama foram especificadas as tabelas, colunas, chaves primárias e estrangeiras. No Apêndice B encontra-se o dicionário de dados.

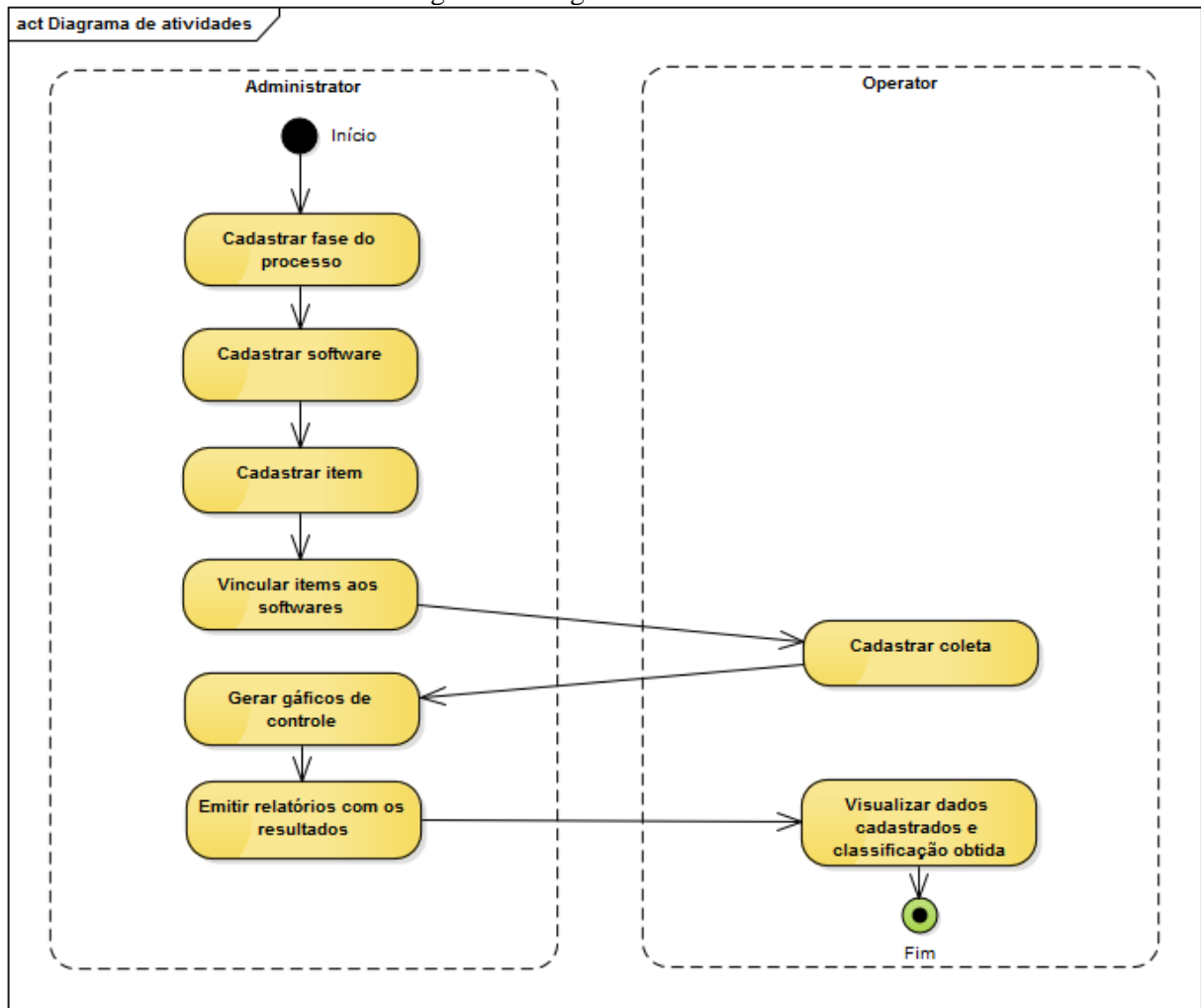
Figura 8 - Modelo entidade relacionamento



3.2.4 Diagrama de atividades

O fluxo padrão para a avaliação das métricas e dos softwares cadastrados no sistema pode ser observado na Figura 9. O usuário administrador é, de uma forma geral, responsável por cadastrar todas as informações que devem ser vinculadas a uma coleta. Esta coleta, por sua vez, é mais comumente realizada pelo operador do sistema, mas também pode ser realizada pelo próprio administrador.

Figura 9 - Diagrama de atividades



O processo se inicia com o administrador cadastrando uma fase do processo, os softwares e os itens a serem avaliados. Em seguida são vinculados os itens que devem ser analisados para cada software cadastrado.

Posteriormente, o operador deve cadastrar as coletas. Esse processo pode ficar em aberto por tempo indefinido, já que o software pode ainda não estar finalizado – em testes ou em desenvolvimento, por exemplo.

A partir disso, o usuário administrador pode visualizar os gráficos de controle dos itens ou métricas de cada software a qualquer momento, além de verificar sua classificação quanto a sua capacidade e estabilidade. Por fim, o operador pode também visualizar as classificações obtidas e as informações cadastradas.

3.3 IMPLEMENTAÇÃO

A seguir estão detalhadas e elencadas as técnicas e ferramentas utilizadas na implementação.

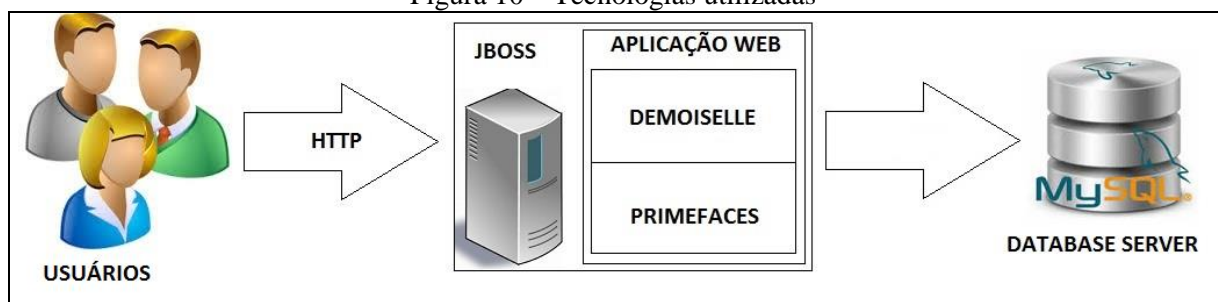
3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do protótipo foi utilizado o ambiente de desenvolvimento Eclipse 4.4.1 e como linguagem de programação a linguagem JAVA na versão 7. As tecnologias utilizadas na implementação do protótipo são:

- a) JBoss AS 7.1.1.Final: o JBoss Application Server é um servidor de aplicações de código aberto desenvolvido pela JBoss para a plataforma Java Enterprise Edition (Java EE). Ele provê um ambiente completo para que outras aplicações sejam executadas dentro dele usando uma gama de serviços;
- b) *framework* Demoiselle: implementa o conceito de *framework* integrador. Seu objetivo é facilitar a construção de aplicações minimizando tempo dedicado à escolha e integração de *frameworks* especialistas, o que resulta no aumento da produtividade e garante a manutenibilidade dos sistemas. Disponibiliza mecanismos reusáveis voltados às funcionalidades mais comuns de uma aplicação (arquitetura, segurança, transação, mensagem, configuração, tratamento de exceções, entre outros);
- c) PrimeFaces: é um *framework* que disponibiliza um conjunto de componentes de código aberto para o Java Server Faces (JSF) a partir da versão 2.0, permitindo criar interfaces ricas para aplicações web de forma simplificada e eficiente. Sua facilidade de configuração, boa documentação e grande variedade de componentes o tornam muito atrativo para os projetos de software web;
- d) MySQL: é um Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR), desenvolvido como projeto de código aberto. É conhecido por sua facilidade de uso e por sua ampla utilização. Sua interface simples, e também sua capacidade de rodar em vários sistemas operacionais, são alguns dos motivos para este programa ser tão usado atualmente, e seu uso estar crescendo cada vez mais.

Na Figura 10 é ilustrado de forma simplificada, na forma de um fluxograma, as tecnologias citadas acima.

Figura 10 – Tecnologias utilizadas



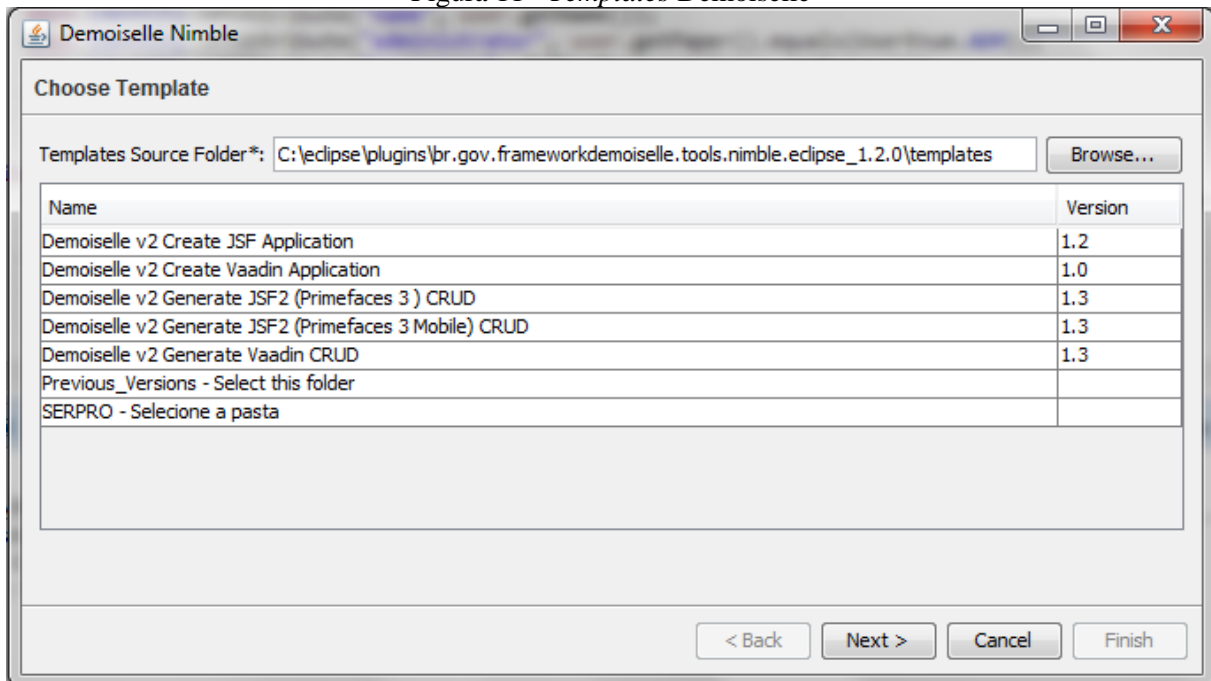
3.3.2 Implementação do protótipo

Para a implementação do protótipo a língua inglesa foi adotada como padrão, devido familiaridade tida por ela e pela presença em larga escala no âmbito tecnológico. Foi utilizada a especificação JSF (como já citado anteriormente) com a implementação de referência Oracle Mojarra. Uma característica importante na arquitetura JSF é a separação entre as camadas de apresentação que, por padrão, baseia-se nos *Facelets* e um conjunto de classes conhecidas como *Managed Beans* e a camada de aplicação.

Em conjunto com o JSF, foi utilizada a biblioteca de componentes Primefaces. Esta biblioteca provê uma versão melhorada de praticamente todos os componentes JSF, além de possuir uma ampla gama de recursos diferenciados, como um gerador de gráficos estatísticos e vários temas pré-definidos, por exemplo. Uma grande vantagem da sua utilização é a sua facilidade de configuração, sendo necessária apenas a adição de um único arquivo com a extensão `jar` no projeto web.

Para a construção da arquitetura do protótipo seguindo o padrão *Model View Controller* (MVC) foi utilizado o *framework* Demoiselle. Uma característica importante do *framework* é que ele já define uma arquitetura de referência que provê independência de interface e de banco de dados. Usualmente, as aplicações são constituídas por pelo menos três camadas. Desta forma é comum separar as lógicas de apresentação, regras de negócio e persistência. O Demoiselle já fornece estereótipos que visam tornar esta separação mais clara, respectivamente: `@ViewController`, `@BusinessController` e `@PersistenceController`.

Outra característica é que no Eclipse há um *plugin* chamado Demoiselle Nimble para auxiliar na criação dos projetos. Este *plugin* é um processador automatizado de *templates* genéricos de código Java. A Figura 11 mostra todos os *templates* disponíveis pelo *plugin* Demoiselle Nimble no Eclipse. Para a criação do protótipo foi utilizado o *template* Demoiselle v2 Generate JSF2 (Primefaces 3) CRUD que realiza toda a geração dos artefatos necessários para as operações de CRUD de uma dada entidade, baseando-se no *framework* Demoiselle versão 2, e utilizando as bibliotecas JSF 2.0 e PrimeFaces a partir da versão 3.0. Todos os *templates* oferecidos pelo Demoiselle carregam consigo a sua arquitetura de referência com três módulos básicos: Core, Componentes e Extensões.

Figura 11 - *Templates* Demoiselle

O Core do Demoiselle contém as funcionalidades comuns a todos os projetos e extensões do *framework* de forma independente de camadas de apresentação e persistência. O Core é o núcleo do *framework*. Sem ele, os componentes e as extensões não funcionariam.

As Extensões estendem o Core com funcionalidades extras e específicas a um domínio ou tecnologia. Neste contexto, caso a aplicação necessite de persistência com JPA ou da biblioteca de componentes Primefaces, o *framework* fornecerá facilidades através destas extensões. Cabe destacar que as extensões não possuem vida própria, pois estão diretamente ligadas ao núcleo do *framework*.

Os Componentes são artefatos separados e não são dependentes diretamente do Core. Desta forma, o seu ciclo de vida é totalmente independente do Core e Extensões. Um componente disponibiliza novas funcionalidades ao usuário e não precisa, necessariamente, estender o comportamento do Core.

Fazendo uso da solução proposta pelo Apache Maven, o Demoiselle disponibiliza alguns arquivos `pom.xml`, que são os arquivos *eXtensible Markup Language* (XML) que contém todas as informações necessárias para a ferramenta gerenciar o projeto, entre as quais está o gerenciamento de dependências (bibliotecas) e build do projeto. Estes arquivos foram divididos em perfis, para que o desenvolvedor possa escolher qual usar de acordo com o tipo de aplicação que está desenvolvendo.

No protótipo criado neste trabalho foi utilizada a extensão `demoiselle-jsf-parent` como pode ser observado nas linhas 13 a 17 do Quadro 1. Esta especialização contém as

configurações úteis e necessárias para todas as aplicações que utilizarão a tecnologia JSF 2.0 para camada de apresentação. Também pode ser verificada a utilização do Core (linha 22) e dos temas (linha 27) do *framework* Primefaces. Mais abaixo, o componente `mysql-connector-java` define o *driver* de conexão com o banco de dados MySQL. Por último, a extensão `demoiselle-jpa` (linha 37) será responsável por delegar o controle das transações para o `javax.persistence.EntityManager` da especificação JPA.

Quadro 1 - Arquivo Apache Maven `pom.xml`

1	<code><?xml version="1.0" encoding="UTF-8"?></code>
2	<code><project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>
3	<code> xmlns="http://maven.apache.org/POM/4.0.0"</code>
4	<code> xsi:schemaLocation="http://maven.apache.org/POM/4.0.0</code>
5	<code> http://maven.apache.org/maven-v4_0_0.xsd"></code>
6	
7	<code> <modelVersion>4.0.0</modelVersion></code>
8	<code> <groupId>br.com.furb.pmattiollo</groupId></code>
9	<code> <artifactId>tcc</artifactId></code>
10	<code> <version>0.0.1</version></code>
11	<code> <packaging>war</packaging></code>
12	
13	<code> <parent></code>
14	<code> <groupId>br.gov.frameworkdemoiselle</groupId></code>
15	<code> <artifactId>demoiselle-jsf-parent</artifactId></code>
16	<code> <version>2.4.0</version></code>
17	<code> </parent></code>
18	
19	<code> <dependencies></code>
20	<code> <dependency></code>
21	<code> <groupId>org.primefaces</groupId></code>
22	<code> <artifactId>primefaces</artifactId></code>
23	<code> <version>5.2</version></code>
24	<code> </dependency></code>
25	<code> <dependency></code>
26	<code> <groupId>org.primefaces.themes</groupId></code>
27	<code> <artifactId>all-themes</artifactId></code>
28	<code> <version>1.0.10</version></code>
29	<code> </dependency></code>
30	<code> <dependency></code>
31	<code> <groupId>mysql</groupId></code>
32	<code> <artifactId>mysql-connector-java</artifactId></code>
33	<code> <version>5.1.34</version></code>
34	<code> </dependency></code>
35	<code> <dependency></code>
36	<code> <groupId>br.gov.frameworkdemoiselle</groupId></code>
37	<code> <artifactId>demoiselle-jpa</artifactId></code>
38	<code> <scope>compile</scope></code>
39	<code> </dependency></code>
40	<code> </dependencies></code>
41	<code></project></code>

A partir da definição das entidades, como pode ser observado na Figura 6, o *plugin* Demoiselle Nimble é capaz de gerar os pacotes necessários para a criação da aplicação, as classes propriamente ditas e os arquivos de configuração de acordo com o *template* escolhido. No caso do *template* escolhido, o Demoiselle v2 Generate JSF2 (Primefaces 3) CRUD, os principais arquivos de configuração gerados são: `messages.properties`, `web.xml`, `persistence.xml` e `beans.xml`. Estes arquivos são responsáveis pela configuração das mensagens de erro do projeto, pelo como será feito o *deploy* do projeto web, pela configuração das entidades que vão ser persistidas no banco de dados e pela delegação dos interceptadores das requisições de segurança e exceções gerenciadas pelo próprio Demoiselle, respectivamente.

Nos quadros 2, 3, 4 e 5 os arquivos podem ser visualizados com as configurações utilizadas no projeto do protótipo.

Quadro 2 - Arquivo `message.properties`

1	<code>button.add.new=Insert New</code>
2	<code>button.back=Back</code>
3	<code>button.delete=Remove</code>
4	<code>button.dialog.no=No, sorry\!</code>
5	<code>button.dialog.yes=Yes, sure\!</code>
6	<code>button.edit=Edit</code>
7	<code>button.new=New</code>
8	<code>button.save=Save</code>
9	<code>button.finalize=Finalize</code>
10	
11	<code>label.action=Action</code>
12	<code>label.dialog.alert=Alert</code>
13	<code>label.dialog.delete=Remove</code>
14	<code>label.confirm.delete=Confirm?</code>
15	<code>label.date.pattern=dd/MM/yyyy hh:mm:ss</code>
16	
17	<code>main.app.title=CEP-S</code>
18	<code>main.app.welcome=Welcome to the example application CEP-S.</code>
19	
20	<code>page.error.404.title=Error 404</code>
21	<code>page.error.404.message=Page not found.</code>
22	
23	<code>page.error.500.title=Error 500</code>
24	<code>page.error.500.message=Sorry, the system fail. Try to redo the operation.</code>
25	<code>page.error.500.message.contact=contact</code>
26	
27	<code>control.aces.has.not.paper=User has not paper.</code>
28	<code>control.aces.has.not.permission=User has not permission.</code>

Quadro 3 - Arquivo web.xml

1	<?xml version="1.0"?>
2	<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3	xmlns="http://java.sun.com/xml/ns/javaee"
4	xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
5	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6	http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
7	version="3.0">
8	<filter>
9	<filter-name>Character Encoding Filter</filter-name>
10	<filter-class>br.com.furb.pmattiollo.tcc.util.CharacterEncodingFilter</filter-
11	class>
12	</filter>
13	<filter-mapping>
14	<filter-name>Character Encoding Filter</filter-name>
15	<url-pattern>*/</url-pattern>
16	</filter-mapping>
17	<filter>
18	<filter-name>PrimeFaces FileUpload Filter</filter-name>
19	<filter-class>org.primefaces.webapp.filter.FileUploadFilter</filter-class>
20	</filter>
21	<filter-mapping>
22	<filter-name>PrimeFaces FileUpload Filter</filter-name>
23	<servlet-name>Faces Servlet</servlet-name>
24	</filter-mapping>
25	<context-param>
26	<param-name>primefaces.THEME</param-name>
27	<param-value>black-tie</param-value>
28	</context-param>
29	<servlet>
30	<servlet-name>Faces Servlet</servlet-name>
31	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
32	<load-on-startup>1</load-on-startup>
33	</servlet>
34	<servlet-mapping>
35	<servlet-name>Faces Servlet</servlet-name>
36	<url-pattern>*.jsf</url-pattern>
37	</servlet-mapping>
38	<security-constraint>
39	<display-name>Restrict raw XHTML Documents</display-name>
40	<web-resource-collection>
41	<web-resource-name>XHTML</web-resource-name>
42	<url-pattern>*.xhtml</url-pattern>
43	</web-resource-collection>
44	<auth-constraint />
45	</security-constraint>
46	</web-app>

Quadro 4 - Arquivo `persistence.xml`

1	<code><?xml version="1.0" encoding="UTF-8"?></code>
2	<code><persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"</code>
3	<code> xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>
4	<code> xsi:schemaLocation="http://java.sun.com/xml/ns/persistence</code>
5	<code> http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"></code>
6	
7	<code> <persistence-unit name="CEPDS" transaction-type="RESOURCE_LOCAL"></code>
8	<code> <non-jta-data-source>java:jboss/datasources/CEPDS</non-jta-data-source></code>
9	
10	<code> <class>br.com.furb.pmattiollo.tcc.domain.CalculationEntity</class></code>
11	<code> <class>br.com.furb.pmattiollo.tcc.domain.CollectEntity</class></code>
12	<code> <class>br.com.furb.pmattiollo.tcc.domain.ItemEntity</class></code>
13	<code> <class>br.com.furb.pmattiollo.tcc.domain.SoftwareEntity</class></code>
14	<code> <class>br.com.furb.pmattiollo.tcc.domain.ProcessEntity</class></code>
15	<code> <class>br.com.furb.pmattiollo.tcc.domain.UserEntity</class></code>
16	
17	<code> <properties></code>
18	<code> <property name="hibernate.dialect"</code>
19	<code> value="org.hibernate.dialect.MySQL5InnoDBDialect" /></code>
20	<code> <property name="hibernate.show_sql" value="true" /></code>
21	<code> <property name="hibernate.format_sql" value="false" /></code>
22	<code> <property name="hibernate.hbm2ddl.auto" value="update" /></code>
23	<code> </properties></code>
24	<code> </persistence-unit></code>
25	<code></persistence></code>

Quadro 5 - Arquivo `beans.xml`

1	<code><beans xmlns="http://java.sun.com/xml/ns/javaee"</code>
2	<code> xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>
3	<code> xsi:schemaLocation="http://java.sun.com/xml/ns/javaee</code>
4	<code> http://java.sun.com/xml/ns/javaee/beans_1_0.xsd"></code>
5	
6	<code> <interceptors></code>
7	<code> <class>br.gov.frameworkdemoiselle.transaction.TransactionalInterceptor</class></code>
8	<code> <class>br.gov.frameworkdemoiselle.security.RequiredPermissionInterceptor</class></code>
9	<code> <class>br.gov.frameworkdemoiselle.security.RequiredRoleInterceptor</class></code>
10	<code> <class>br.gov.frameworkdemoiselle.exception.ExceptionHandlerInterceptor</class></code>
11	<code> </interceptors></code>
12	<code></beans></code>

Na parte de geração de código, para cada entidade escolhida, o *template* gerou arquivos pertencentes às três camadas do MVC. Na camada de visão foram criados os arquivos `.xhtml`, que definem a utilização dos componentes visuais, principalmente do Primefaces e o pacote `view` com classes Java contendo a anotação `@ViewController`. Na camada de controle foram gerados os pacotes `convertes` e `business` contendo classes com as anotações `@FacesConverter` e `@BusinessController`, respectivamente. Por fim, na camada de persistência, foi criado o pacote `persistence` contendo classes com a anotação `@PersistenceController`.

Para demonstrar o papel do *Demoiselle Nimble*, são exibidos nas imagens a seguir todos os arquivos gerados para a funcionalidade de cadastro de usuários no sistema. Essa é uma das funcionalidades do protótipo, tendo como objetivo criar os usuários e permitir o acesso dos mesmos.

Pode-se verificar no Quadro 6 que alguns componentes, como por exemplo: `p:menubar`, `p:submenu` e `p:menuitem` do *Primefaces*, são utilizados através do prefixo `p:` nas *tags* do arquivo. No Quadro 7 é demonstrado o arquivo Java pertencente à camada de visão definida pela implementação do *Demoiselle*. Pode-se verificar que, além da anotação de referência, a classe generaliza o comportamento da classe `AbstractEditPageBean` já implementado.

Quadro 6 - Arquivo `xhtml` gerado

1	<code><ui:composition xmlns="http://www.w3.org/1999/xhtml"</code>
2	<code>xmlns:f="http://java.sun.com/jsf/core"</code>
3	<code>xmlns:p="http://primefaces.org/ui"</code>
4	<code>xmlns:h="http://java.sun.com/jsf/html"</code>
5	<code>xmlns:ui="http://java.sun.com/jsf/facelets"></code>
6	<code><h:form></code>
7	<code><p:menubar></code>
8	<code><p:submenu label="#{messages['menu.user']}"></code>
9	<code><p:menuitem value="#{messages['menu.menuitem.new']}">/></code>
10	<code><p:menuitem value="#{messages['menu.menuitem.list']}">/></code>
11	<code></p:submenu></code>
12	<code><p:submenu label="#{messages['menu.process']}"></code>
13	<code><p:menuitem value="#{messages['menu.menuitem.new']}">/></code>
14	<code><p:menuitem value="#{messages['menu.menuitem.list']}">/></code>
15	<code></p:submenu></code>
16	<code><p:submenu label="#{messages['menu.item']}"></code>
17	<code><p:menuitem value="#{messages['menu.menuitem.new']}">/></code>
18	<code><p:menuitem value="#{messages['menu.menuitem.list']}">/></code>
19	<code></p:submenu></code>
20	<code><p:submenu label="#{messages['menu.software']}"></code>
21	<code><p:menuitem value="#{messages['menu.menuitem.new']}">/></code>
22	<code><p:menuitem value="#{messages['menu.menuitem.list']}">/></code>
23	<code></p:submenu></code>
24	<code><p:submenu label="#{messages['menu.collect']}"></code>
25	<code><p:menuitem value="#{messages['menu.menuitem.new']}">/></code>
26	<code><p:menuitem value="#{messages['menu.menuitem.list']}">/></code>
27	<code></p:submenu></code>
28	<code></p:menubar></code>
29	<code></h:form></code>
30	<code></ui:composition></code>

Quadro 7 - Arquivo do pacote `view`

```

1  @ViewController
2  @PreviousView("./user_list.jsf")
3  public class UserEditMB extends AbstractEditPageBean<UserEntity, Long> {
4
5      @Inject
6      private UserBC userBC;
7
8      public List<SelectItem> getPaper() {
9          return userBC.getUserEnum();
10     }
11
12     @Override
13     @Transactional
14     public String delete() {
15         this.userBC.delete(getId());
16         return getPreviousView();
17     }
18
19     @Override
20     @Transactional
21     public String insert() {
22         this.userBC.insert(this.getBean());
23         return getPreviousView();
24     }
25
26     @Override
27     @Transactional
28     public String update() {
29         this.userBC.update(this.getBean());
30         return getPreviousView();
31     }
32
33     @Override
34     protected UserEntity handleLoad(Long id) {
35         return this.userBC.load(id);
36     }
37 }

```

Nos quadros 8 e 9 é definida a camada de controle do protótipo, a primeira estendendo a implementação de referência `DelegateCrud` e a segunda implementando a interface `Converter`, ambas também definidas pelo próprio `Demoiselle`. Representando a camada de persistência, têm-se o Quadro 10 com uma classe Java que generaliza a outra classe `JPACrud`.

Quadro 8 - Arquivo do pacote controller

```

1 @BusinessController
2 public class UserBC extends DelegateCrud<UserEntity, Long, UserDAO> {
3
4     public List<SelectItem> getUserEnum() {
5         List<SelectItem> varUserEnum = new ArrayList<SelectItem>();
6         for (UserEnum eachUserEnum : UserEnum.values()) {
7             varUserEnum.add(new SelectItem(eachUserEnum));
8         }
9         return varUserEnum;
10    }
11 }

```

Quadro 9 - Arquivo do pacote converter

```

1 @FacesConverter(value= "ConversorUser")
2 public class UserConverter implements Converter {
3     private UserDAO userDAO = Beans.getReference(UserDAO.class);
4     @Override
5     public Object getAsObject(FacesContext context, UIComponent component,
6         String value) {
7         try{
8             Object ret = null;
9             if (component instanceof PickList) {
10                Object dualList = ((PickList) component).getValue();
11                DualListModel<?> dl = (DualListModel<?>) dualList;
12                for (Object o : dl.getSource()) {
13                    String id = String.valueOf(((User) o).getId());
14                    if (value.equals(id)) {
15                        ret = o;
16                        break;
17                    }
18                }
19                if (ret == null)
20                    for (Object o : dl.getTarget()) {
21                        String id = String.valueOf(((User) o).getId());
22                        if (value.equals(id)) {
23                            ret = o;
24                            break;
25                        }
26                    }
27            } else {
28                if (value.trim().equals("")) {
29                    ret = null;
30                } else {
31                    Long varId = Long.valueOf(value);
32                    ret = userDAO.load(varId);
33                }
34            }
35            return ret;
36        } catch (Exception e) {
37            e.printStackTrace();
38            return null;
39        }
40    }
41 }

```

Quadro 10 - Arquivo do pacote persistence

```

1  @PersistenceController
2  public class UserDAO extends JPACrud<UserEntity, Long> {
3
4      public UserEntity findById(Long id) throws Exception{
5          Query query = getEntityManager().createNamedQuery("UserEntity.findById");
6          query.setParameter("id", id);
7
8          UserEntity user = null;
9
10         try {
11             user = (UserEntity) query.getSingleResult();
12         } catch(Exception e) {
13             e.printStackTrace();
14         }
15
16         return user;
17     }
18 }

```

3.3.3 Operacionalidade da implementação

Esta seção busca apresentar as operações disponibilizadas pelo protótipo desenvolvido, além de aplicar e justificar o modelo do CEP nas empresas desenvolvedoras de software. Para realizar a demonstração e a fim de demonstrar os resultados obtidos, foi simulado o processo de classificação e acompanhamento de uma avaliação de um software e seus respectivos itens.

O modelo de CEP voltado para a área de desenvolvimento de software foi proposto visando apoiar principalmente as pequenas e médias empresas. Estas empresas são as mais impactadas pela dificuldade de estimativas, visto o grau de subjetividade das medidas de software e pela produção de software sob encomenda, em cada produção, ser um produto único.

Outro grande problema está no fato de que as práticas de medição de software adotadas pelas organizações são aplicadas somente nos modelos de alta maturidade ou tendem a abandonar a prática tão logo obtenham sucesso na avaliação. Segundo Barcellos (2014, p. 58) a realização da medição nos níveis mais elevados de maturidade inclui novos aspectos não abordados pela medição de software tradicional. Não realizar a medição corretamente nos níveis iniciais irá retardar a implantação das práticas da alta maturidade, principalmente o controle estatístico de processos. Independentemente do nível de maturidade em que é realizada, a medição fornece informações úteis ao entendimento da organização e à tomada de decisão.

Nesse contexto, o desenvolvimento deste trabalho propõe uma forma de acompanhamento estatístico dos projetos e das métricas de qualidade definidas sobre eles,

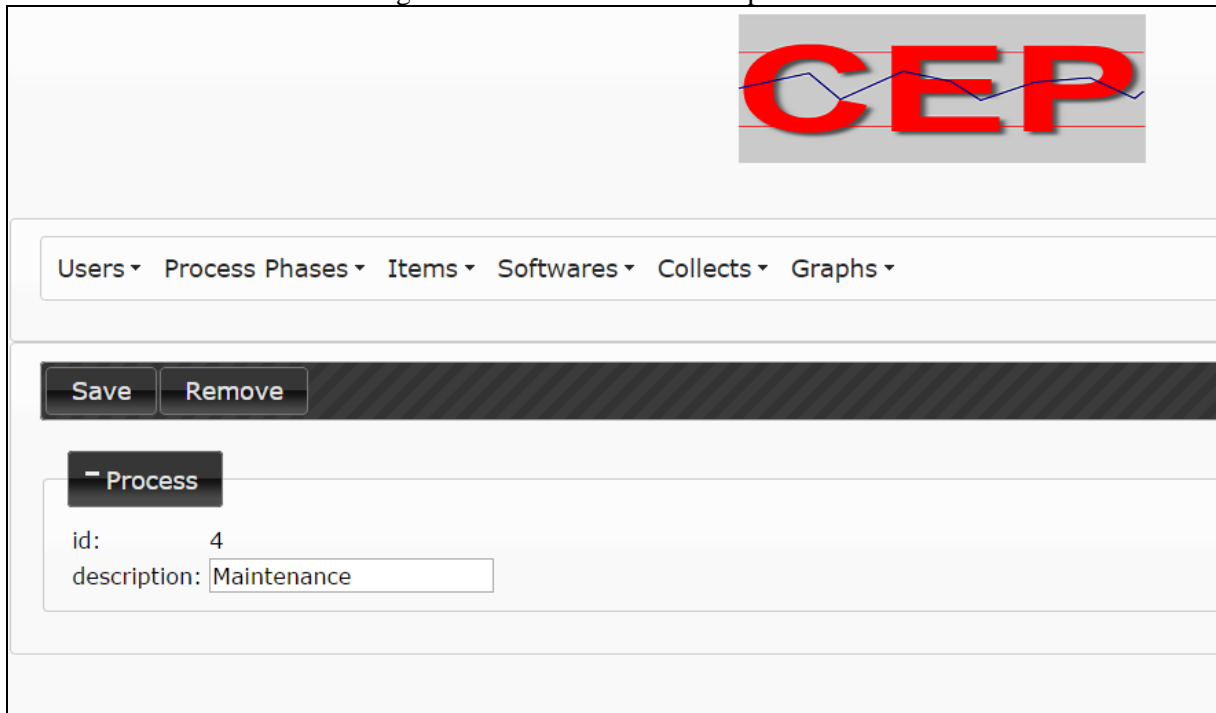
sem depender do nível de maturidade e do tamanho da empresa que o mantém. Para tanto, neste protótipo, a forma de análise e acompanhamento se baseia em gráficos de controle estatísticos, tornando mais fácil a verificação dos índices de capacidade e estabilidade dos processos. Um gráfico de controle, por sua vez, é definido por uma linha central (LC) que representa o valor médio da característica de qualidade, a linha superior que representa o Limite Superior de Controle (LSC) e a linha inferior que representa o Limite Inferior de Controle (LIC).

Dentre os diversos gráficos de controle disponibilizados pelo CEP foram escolhidos três para compor este trabalho, são eles: Gráfico X-Barra Individual, Gráfico Média Móvel Exponencialmente Ponderada e Gráfico de Controle para Não-Conformidades por unidade ou Defeitos. Os gráficos foram escolhidos por serem os mais indicados para amostras de tamanho igual a um que, em geral, são colhidas as amostras de desenvolvimento de software. Estes gráficos citados serão apresentados a seguir na utilização do protótipo.

Para demonstrar a utilização do protótipo será realizada a sua aplicação sob um projeto web de tráfego de arquivos. O projeto já está finalizado e está sendo utilizado pelo cliente final há cerca de cinco anos. Os valores foram coletados desde a sua entrada em produção com um intervalo de três em três meses, aproximadamente.

Iniciando a utilização do protótipo o usuário administrador deve determinar quais as fases do processo de desenvolvimento de software serão abordadas, pois o software ou projeto pode ser avaliado desde a elicitação dos seus requisitos até a utilização do usuário final. Esta fase será vinculada às métricas da avaliação. No caso do software avaliado a fase de avaliação utilizada será a de manutenção, do inglês *maintenance*, como exibido na Figura 12.

Figura 12 - Cadastro de fase do processo



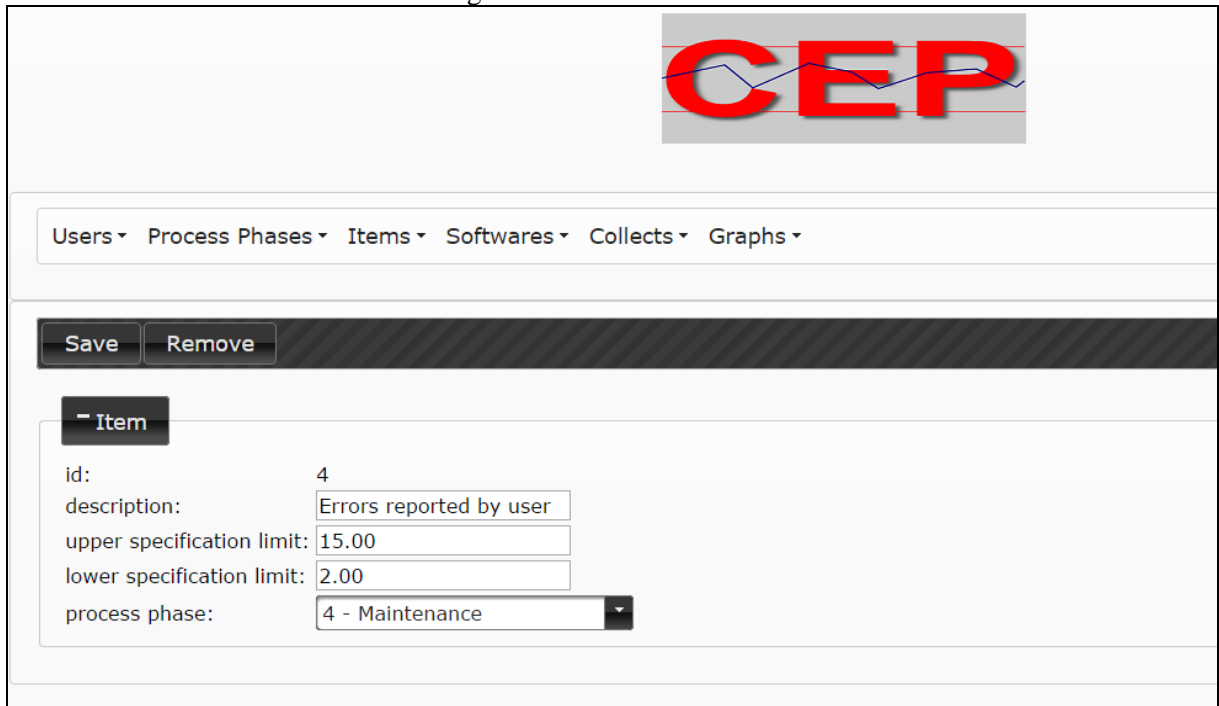
The image shows a web application interface for 'Cadastro de fase do processo'. At the top right, there is a logo with the letters 'CEP' in red, overlaid with a blue line graph. Below the logo is a navigation menu with the following items: 'Users', 'Process Phases', 'Items', 'Softwares', 'Collects', and 'Graphs'. Underneath the menu is a dark bar containing two buttons: 'Save' and 'Remove'. Below this bar is a section titled 'Process' with a minus sign icon. Inside this section, there is a form with two fields: 'id:' followed by the value '4', and 'description:' followed by a text input field containing the word 'Maintenance'.

Em seguida, após uma análise de qualidade devem ser definidos os itens ou as métricas a serem observadas. As medidas devem estar alinhadas a objetivos organizacionais (ou do próprio projeto) a fim de fornecerem informações úteis para as tomadas de decisão. De acordo com Barcellos (2014, p. 23) a definição das medidas deve atender os seguintes critérios:

- a) devem ter definições operacionais completas e satisfatórias para que sejam realizadas medições consistentes;
- b) dados agregados não devem ser considerados;
- c) devem ter baixa granularidade;
- d) devem estar relacionadas a processos críticos;
- e) devem ser consistentes.

A Figura 13 mostra como são criadas as medidas e como são vinculadas às fases do processo.

Figura 13 - Cadastro de item



Users ▾ Process Phases ▾ Items ▾ Softwares ▾ Collects ▾ Graphs ▾

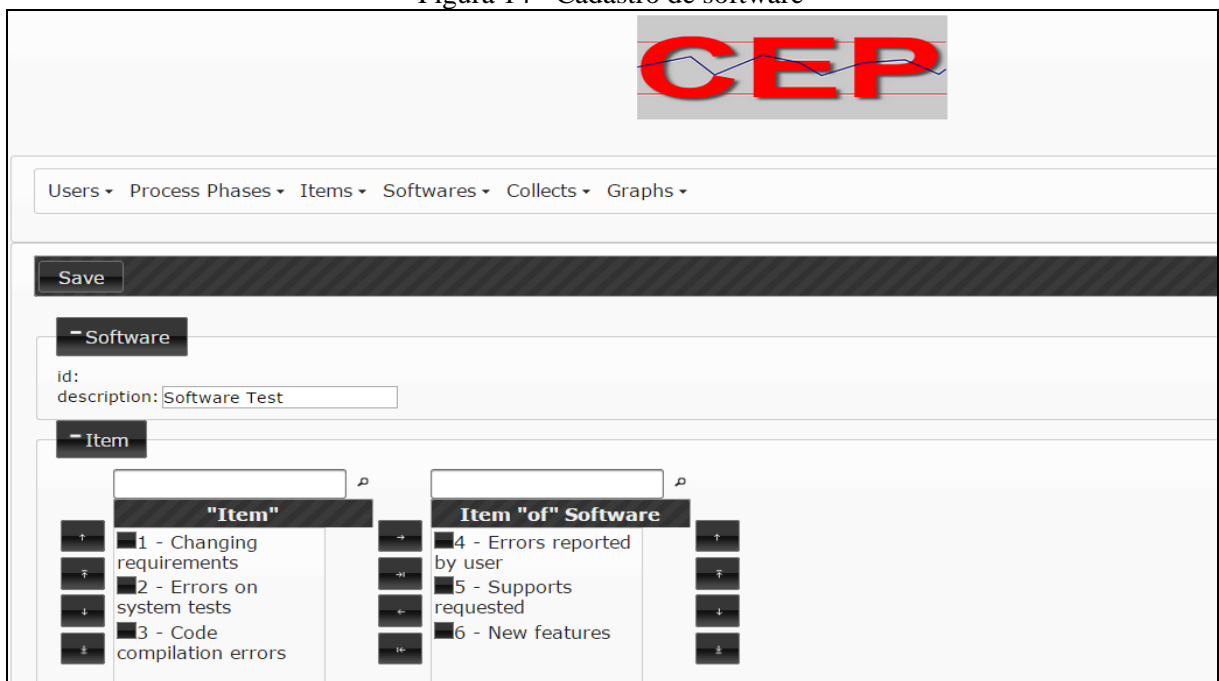
Save Remove

- Item

id: 4
description: Errors reported by user
upper specification limit: 15.00
lower specification limit: 2.00
process phase: 4 - Maintenance ▾

Ainda no cadastro das medidas ou dos itens a serem avaliados no projeto, devem ser definidos os limites de controle do processo a partir da “voz do cliente”. Estes limites, assim como os demais dados deste cadastro, são definidos pelo usuário administrador e descrevem o comportamento esperado para o processo quando ele é executado ou a partir de dados históricos. Depois de finalizado o cadastro de itens, o administrador é responsável pelo cadastro do software a ser avaliado. Nesta etapa serão vinculados todos os itens que se deseja avaliar no projeto como mostrado na Figura 14.

Figura 14 - Cadastro de software



Users ▾ Process Phases ▾ Items ▾ Softwares ▾ Collects ▾ Graphs ▾

Save

- Software

id:
description: Software Test

- Item

"Item"	Item "of" Software
1 - Changing requirements	4 - Errors reported by user
2 - Errors on system tests	5 - Supports requested
3 - Code compilation errors	6 - New features

Com as fases do processo, os itens e os softwares cadastrados o administrador deve comunicar o usuário operador que as coletas dos valores já podem ser realizadas. O processo de coleta poderá ficar em aberto por período indeterminado e pode se tornar uma tarefa custosa. Para tanto, o administrador poderá monitorar o processo e, eventualmente, corrigir algum valor erroneamente inserido.

Na coleta de valores, o operador poderá inserir uma amostragem de valor coletado e vinculá-lo a um software e a um item disponível, ou seja, após ter um software selecionado, somente serão disponibilizados os itens anteriormente associados a ele. Após uma coleta ser finalizada, o usuário operador poderá comunicar o fato ao administrador e as etapas de análise estatística e a classificação já poderão ser acompanhadas.

Na Figura 15 é representada uma coleta feita após o software entrar em produção durante um mês. O item avaliado é o Número de erros reportado pelo usuário final. Após esta amostra, como já citado anteriormente, foi mantido um intervalo de aproximadamente três meses entre as próximas.

Figura 15 - Cadastro de coleta



The screenshot shows the CEP system interface for recording a collection. At the top right, there is a logo for CEP (Control and Evaluation Process) featuring a line graph. Below the logo, there is a navigation menu with the following items: Process Phases, Items, Softwares, and Collects. A 'Save' button is located below the navigation menu. Underneath the 'Save' button, there is a 'Collect' form. The form contains the following fields: 'id' (empty), 'date' (31/10/2010 12:00:00), 'software' (4 - Software Test), 'item' (4 - Errors reported by user), and 'value' (19).

Quanto maior o número de coletas registradas, mais preciso será o resultado da avaliação obtida. Com base nos cinco anos de coletas, o administrador poderá gerar os gráficos de controle estatísticos disponibilizados pelo CEP, como é mostrado logo adiante. Tratando de um volume de dados grande e/ou de dados pertencentes a um projeto antigo, este processo de coleta manual às vezes pode ser uma tarefa trabalhosa, principalmente se os dados pertencerem a um projeto antigo.

Para facilitar o acesso às informações já processadas, muitas vezes por um período longo de tempo, o protótipo disponibiliza a opção de recuperar os dados já inseridos pela ferramenta SonarQube. Esta integração é realizada via banco de dados e pode ser realizada pelo usuário administrador, quando necessário, através da opção `integrate` como pode ser observado Figura 16.

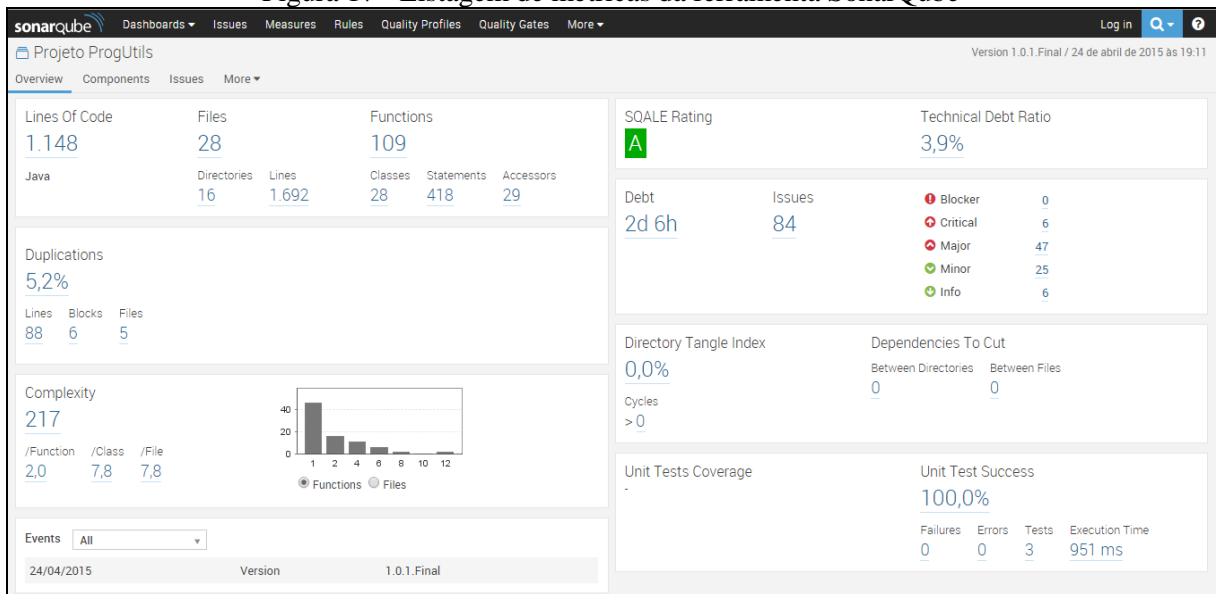
Figura 16 - Listagem dos itens

The screenshot shows the CEP application interface. At the top, there is a navigation menu with options: Users, Process Phases, Items, Softwares, Collects, and Graphs. Below the menu, there is a section titled 'Integrate' with a 'List of Items' table. The table has the following data:

id	description	upper specification limit	lower specification limit
1	Changing requirements	10.00	0.00
2	Errors on system tests	11.00	2.00
3	Code compilation errors	5.00	3.00
4	Errors reported by user	15.00	2.00
5	Supports requested	35.00	10.00
6	New features	15.00	5.00

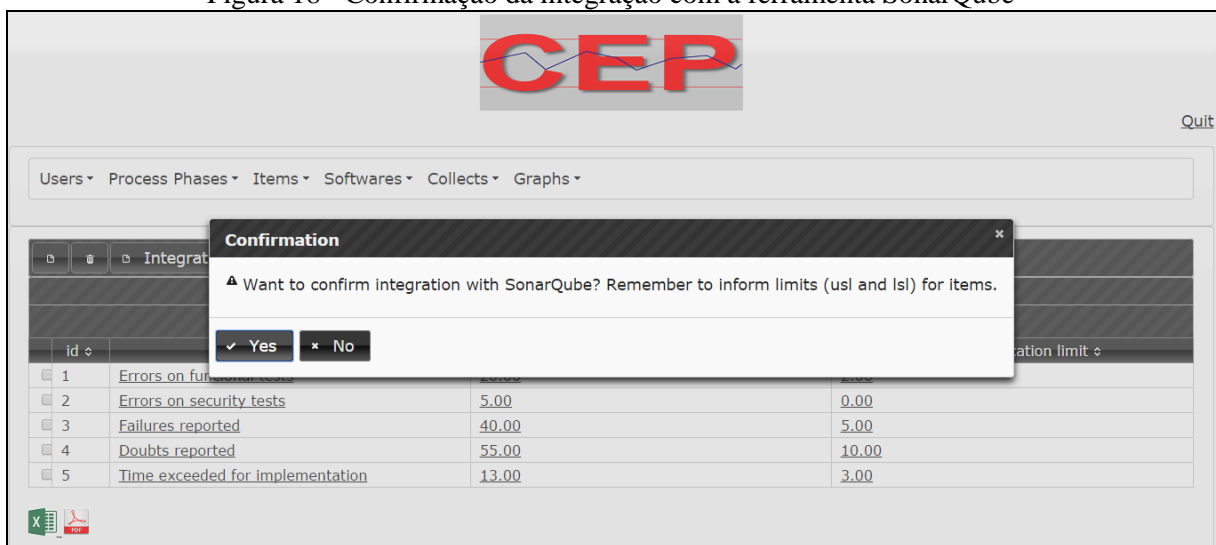
O conceito implementado pela ferramenta SonarQube é compatível com o modelo proposto pelo protótipo e, assim, as fases do processo, os softwares, os itens e as coletas podem ser obtidas através do processo de integração citado. Normalmente o volume de dados da ferramenta é grande, por isso a necessidade de extrair apenas as informações úteis é de suma importância. Durante o processo de captura de informações da base de dados do SonarQube foram consideradas as fases do processo `Tests`, `Development` e `Documentation` que tornam-se as mais compatíveis com o propósito do protótipo. Na Figura 17 são exibidas como exemplo, as métricas obtidas por meio da análise de código fonte de um projeto cadastrado na ferramenta SonarQube.

Figura 17 - Listagem de métricas da ferramenta SonarQube



Após o usuário administrador selecionar a opção de integração, o protótipo irá conectar-se a base de dados do SonarQube e recuperar as informações compatíveis com o modelo de dados, como foi descrito anteriormente. Na Figura 18 é mostrado a confirmação da operação de integração com a ferramenta SonarQube. Nesta etapa um aviso é mostrado ao usuário para informar que os valores referentes aos limites de especificação superior e inferior dos itens (ou métricas) a serem recuperados no processo de integração devem ser preenchidos posteriormente. Estes limites não são disponibilizados pelo SonarQube e se fazem necessários para a realização dos cálculos estatísticos.

Figura 18 - Confirmação da integração com a ferramenta SonarQube



O gráfico de controle X-Barra Individual ou gráfico de Shewhart, para amostras de tamanho igual a um, se constitui como alternativa ao uso dos gráficos convencionais. Para que sua utilização seja viável, é necessário que as amostras sejam independentes e que a

distribuição seja normal. A não normalidade da distribuição das amostras afeta os limites de controle de três sigma e tornam-se inapropriados. Como pode ser visto na Figura 19, este gráfico é mais indicado para quando o deslocamento na média do processo for entre moderada a grande. A Fórmula do gráfico pode ser analisada na Figura 20.

Figura 19 - Gráfico de controle X-Barra Individual

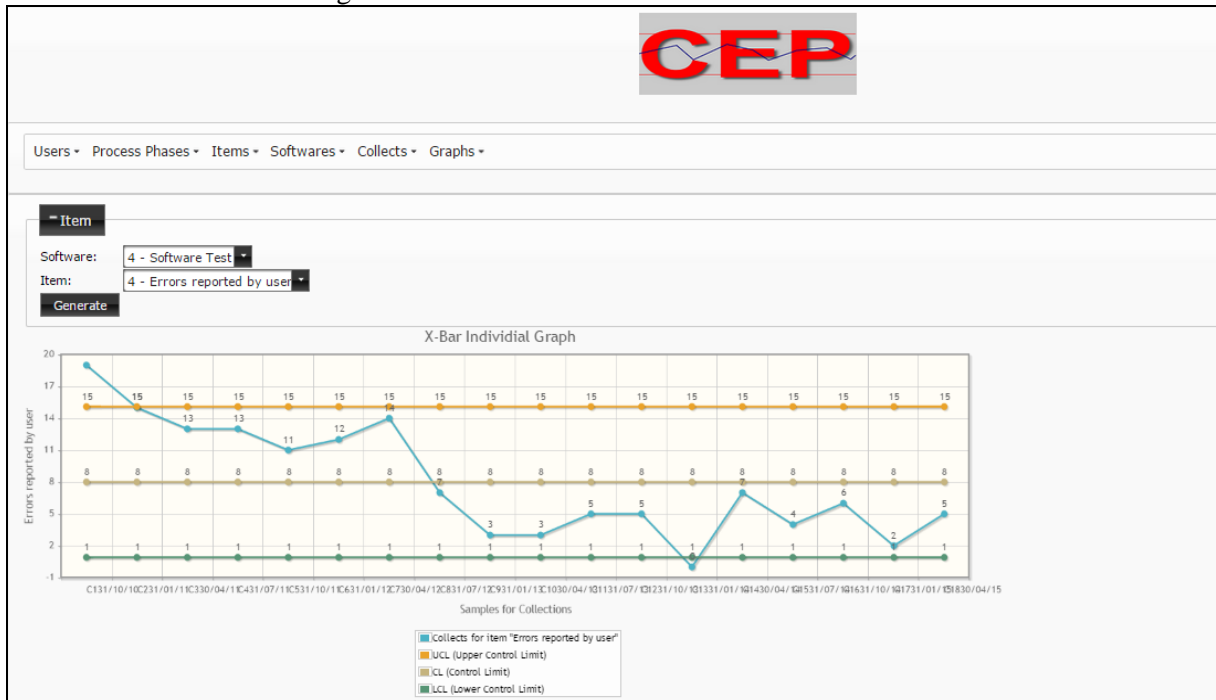


Figura 20 - Fórmula do gráfico de controle X-Barra Individual

$$LSC = \bar{x} + 3 \frac{\bar{M}R}{d_2}$$

$$LC = \bar{x}$$

$$LIC = \bar{x} - 3 \frac{\bar{M}R}{d_2}$$

Onde $\bar{M}R$ é a média da amplitude móvel e é calculado pela equação:

$$MR_i = |x_i - x_{i-1}|$$

$$d_2 = 1,128$$

Fonte: Fonseca (2010).

Para demonstrar a obtenção do resultado do LSC do gráfico de controle X-Barra Individual e a realização do cálculo da amplitude móvel, tomando como exemplo a gramatura de amostras fictícias, são apresentados os quadros 11 e 12, respectivamente.

Quadro 11 - Trecho de código para a geração do gráfico X-Barra Individual

```

1 public BigDecimal getUclResult() {
2     BigDecimal average = new BigDecimal(0.0);
3     BigDecimal preAverage = new BigDecimal(0.0);
4     BigDecimal curAverage = new BigDecimal(0.0);
5     BigDecimal mobileAverage = new BigDecimal(0.0);
6     BigDecimal divisor = new BigDecimal(0.0);
7     int count = 0;
8
9     for(CollectEntity collect : collectList) {
10        average = average.add(collect.getValue());
11        curAverage = collect.getValue();
12        if(count == 0) {
13            preAverage = collect.getValue();
14        } else {
15            mobileAverage = mobileAverage.add(curAverage.subtract(preAverage));
16            preAverage = curAverage;
17        }
18        count ++;
19    }
20    if(collectList.size() > 1) {
21        divisor = new BigDecimal(collectList.size() - 1);
22    } else {
23        divisor = new BigDecimal(collectList.size());
24    }
25    average = average.divide(new BigDecimal(collectList.size()), SCALE, ROUND);
26    mobileAverage = mobileAverage.divide(divisor, SCALE, ROUND);
27
28    ucl = mobileAverage.divide(new BigDecimal(D2), SCALE, ROUND);
29    ucl = average.add(new BigDecimal(SIGMA).multiply(ucl));
30    return ucl;
31 }

```

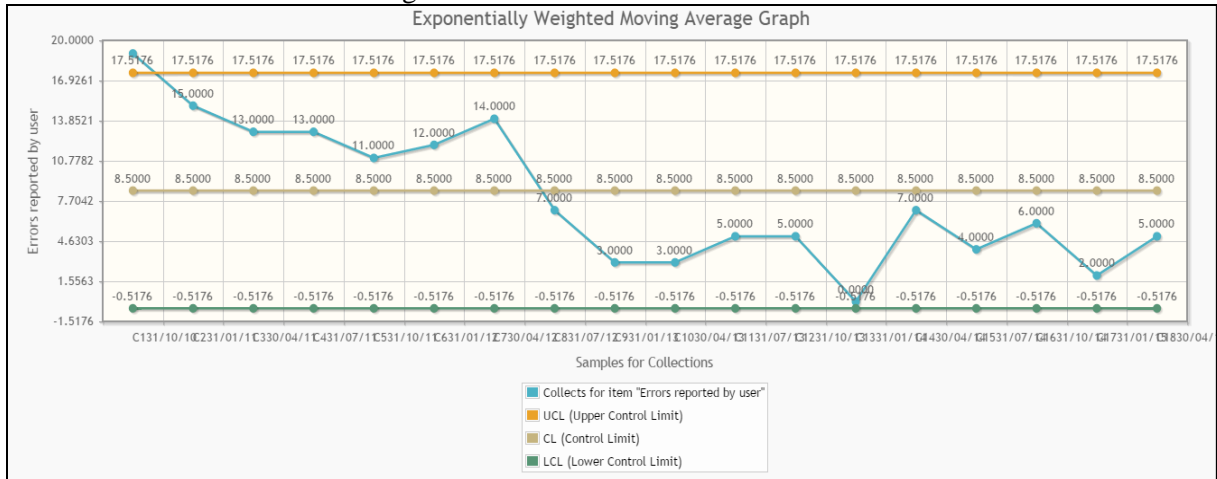
Quadro 12 - Cálculo da amplitude móvel

Lote	Gramatura (g/m ²)	MR : Amplitude Móvel
1	88,2	
2	88,9	0,7
3	90,5	1,6
4	90,3	0,2
5	90	0,3
6	90,8	0,8
7	91,4	0,6
8	91,3	0,1
9	89	2,3
10	90,7	1,7
Total	90,11	0,922222222

O gráfico de controle da Média Móvel Exponencialmente Ponderada (EWMA), identificado na Figura 21, é uma boa alternativa para detectar pequenas mudanças. Os modelos de gráficos de controle com memória, também denominados de gráficos de controle,

são aprimoramentos dos gráficos de controle desenvolvidos para minimizar a ocorrência de alarmes para causas não atribuíveis.

Figura 21 - Gráfico de controle EWMA



Na Figura 22 pode-se observar melhor o cálculo a ser aplicado sobre o gráfico de controle Média Móvel Exponencialmente Ponderada. A ponderação e escolha dos parâmetros λ e L para o procedimento de planejamento ótimo de um gráfico EWMA consistem na seleção adequada desta combinação (λ, L) capaz de fornecer o melhor desempenho. Quando $L = 3$ funciona razoavelmente bem, particularmente com valores maiores de λ , justificando os valores de 3 para L e $0,8 \lambda$ utilizados no protótipo. No Quadro 13 é apresentado um trecho do código implementado com base nos parâmetros definidos para a geração do gráfico de controle EWMA.

Figura 22 – Cálculo do gráfico EWMA

$$Z_i = \lambda x_i + (1 - \lambda)Z_{i-1}$$

Onde $0 < \lambda \leq 1$ é a constante e o valor inicial Z_0 é o alvo do processo, $Z_0 = \mu_0$.
Os parâmetros do gráfico MMEP são definidos como:

$$LSC = \mu_0 + L\sigma \sqrt{\frac{\lambda}{(2 - \lambda)} [1 - (1 - \lambda)^{2i}]}$$

$$LC = \mu_0$$

$$LIC = \mu_0 - L\sigma \sqrt{\frac{\lambda}{(2 - \lambda)} [1 - (1 - \lambda)^{2i}]}$$

Fonte: Fonseca (2010).

Quadro 13 - Trecho de código para a geração do gráfico EWMA

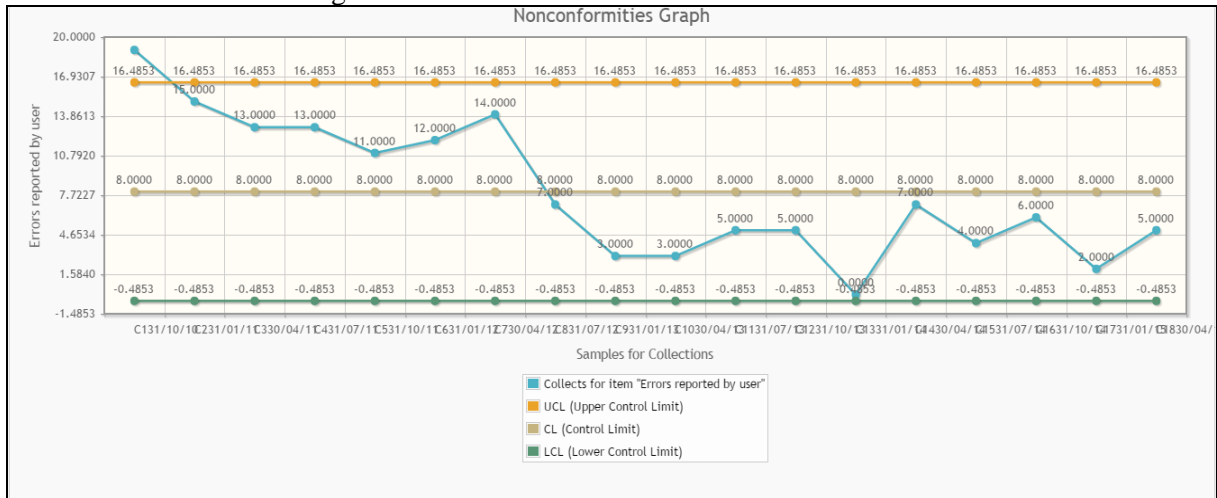
```

1  @Override
2  public BigDecimal getUclResult() {
3      BigDecimal u0 = getU0();
4      BigDecimal staDev = new BigDecimal(getStandardDeviation(collectList));
5
6      ucl = staDev.multiply(new BigDecimal(Math.sqrt(Y / (2 / Y))));
7      ucl = u0.add(new BigDecimal(SIGMA).multiply(ucl));
8      return ucl;
9  }
10
11 @Override
12 public BigDecimal getClResult() {
13     return getU0();
14 }
15
16 @Override
17 public BigDecimal getLclResult() {
18     if(lcl == null) {
19         BigDecimal u0 = getU0();
20         BigDecimal staDev = new BigDecimal(getStandardDeviation(collectList));
21
22         lcl = staDev.multiply(new BigDecimal(Math.sqrt(Y / (2 / Y))));
23         lcl = u0.subtract(new BigDecimal(SIGMA).multiply(lcl));
24     }
25
26     return lcl;
27 }
28
29 private BigDecimal getU0() {
30     BigDecimal average = item.getUsl().subtract(item.getLsl());
31     average = average.divide(new BigDecimal(2), SCALE, ROUND);
32
33     return item.getLsl().add(average);
34 }

```

De acordo com Fonseca (2010, p. 54), os gráficos para controle de número de defeitos são muito utilizados em serviços e em melhorias de processos não industriais, onde as características de interesse são, em geral, mensuráveis em uma escala não numérica. São conhecidos como gráficos para controle de atributos e normalmente são menos precisos que os gráficos para o controle de variáveis. O gráfico gerado pelo protótipo pode ser observado na Figura 23.

Figura 23 - Gráfico de Não conformidades/Defeitos



A fórmula para a geração do gráfico de controle para Não conformidades é definida na Figura 24. Seguindo a implementação do protótipo, se dá a utilização da fórmula apresentada na literatura, porém com algumas peculiaridades. Como já citado anteriormente, os gráficos utilizados são mais indicados para amostras de tamanho igual a um, tornando o parâmetro n (número de amostras) do cálculo insignificante. Tal consideração pode ser visualizada no trecho de código do Quadro 14.

Figura 24 - Fórmula do gráfico de controle de Não conformidades/Defeitos

$$LSC = \bar{u} + 3\sqrt{\frac{\bar{u}}{n}}$$

$$LC = \bar{u}$$

$$LIC = \bar{u} - 3\sqrt{\frac{\bar{u}}{n}}$$

Onde $\bar{u} = \frac{\sum Defeitos}{\sum Amostras}$ e o valor plotado é o $\frac{Defeitos_i}{Amostra_i}$

e n = número de amostras

Fonte: Fonseca (2010).

Quadro 14 - Trecho de código para a geração do gráfico de Não conformidades/Defeitos

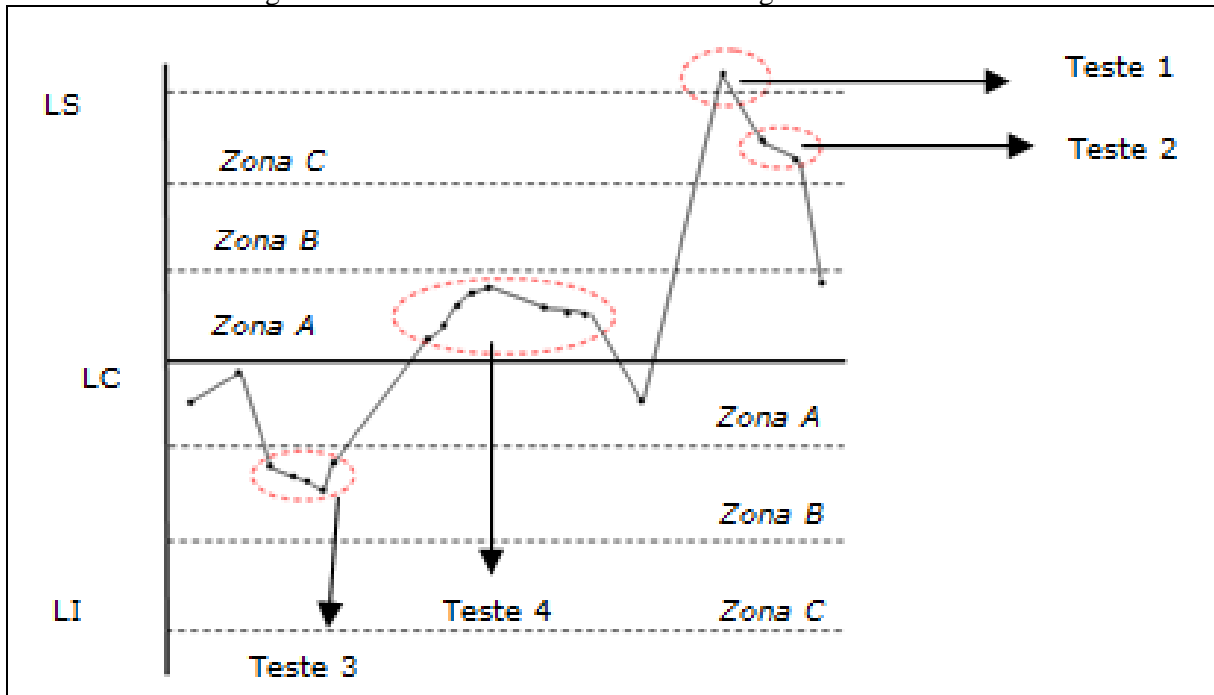
```
1 public BigDecimal getUclResult() {
2     BigDecimal sumDefects = new BigDecimal(0.0);
3     BigDecimal average = new BigDecimal(0.0);
4
5     for(CollectEntity collect : collectList) {
6         sumDefects = sumDefects.add(collect.getValue());
7     }
8
9     average = sumDefects.divide(new BigDecimal(collectList.size()), SCALE, ROUND);
10
11     BigDecimal sqrtValue = new BigDecimal(Math.sqrt(average.doubleValue()));
12
13     ucl = average.add(new BigDecimal(SIGMA).multiply(sqrtValue));
14     return ucl;
15 }
```

Após todos os cálculos serem realizados e dos gráficos gerados, é realizada a classificação dos itens dos softwares de acordo com a sua capacidade e estabilidade dentro do processo. Para tanto existem alguns testes que devem ser realizados. Para determinar se um processo ou um item é estável existem diversos testes que podem ser realizados, porém apenas os quatro principais são realizados:

- a) teste 1: presença de algum ponto fora dos limites de controle três sigma;
- b) teste 2: presença de pelo menos dois de três valores sucessivos do mesmo lado e a mais de dois sigma da linha central;
- c) teste 3: presença de pelo menos quatro de cinco valores sucessivos do mesmo lado e a mais de um sigma da linha central;
- d) teste 4: presença de oito pontos sucessivos no mesmo lado da linha central.

Na Figura 25 são exibidos os testes de estabilidade quando aplicados sobre um gráfico de controle qualquer, onde LS representa o limite superior, LC o limite central e LI o limite inferior.

Figura 25 - Testes de estabilidade sobre um gráfico de controle



Fonte: Barcellos (2014).

Em seguida, é realizado o teste de capacidade. Os limites de controle do processo são a “voz do processo” e os limites que se esperam que o processo alcance são a “voz do cliente”. De forma simples, um processo é capaz se a “voz do processo” atende a “voz do cliente”. Os limites máximo e mínimo são determinados no momento do cadastro do item, tal como foi mostrado na Figura 13.

Seguindo o cálculo mostrado na Figura 26 pode-se determinar que se o índice de capacidade é igual ou maior que 1 e os limites do processo são internos aos limites especificados, o processo é capaz. Se o índice de capacidade é menor que 1, o processo não é capaz.

Figura 26 - Cálculo para o teste de capacidade do processo

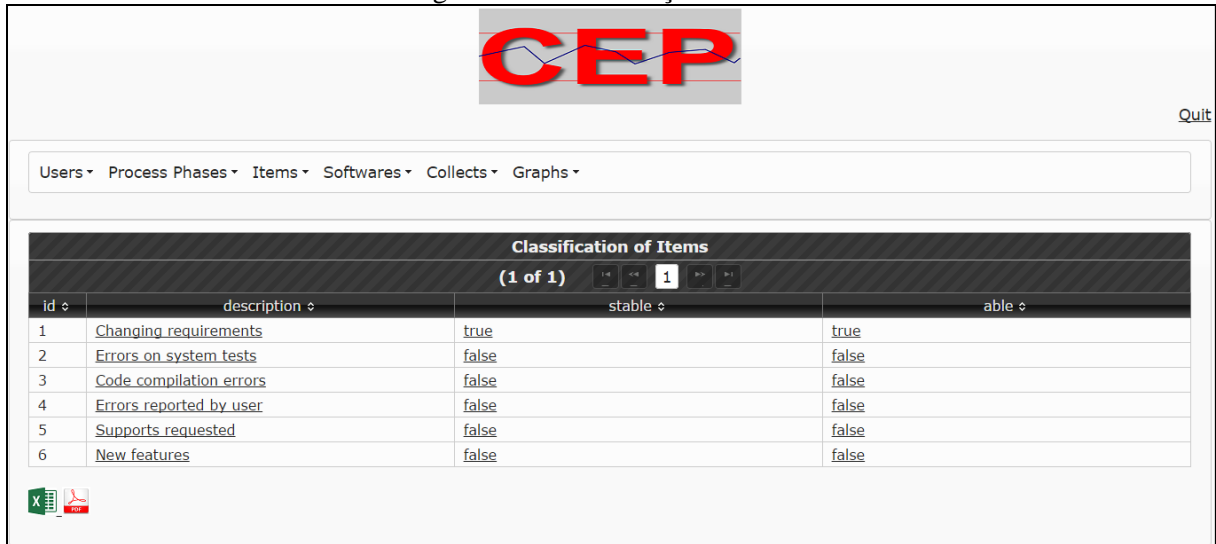
$$C_p = \frac{USL - LSL}{UCL - LCL} \quad \left\{ \begin{array}{l} USL \text{ e } LSL = \text{limites de controle especificados (voz do cliente)} \\ UCL \text{ e } LCL = \text{limites de controle do processo (voz do processo)} \end{array} \right.$$

Fonte: Barcellos (2014).

Contudo, o protótipo separa o modo como a geração dos gráficos e a classificação do processo são realizados. A geração dos gráficos está associada a um software e a um item, enquanto que a classificação do processo, como o nome sugere, considera todos os valores coletados para cada item, independente de software.

Após finalizada a classificação, os resultados podem ser visualizados tanto pelo usuário administrador, quanto pelo operador, como é mostrado na Figura 27.

Figura 27 - Classificação dos itens



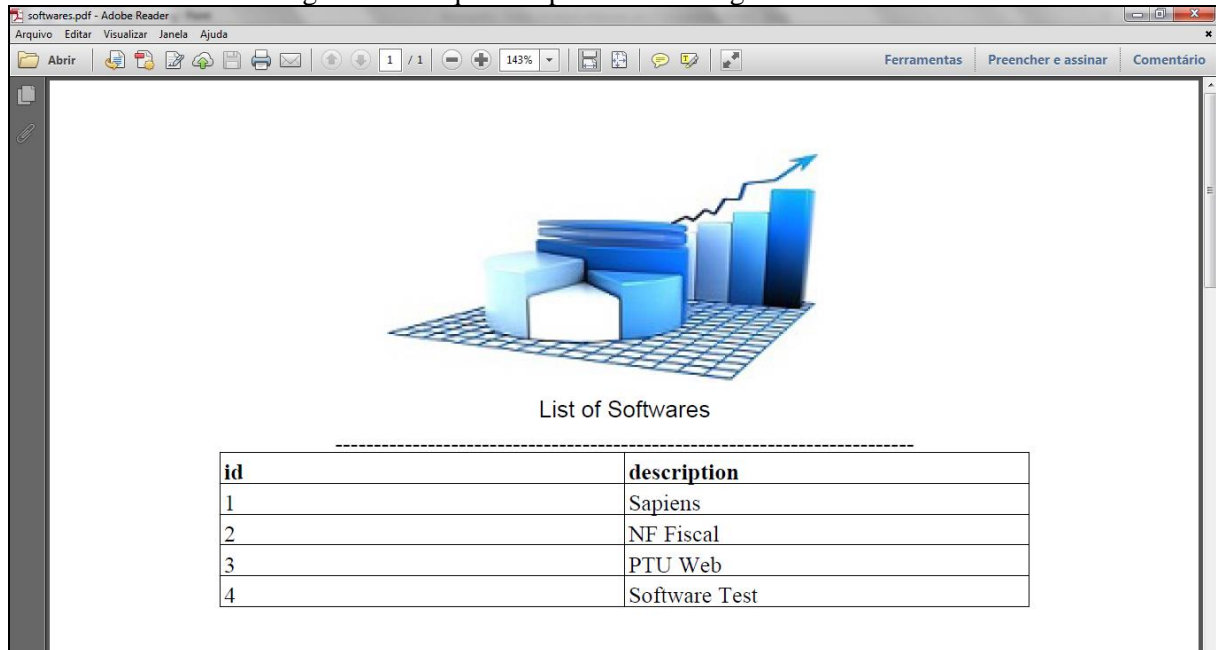
Classification of Items			
(1 of 1)			
id	description	stable	able
1	Changing requirements	true	true
2	Errors on system tests	false	false
3	Code compilation errors	false	false
4	Errors reported by user	false	false
5	Supports requested	false	false
6	New features	false	false

Na Figura 27 é mostrada a classificação de todos os itens (ou métricas) cadastradas no protótipo, indicando sua capacidade e estabilidade. Desta forma, o comportamento dos itens pode ser avaliado dentro do processo e as tomadas de decisão podem ser determinadas. Conforme exibido, apenas o item *Changing requirements* encontra-se estável e capaz.

De forma geral, para os demais itens, o usuário administrador (ou o responsável pelo processo) poderá identificar duas situações possíveis quanto a classificação obtida. Para os itens que encontram-se estáveis, porém incapazes, é necessária a adaptação do processo para que os requisitos estabelecidos pelo cliente sejam atendidos. Outra situação seria para os itens que encontram-se instáveis e incapazes. Nesse caso é necessária a identificação e a remoção das causas especiais de variação do processo para estabilizá-lo.

Por fim, o usuário administrador pode realizar a exportação dos dados distribuídos em planilhas, nos formatos: pdf e xls. Além da classificação dos itens, o administrador tem a possibilidade de exportar a listagem de todos os valores associados aos itens e aos softwares cadastrados. Como exemplo, a Figura 28 apresenta a listagem de softwares cadastrados no protótipo exportada no formato pdf. Desta forma, os dados podem ser disponibilizados para outras pessoas, como os demais membros de uma equipe, por exemplo.

Figura 28 - Arquivo exportado da listagem dos softwares



3.4 RESULTADOS E DISCUSSÕES

O protótipo criado mostrou-se transparente quanto à sua utilização. Mesmo que todos os cálculos e gráficos oferecidos sejam os mais indicados para os processos de desenvolvimento de software, o protótipo pode ser utilizado para as informações sobre qualquer área de aplicação. Mesmo que a aplicação do CEP seja primitiva em diversas áreas do setor industrial, quando aplicado ao desenvolvimento de software torna-se um assunto sobre o qual ainda há muito a ser discutido.

Para o administrador criar um processo de avaliação, ele deve cadastrar a fase do processo, os itens ou as métricas a serem observadas no processo e o software a ser avaliado. Após esses passos ele deve notificar o(s) usuário(s) operador(es) para que o processo de coleta possa ser iniciado.

O operador poderá iniciar a coleta dos valores, cujo tempo pode ser indeterminado. Para isso ele deverá primeiro recuperar os valores e associá-los a um item e a um software. No tocante às coletas, a partir da primeira coleta o administrador já poderá obter valores sobre o desempenho do processo. Caberá a ele notificar o operador quando o processo de coleta deverá ser finalizado.

Concluído o processo de coleta através da notificação fornecida pelo operador ou pela análise do próprio usuário administrador, este último poderá visualizar os gráficos de controle fornecidos. Finalizando o processo, a sua classificação de acordo com sua estabilidade e capacidade estará disponível para visualização a todos os usuários do sistema.

Com relação aos trabalhos correlatos, esses apresentam soluções para a avaliação e/ou armazenamento de métricas de softwares através de modelos que utilizam como referência as metodologias baseadas na área de medição de software e no CEP. Estas soluções foram estudadas para comparar e verificar o desenvolvimento do protótipo de avaliação.

O modelo CEP-S desenvolvido por Fonseca (2010) propõe o uso do CEP para os processos de elicitação de requisitos e implementação sobre o desenvolvimento de software. O trabalho de Batista (2005) teve como objetivo desenvolver um programa de medição e uma ferramenta de apoio denominada Vigia voltados para organizações de alto nível de maturidade do MR-MPS e do CMMI, visando apoiar o processo de implantação, aplicação, coleta e validação e análise dos dados. Bogoni (2007) desenvolveu um método para extração, organização e apresentação de métricas para Processo de Desenvolvimento de Software baseada em um ambiente de *Data Warehouse*.

O Quadro 15 apresenta as principais diferenças entre os trabalhos correlatos citados e o trabalho desenvolvido. Como principal ponto positivo deste trabalho têm-se a aplicação dos cálculos do CEP através de uma ferramenta simplificada, ainda com a vantagem de recuperar dados históricos de uma outra ferramenta (o SonarQube). Contudo, a funcionalidade para propor medidas às tomadas de decisão após a avaliação dos itens e dos softwares agregaria muito ao trabalho desenvolvido e, desta forma, pode ser tratado como uma extensão para a continuidade deste.

Quadro 15 - Comparativo entre os trabalhos

FUNCIONALIDADES	TCC	Fonseca	Batista	Bogoni
Aplica o CEP	Sim	Sim	Sim	Não
Fornece uma ferramenta para a aplicação	Sim	Não	Sim	Não
Aplicado a todos os níveis de maturidade	Sim	Sim	Não	Sim
Propõe uma nova metodologia de aplicação	Não	Sim	Sim	Sim
Permite a recuperação de dados passados	Sim	Não	Não	Sim
Propõe medidas para tomadas de decisão	Não	Sim	Sim	Não
Integrado com alguma ferramenta de medição e avaliação de qualidade de software	Sim, SonarQube	Não	Não	Não

4 CONCLUSÕES

Após o estudo realizado sobre o CEP, verificou-se que toda a complexidade de todo seu processo justifica a criação de uma ferramenta de auxílio. Partindo do princípio que o CEP, na área de desenvolvimento de software, trata-se de algo ainda muito novo e pouco explorado, a avaliação da aplicação com base em modelos estatísticos torna-se muito útil à empresas que buscam ter uma visão de como estão os seus processos, assim como tomar medidas para melhorá-los.

Este trabalho apresentou o desenvolvimento de um protótipo que teve como principal objetivo o auxílio do processo de avaliação de um software utilizando os modelos estatísticos do CEP, proporcionando a visualização de gráficos de controle e a classificação do processo de acordo com sua estabilidade e capacidade em tempo real. O protótipo visa eliminar o uso de processos trabalhosos e primitivos em outros softwares como planilhas e editores de texto. A partir dos dados gerados pelo protótipo, um usuário poderá notificar outros membros responsáveis de uma equipe e, a partir disso, determinar medidas a serem aplicadas sobre um determinado processo da empresa.

Após o estudo e compreensão do *framework* Demoiselle, esse foi de grande utilidade no desenvolvimento do protótipo. A grande diversidade de materiais e componentes e a possibilidade da geração de *templates* oferecidos foi uma grande facilitadora na criação da arquitetura MVC do projeto web. Outro grande facilitador para a criação dos componentes visuais do protótipo – a camada de apresentação foi – o *framework* Primefaces que, através de um ambiente web demonstrativo, permite o estudo e a melhor compreensão dos seus componentes, além de ser totalmente compatível com o Demoiselle por fazer uso da tecnologia JSF.

Tomando como base os trabalhos correlatos estudados, pode-se afirmar que o protótipo criado neste trabalho diferencia-se em vários aspectos. Todos os trabalhos propõem uma metodologia de avaliação própria. A principal contribuição e diferencial deste trabalho com relação aos demais trabalhos correlatos trata-se da disponibilização de uma ferramenta de avaliação que usa os métodos estatísticos do CEP, já comprovados em outras áreas da indústria, para ser aplicado em empresas desenvolvedoras de software, independentemente do seu nível de maturidade segundo o CMMI e MR-MPS.

4.1 EXTENSÕES

Como sugestões de extensões para a continuidade do presente trabalho em trabalhos futuros têm-se:

- a) disponibilizar um maior número de gráficos de controle pertencentes ao CEP;
- b) permitir a integração com outras ferramentas para recuperar informações de projetos passados, como por exemplo o Mantis BugTracker;
- c) disponibilizar um maior número de testes de estabilidade sobre os processos;
- d) disponibilizar um mecanismo para a criação de *baselines* de desempenho para os processos estáveis para descrever o comportamento esperado dos mesmos;
- e) disponibilizar um mecanismo para a exportação dos gráficos de controle;
- f) disponibilizar um mecanismo para a tomada de medidas de controle a serem aplicadas sobre os itens e os softwares avaliados que não estão de acordo.

REFERÊNCIAS

- BARCELLOS, Monalessa P.; ROCHA, Ana R. C.; SOUZA, Gleison S. **Medição de software e controle estatístico de processos**. 8ª ed. Brasília, DF: [s.ed.], 2012.
- BARCELLOS, Monalessa P. **Medição de software e controle estatístico de processos: a caminho da alta maturidade**. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 13., 2014, Blumenau. **Anais eletrônicos...** Blumenau: FURB, 2014. p. 161-221. Disponível em: <<http://sbqs2014.inf.furb.br/>>. Acesso em: 04 jun. 2015.
- BATISTA, Gabriela F. **Programa de medição para organizações de alta maturidade**. 2005. 116 f. Dissertação (Mestrado em Engenharia de Computação) - Departamento de Engenharia de Computação e Automação Industrial, Universidade Estadual de Campinas, Campinas.
- BOGONI, Leandro P. **Um método evolutivo para aplicação de programas de métricas em processos de desenvolvimento de software**. 2007. 79 f. Dissertação (Mestrado em Ciência da Computação) – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.
- FLORAC, Willian A.; CARLETON, Anita D. **Measuring the software process: statistical process control for software process improvement**. Indianapolis: Pearson Education, 1999.
- FONSECA, Patrícia C. **Modelo para controle estatístico de desenvolvimento de software**. 2010. 144 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Ciências Exatas, Universidade Federal de Minas Gerais, Belo Horizonte.
- FRANCISCANI, Juliana F.; PESTILI, Ligia C. CMMI e MPS.BR: um estudo comparativo. **Revista Rumos**. [Minas Gerais], v. 6, n. 3, 2012. Disponível em: <<http://www.unicerp.edu.br/index.php/pesquisas/revista-rumos>>. Acesso em: 04 jun. 2015.
- GRESSE VON WANGENHEIM, Christiane; VON WANGENHEIM, Aldo; LINO, Juliana I. **Medição de software: guia prático**. [Florianópolis]: Bookess, 2012. Disponível em: <<http://www.researchgate.net/publication/>>. Acesso em: 08 jul. 2015.
- MARETTO, Ciro X. **Uma arquitetura de referência para medição de software**. 2013. 190 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Federal do Espírito Santo, Espírito Santo.
- RIBEIRO, José L. D.; TEN CATEN, Carla S. **Controle Estatístico do Processo**. 2012. 172 f. Dissertação (Mestrado em Engenharia de Produção) - Curso de Pós-graduação em Engenharia de Produção, Universidade Federal do Rio Grande do Sul, Porto Alegre.
- SONARQUBE GROUP. **SonarQube demonstration**. [S.I.], [2014?]. Disponível em: <<http://nemo.sonarqube.org/>>. Acesso em: 17 set. 2014.

APÊNDICE A – Principais casos de uso

Este apêndice apresenta os dois principais casos de uso presentes no diagrama da Figura 5, trata-se dos seguintes: UC04 – Cadastrar item, UC13 – Recuperar informações do SonarQube e UC14 – Realizar cálculos CEP.

No Quadro 16 é descrito com detalhes o caso de uso UC04, o qual representa a definição de uma nova métrica a ser observada.

Quadro 16 - Descrição do UC04

UC04 - Cadastrar item	
Descrição	Permite cadastrar um item ou uma métrica a ser avaliada que, posteriormente será associada à um ou mais softwares cadastrados pelo administrador.
Pré-condição	O usuário logado deve possuir o papel de administrador.
Cenário principal	<ol style="list-style-type: none"> 1) O administrador escolhe a descrição do item; 2) O administrador escolhe o limite máximo de controle esperado; 3) O administrador escolhe o limite mínimo de controle esperado; 4) O administrador escolhe a fase do processo a qual o item pertence; 5) O administrador solicita o registro do item; 6) O protótipo registra o item com as informações informadas nos passos anteriores pelo administrador; 7) O protótipo retorna para a listagem dos itens.
Exceção 01	No passo 5 se alguma das informações a serem definidas pelo administrador nos passos anteriores não estiver preenchida é gerada uma mensagem informando que todos os campos devem ser preenchidos.
Pós condição	Foi criado um item a ser avaliado pelo protótipo, tornando-se disponível para ser vinculado aos softwares e as coletas a serem cadastradas.

O caso de uso UC13 é descrito com mais detalhes no Quadro 17, representando o processo de integração do protótipo com a ferramenta SonarQube.

Quadro 17 - Descrição do UC13

UC13 – Recuperar informações do SonarQube	
Descrição	Permite o usuário administrador recuperar as informações de processos passados sobre uma base de informações da ferramenta SonarQube.
Pré-condição	O usuário logado deve possuir o papel de administrador e a base de dados do SonarQube deve estar localizada no mesmo banco de dados do protótipo.
Cenário principal	<ol style="list-style-type: none"> 1) O administrador acessa a tela de listagem dos itens; 2) O administrador solicita que o processo de integração seja iniciado; 3) O protótipo verifica se as fases do processo fornecidas pelo SonarQube já existem, caso contrário elas serão criadas; 4) O protótipo verifica se os softwares fornecidos pelo SonarQube já existem, caso contrário eles serão criados; 5) O protótipo verifica se os itens fornecidos pelo SonarQube já existem, caso contrário eles serão criados; 6) O protótipo insere as informações das coletas fornecidas pelo SonarQube; 7) O protótipo exibe uma mensagem informando ao administrador que os limites inferior e superior de cada item recuperado deverá ser informado manualmente; 8) O protótipo retorna para a listagem dos itens.
Exceção 01	No passo 2, caso não haja uma base de dados do SonarQube uma mensagem de erro será apresentada ao administrador.
Exceção 02	No passo 6, caso o processo de integração for executado mais de uma vez as coletas serão inseridas novamente, e assim, duplicadas.
Pós condição	Todos os softwares, itens e coletas armazenados pela Ferramenta SonarQube que sejam dos domínios esperados foram inseridos na base de dados do protótipo.

O caso de uso UC14, descrito detalhadamente no Quadro 18, representa o processo de realização dos cálculos estatísticos do CEP.

Quadro 18 - Descrição do UC14

UC14 – Realizar cálculos CEP	
Descrição	Permite o usuário administrador solicitar a realização dos cálculos estatístico do CEP.
Pré-condição	O usuário logado deve possuir o papel de administrador e pelo menos um software e um item devem ser selecionados.
Cenário principal	<ol style="list-style-type: none"> 1) O protótipo recupera todas as coletas realizadas até o momento; 2) O protótipo realiza os cálculos; 3) Insere o resultado na base de dados.
Exceção 01	No passo 6, caso não haja nenhuma coleta feita sobre o software e o item selecionado, será gerado um erro no processo de realização dos cálculos.
Pós condição	Os cálculos foram inseridos na base de dados do protótipo.

APÊNDICE B – Dicionário de dados do MER

Neste apêndice é apresentado o dicionário de dados do modelo entidade relacionamento conforme exibido no Quadro 19.

Quadro 19 - Dicionário de dados

KEY	CAMPO	DESCRIÇÃO	TIPO	TAM.
Tabela: calculation		Tabela de cálculos		
PK	calculation_id	Identificador	BIGINT	20
FK	item	Chave de ligação com o item	BIGINT	20
FK	software	Chave de ligação com o software	BIGINT	20
	calculation_type	Tipo do cálculo	INT	11
	ucl	Limite de controle superior	DECIMAL	19,2
	cl	Limite de controle	DECIMAL	19,2
	lcl	Limite de controle inferior	DECIMAL	19,2
Tabela: collect		Tabela de coletas		
PK	collect_id	Identificador	BIGINT	20
FK	item	Chave de ligação com o item	BIGINT	20
FK	software	Chave de ligação com o software	BIGINT	20
	start_date	Data de início	DATE	
	value	Valor	DECIMAL	19,2
Tabela: item		Tabela de itens ou métricas		
PK	item_id	Identificador	BIGINT	20
FK	process	Chave de ligação com a fase do processo	BIGINT	20
	description	Descrição	VARCHAR	100
	able	Capacidade	CHAR	1
	stable	Estabilidade	CHAR	1
	usl	Limite superior especificado	DECIMAL	19,2
	isl	Limite inferior especificado	DECIMAL	19,2
Tabela: software		Tabela de softwares		
PK	software_id	Identificador	BIGINT	20
	description	Descrição	VARCHAR	100
Tabela: software_item		Tabela de ligação entre softwares e itens		
FK	software_id	Chave de ligação com o software	BIGINT	20
FK	item_id	Chave de ligação com o item	BIGINT	20
Tabela: user		Tabela de usuários		
PK	user_id	Identificador	BIGINT	20
	login	Login	VARCHAR	50
	name	Nome	VARCHAR	100
	email	Email	VARCHAR	100
	paper	Papel de acesso	VARCHAR	50
	password	Senha	VARCHAR	50
Tabela: process		Tabela de fases do processo		
PK	process_id	Identificador	BIGINT	20
	description	Descrição	VARCHAR	100