Universidade Federal de Santa Catarina

Ferramenta para contagem de linhas de código e suporte à análise gerencial de projeto

Bruno Saibro Silveira

Trabalho de Conclusão do Curso de Sistemas de Informação

Orientador(a): Roberto Silvino da Cunha

Florianópolis, 2014.

Bruno Saibro Silveira

Ferramenta para contagem de linhas e suporte à análise gerencial de projeto

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel, do curso de Sistemas de Informação da Universidade Federal de Santa Catarina

Orientador(a): Roberto Silvino da Cunha

Florianópolis, 2014

Errata

Folha	Linha	Onde se lê	Leia-se

Bruno Saibro Silveira

Ferramenta para contagem de linhas e suporte à analise gerencial de projeto

Este trabalho de conclusão de curso foi julgado e aprovado para obtenção do grau de Bacharel em Sistemas de Informação da Universidade Federal de Santa Catarina

_____

Roberto Silvino da Cunha, M Sc.

Orientador

_____

Prof. Ricardo Pereira e Silva, D Sc.

Co Orientador

_____

Leandro José Komosinski, D Sc.

_____

André Fabiano Dyck, M Sc.

Dedico este trabalho às pessoas importantes para mim.

Agradecimentos

Agradeço aos pais a quem tudo devo, pelo apoio durante a duração do curso e por toda a minha vida. Agradeço à namorada pelo amor e apoio. Agradeço ao orientador Roberto, pela ajuda durante o desenvolvimento do trabalho e também por tudo o que aprendi no decorrer do desenvolvimento.

Resumo

Obter métricas a partir do código de um software é parte importante no processo de desenvolvimento de software. O planejamento de um projeto é uma das atividades fundamentais no gerenciamento de software, ele será realizado de maneira mais precisa dispondo de métricas de produtividade. Para a medição é desejável que seja realizado através de ferramental adequado para oferecer subsídio na hora de fazer a análise e o planejamento, fornecendo dados precisos.

Este trabalho, tem como objetivo o desenvolvimento da concepção e implementação de um plugin para o Redmine que integre outros plugins para fornecer dados objetivos derivados da contagem de linhas de código, que possam ser analisados.

Relatórios podem ser gerados para dar suporte à análise do gestor do projeto quanto ao andamento do mesmo. Fazer análise de produtividade e o planejamento de cronograma. As métricas de produtividade ainda serão usadas para a determinação de duração das inspeções de código, que também será útil na precisão do planejamento do projeto.

Palavras-Chave: Redmine, Ruby, Rails, Medição e Análise, Engenharia de Software, Contagem de Linhas

Abstract

Gather metrics from the code of a software is an important part in the Software Development Process. Planning the project is one of the fundamental activities in software management, it can be done more accurately with productivity metrics. For software measurement is desirable that it is done through proper allowance to offer time to do the analysis and planning by providing accurate data tooling.

This work aims at the development of the design and implementation of a plugin for Redmine that integrates other plugins to provide objective data derived from counting lines of code, which can be analyzed.

Reports can be generated to support the analysis of the project manager as to the progress of the project. Making productivity analysis and planning schedule. The productivity metrics will still be used for the determination of duration of code inspections, which will also be useful for precise planning of the project.

Palavras-Chave: Redmine, Ruby, Rails, Software Measurement, Software Engineering, Line Counting

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CMMI – Capability Maturity Model Integration

SEI – Software Engineering Institute

Sumário

1 Introdução

1.1 Contextualização

Há uma demanda muito grande para desenvolvimento de software. Atualmente o mercado exige um produto cada vez mais complexo, que seja desenvolvido com qualidade, seja barato e cada vez mais cumprindo prazos menores. Para a organização se adequar à demanda do mercado, é de grande valia adotar modelos de maturidade, padrões e metodologias.

Porém, segundo SEI (2006, p. 3) as abordagens existentes focam em uma parte específica do negócio, não utilizando uma abordagem sistêmica para todos os problemas enfrentados pelas organizações.

O CMMI (Capability Maturity Model Integration) é um modelo para melhoria no processo de software desenvolvido pelo SEI (Software Engineering Institute) da Universidade Carnegie Mellon amplamente utilizado, que reúne práticas que são necessárias para atingir maturidade no processo de software.

> O CMMI® (Capability Maturity Model® Integration) oferece uma oportunidade de evitar ou eliminar essas barreiras e compartimentalizações por meio de modelos integrados que transcendem as disciplinas. O CMMI para Desenvolvimento consiste das melhores práticas relativas às atividades de desenvolvimento e manutenção aplicadas a produtos e serviços. Ele abrange práticas que cobrem o ciclo de vida do produto desde a concepção até a entrega e manutenção, e se concentra no trabalho necessário para construção e manutenção do produto em sua totalidade. (SEI 2006)

Uma das áreas de processo do CMMI-DEV, no nível de maturidade 2 é Medição e Análise. Esta área de processo serve para dar suporte ao desenvolvimento de software, fornecendo informações que auxiliam à tomada de decisão. Ela ajuda a saber, por exemplo, qual será a duração de um projeto, seu custo entre outros. Além disso, fornece subsídios para a realização de um planejamento mais preciso.

Medição e Análise envolve coleta de dados quantitativos e qualitativos tanto sobre os produtos de trabalho do projeto quanto do processo utilizado no seu desenvolvimento. Com base nos dados coletados, análises podem ser produzidas, fazendo com que ajustes possam ser realizados, medidos e comparados com o que foi feito antes dos ajustes. Dessa forma, diversos ciclos de medição e análise podem ser realizados até que os objetivos de qualidade sejam atingidos.

Assim, a medição não é um fim em si, mas um meio para vários fins, tal como a gerência de projetos, o controle e melhoria de processo e a garantia de qualidade, entre outros. (WANGENHEIM; WANGENHEIM; LINO, 2012, p.10).

Nesse contexto apresentado, se insere a contagem de linhas. A contagem de linhas em um arquivo de código é uma das métricas necessárias para a área de medição e análise poder atuar e permite ao gerente realizar análises de produtividade, planejamento de cronograma e estimativas com precisão. Além da simples contagem de linhas, uma análise do arquivo pode ser feita, identificando padrões de nomenclatura de variáveis e métodos, quantidades de comentários no código que podem gerar documentação, entre outras.

A contagem de linhas também torna possível prever uma duração de inspeção de código.

Segundo Fagan (1979), a inspeção é um método formal, eficiente e econômico de se encontrar defeitos em código. Em uma inspeção, um defeito é qualquer parte de um produto de trabalho que impede o inspetor de aprová-lo. E quanto antes um defeito é encontrado, menor é o custo para resolvê-lo.

Seu objetivo é que todos os inspetores cheguem em um consenso sobre um trabalho ou produto e aprove sua utilização em um projeto. (WIKIPÉDIA, 2013)

A equipe que faz a inspeção de acordo com o processo de inspção desenvolvido por Fagan em meados de 1970 cita 4 papéis:

1. **Moderador:** O líder da inspeção. Ele planeja e coordena a inspeção.
2. **Autor:** Quem produziu o trabalho a ser inspecionado.
3. **Designer:** Quem fez o planejamento e requisitos do software.
4. **Inspetor:** Escreve e/ou executa casos de teste para encontrar defeitos.

A inspeção de código aumenta a produtividade e melhora a qualidade final do software (Fagan, 1976, p. 205).

Sendo assim, uma ferramenta para fazer contagem de linhas e auxiliar o processo de inspeção, será de importante utilidade para a equipe de inspeção.

1.2 Problema

Determinar como está o andamento de um projeto sem ter métricas para auxiliar a análise pode ser um trabalho difícil. Estipular prazos corretamente e cumpri-los é importante.

Conhecer a produtividade da equipe de desenvolvimento pode trazer diversos benefícios para diversas áreas do processo, trazendo a possibilidade do acompanhamento no desenvolvimento com resultados objetivos.

Com as novas métricas disponíveis em uma ferramenta integradora, proposta nesse trabalho, dentro de uma sistema de Controle de Alterações (REDMINE) traria precisão no planejamento e informações objetivas sobre a produtividade do projeto.

1.3 Hipótese de pesquisa

O presente trabalho visa contribuir em diversas áreas do desenvolvimento de software, como na Gestão de Projeto, auxiliando na precisão dos cronogramas, no acompanhamento do andamento do desenvolvimento. Na Gestão da Qualidade medindo a produtividade da equipe e acompanhando a produtividade, tornando objetiva a avaliação da alteração no processo. Na Gestão da Configuração, organizando e facilitando o acesso às informações de versionamento de cada tarefa. No desenvolvimento do projeto, organizando e facilitando o acesso às informações de alterações no mesmo.

Baseado na questão em estudo, a hipótese formulada é: Se for implementada a contagem de linhas de código, em conjunto com a ferramenta de Inspeção (FARIAS; DOS SANTOS, 2013) e com ferramenta de versionamento, então serão disponibilizadas informações objetivas a diversas áreas de processo para que possam acompanhar, e automatizar a coleta de dados nessas áreas do processo de desenvolvimento.

1.4 Objetivos

1.4.1 Objetivo Geral

O objetivo geral do presente trabalho é o desenvolvimento de um plugin para o Redmine que integre outros plugins, para fornecer dados objetivos derivados da contagem de linhas de código, que possam ser analisados.

1.4.2 Objetivos Específicos

- Estudar a fundamentação teórica sobre Engenharia de Software, Processos de Desenvolvimento de Software e Medição e Análise;
- Analisar o estado da arte com relação a ferramentas que fazem esse tipo de contagem e permite esse tipo de análise;
- Escolher e aplicar uma abordagem para medição
- Planejar e implementar interface para usuário;
- Fazer a integração entre o Redmine e o GIT, para relacionar conceitos do Redmine, como projetos e tarefas à *branches* e repositórios;
- Realizar o levantamento e classificação das métricas que serão utilizadas.
- Planejar e implementar a realimentação dos dados obtidos na coleta de dados da contagem de linhas de código, para o planejamento de novas atividades.
- Realizar a interpretação sobre as métricas levantadas de acordo com o modelo escolhido na abordagem para medição.

1.5 Metodologia

No presente trabalho, é realizado uma pesquisa para fundamentação teórica sobre CMMI, Análise e Medição de software e a concepção e desenvolvimento em si da ferramenta. A metodologia de desenvolvimento está dividida em quatro etapas:

Etapa 1: O estudo da fundamentação teórica é feito através de leitura de livros e artigos relevantes na área. Considerando principalmente material sobre medição e análise de software, bem como processo de desenvolvimento.

Etapa 2: A segunda etapa é a prototipação da ferramenta, desenvolvimento dos requisitos, construção de telas e definição de funcionalidades. Também é escolhida uma abordagem para medição que foi utilizada no trabalho.

Etapa 3: A terceira etapa é o desenvolvimento da ferramenta em si, a etapa onde o código é gerado e a ferramenta é concebida. O código é versionado utilizando o Git e é hospedado no Github.

Etapa 4: A última etapa é um estudo de caso aplicando a utilização da ferramenta, coletando dados para a realimentação dos dados para novas atividades e por fim a interpretação dos resultados coletados e realimentados.

2 Fundamentação Teórica

2.1 CMMI

O CCMI (Capability Maturity Model Integration) é um modelo desenvolvido pelo SEI (Software Engineering Institute) com o apoio do Departamento de Defesa dos Estados Unidos da América com práticas genéricas ou específicas, que são necessárias para atingir maturidade nas determinadas disciplinas:

1. Systems Engineering (SE);
2. Software Engineering (SW);
3. Integrated Product and Process Development (IPPD).

O modelo tem como objetivo estabelecer – com base em estudos, históricos e conhecimento operacional – um conjunto de “melhores práticas” que devem ser utilizadas para um fim específico. (WIKIPÉDIA, 2013).

O CMMI contempla duas representações: Contínua e Por Estágios. Cada uma delas utilizam diferentes caminhos para a melhoria do processo.

2.1.1 Representação Contínua

A Representação Contínua permite que uma organização utilize a ordem de melhoria que atende da melhor forma os objetivos de negócio da empresa.

> A representação contínua oferece máxima flexibilidade na utilização de um modelo CMMI para melhoria de processo. Uma organização pode focar na melhoria do desempenho de um ponto problemático associado a um processo isolado, ou pode trabalhar em várias áreas que estejam fortemente ligadas aos objetivos estratégicos da organização. A representação contínua também permite que uma organização melhore diferentes processos com diferentes ênfases ao longo do tempo. Existem algumas limitações nas escolhas de uma organização devido a dependências entre algumas áreas de processo. (SEI 2006, p. 10)

Segundo SEI (2006), se a organização tem conhecimento dos seus processos que precisam de melhorias e se as dependências entre as áreas de processo descritas no CMMI são bem compreendidas é uma boa escolha utilizar a representação contínua.

2.1.2 Representação Por Estágios

A representação por estágios oferece uma forma sistemática e estruturada para abordar a melhoria de processo, baseada em modelo, enfocando um estágio por vez. (SEI 2006, p. 10)

A cada avanço de estágios é estabelecida uma infraestrutura que serve como base para o próximo estágio.

A representação por estágios é organizada em níveis de maturidade, cada nível de maturidade contempla uma quantidade de áreas de processo, definindo dessa forma um caminho de melhoria contínua, incremental e duradoura.

A figura 1, mostra os 5 níveis de maturidade e suas características:

Figura 1: Níveis de maturidade do CMMI – Fonte (ISDBrasil 2013)

Para começar do zero ou caso não se saiba por onde começar a representação por estágios é uma boa opção.

2.1.3 Áreas de processo

As áreas de processo são práticas que ao serem implementadas em conjunto, ajudam a concretizar metas e realizar melhorias. Cada nível de maturidade engloba um conjunto dessas áreas.

A figura 2 a seguir mostra as áreas de processo e seus respectivos níveis de maturidade:

	Nível 2 (7)	Nível 3 (11)	Nível 4 (2)	Nível 5 (2)
Gestão de Processos (5) [*Organização*]		**F**ocos nos processos da *organização* **Definição** dos processos da *organização* + IPPD **T**reinamento na *organização*	**Desempenho** dos processos da *organização*	**I**mplantação de **inovações** na *organização*
Gestão de [*Projetos*] (6)	**P**lanejamento de *projeto* **M**onitoramento e controle de *projeto* **G**estão de Contrato com fornecedores	**G**estão **integrada** de *projeto* + IPPD **G**estão de riscos	**G**estão **quantitativa** de *projeto*	
Engenharia (6) [*Requisitos*]	**G**estão de *requisitos*	**D**esenvolvimento de *requisitos* **S**olução técnica **Integração** de produto **V**erificação **V**alidação		
Suporte (5) [*Análise*]	**M**edição e *análise* **G**arantia de qualidade de processo e produto **G**estão de configuração	***A**nálise* e tomada de decisões		***A**nálise* e resolução de **causas**

Figura 2 – Níveis de maturidade e suas áreas de processo – Bruno Maroto (2013)

2.2 Medição e Análise

A medição pode ajudar a levantar informações sobre o projeto que poderão ser utilizadas pela equipe gestora na tomada de decisão, para poder agir no redirecionamento da equipe em tempo hábil no intuito de manter os objetivos do projeto.

Medição ajuda, por exemplo, a saber:

- Qual a duração de um projeto?
- Qual o custo de desenvolvimento?
- Quanto já foi desenvolvido?
- Quanto foi gasto de esforço em correções?

“Fornecendo informações para responder estas questões, a medição pode oferecer dados quantitativos e qualitativos para que, sistematicamente, os projetos sejam gerenciados e os processos e produtos de software sejam controlados.“ (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 10)

2.2.1 Conceitos fundamentais

Segundo Roberts (1979), a medição, de modo geral, visa capturar características de entidades e de seus relacionamentos para manipulá-los de uma maneira formal. É um processo onde valores são atribuídos das entidades do mundo real como uma forma de descrevê-los de acordo com regras claramente definidas.

Segundo Wangenheim, et al. (2012) medidas de software são usadas para quantificar atributos do processo, projeto ou produto de software. Por exemplo, pode-se utilizar medidas para analisar a produtividade, o esforço gasto em uma tarefa, a confiabilidade do sistema desenvolvido, etc.

Abaixo são detalhados os termos envolvidos na definição de medição:

- **Medição:** A **medição** é o processo pelo qual números ou símbolos são atribuídos aos **atributos** de **entidades** no mundo real, descrevendo-os de acordo com regras claramente definidas (FENTON; 1994).
- **Entidade**: Pode ser: (WANGENHEIM, C.; et al., LINO; J., 2012, p. 15)
    - Um produto (qualquer artefato produzido ou modificado durante o processo de software, como, por exemplo, a especificação de requisitos, código-fonte, relatório de testes, etc.).
    - Um processo (qualquer processo/atividade no processo de software, como, por exemplo, programação, teste de software, etc.) ou
    - Um recurso utilizado no processo de software.
- **Atributo:** (WANGENHEIM, C.; et al., 2012, p. 15) É uma característica ou propriedade da entidade, como, por exemplo, o tamanho ou funcionalidade (de uma especificação), ou o custo ou duração (de uma atividade).
- **Medida:** É uma variável a qual é atribuído um valor como resultado de medição (ISO07).

A Tabela 1 apresentada em (WANGENHEIM, C.; et al., 2012, p. 16), apresenta exemplos de entidades, atributos e medidas

Entidade	***Atributo***	***Medida***
Projeto	Duração	Data início, data fim
	Esforço	Homens-hora por atividade
Processo	Capacidade	Nível de capacidade
Produto	Tamanho	SLOC, PCU, PF
	Confiabilidade	N de defeitos

Tabela 1: Exemplos de entidades, atributos e medidas

Um **modelo de informação de medição** é uma estrutura que mapeia as necessidades de informações, as entidades e os atributos de interesse (ISO, 2007). Um modelo de informação baseado na norma ISO/IEC 15939 (ISO, 2007) identifica conceitos básicos que fazem parte de um modelo de informação de medição (Figura 3). (WANGENHEIM; C., et al., 2012, p.18)

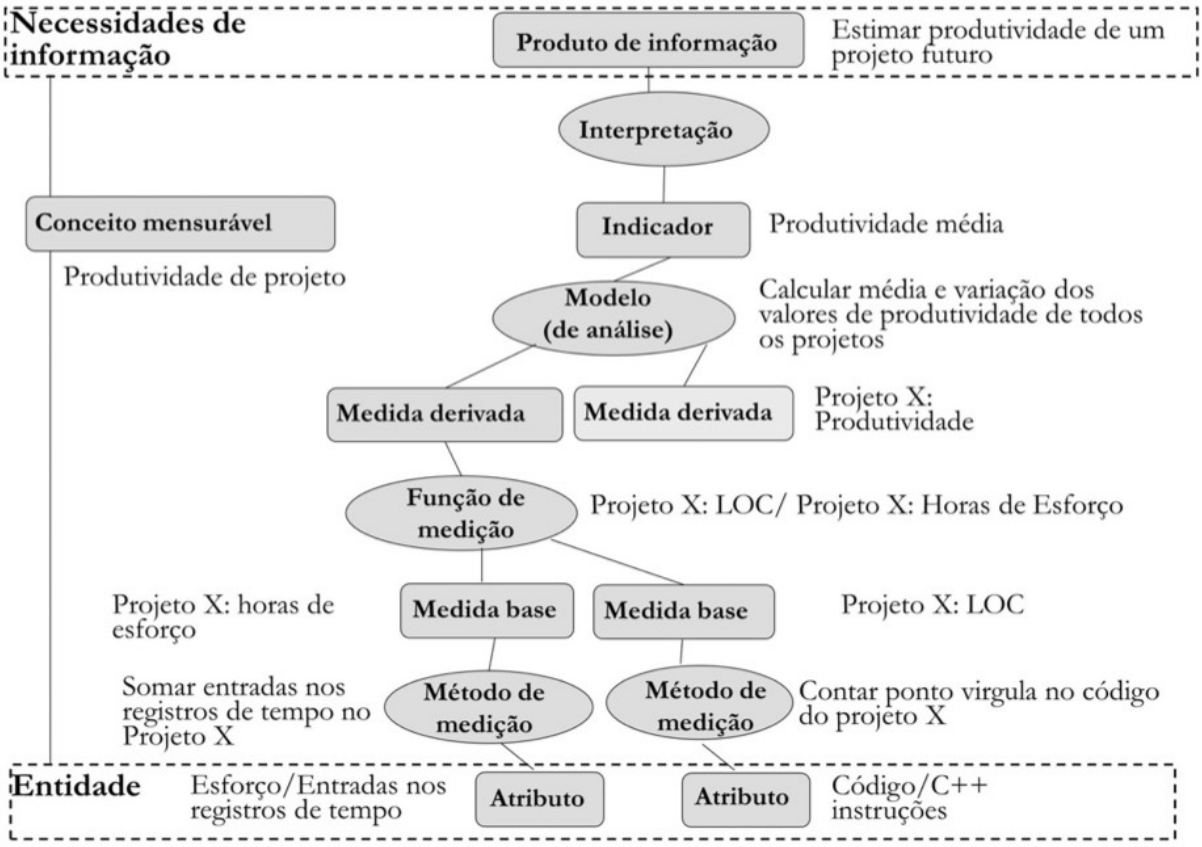


Figura 3: Modelo de informação de medição (ISO, 2007)

Um **conceito mensurável** é uma relação abstrata entre atributos de entidades e necessidades de informação. Por exemplo, uma necessidade de informação pode ser a necessidade de comparar a produtividade de uma equipe de projeto com uma taxa alvo. O conceito mensurável neste caso é a taxa de produtividade de desenvolvimento de software. (WANGENHEIM; C., et al., 2012, p. 18)

Medidas-base são definidas em termo de um atributo e um método para quantificá-lo. Uma medida base é funcionalmente independente de outras medidas e captura informações sobre um único atributo. (WANGENHEIM; C., et al., 2012, p. 19)

O **método de medição** é uma sequência lógica de operações utilizadas para quantificar um atributo no que diz respeito a uma escala especificada. Métodos de medição podem ser (ISO, 2007):

- Subjetivos: quantificação envolvendo julgamento humano
- Objetivos: quantificação com base em regras numéricas que podem ser implementadas por meios humanos ou automatizados

A **escala** é um conjunto ordenado de valores, contínuos ou discretos, ou um conjunto de categorias à qual o atributo é mapeado. O tipo da escala depende da natureza da relação entre

os valores da escala. Quatro tipos de escala são comumente definidos (ISO, 2007): nominal, ordinal, intervalar e proporcional, às vezes também chamada escala ratio. O método de medição normalmente afeta o tipo de escala que pode ser utilizada com confiabilidade dentro de um atributo específico. Por exemplo, métodos subjetivos de medição suportam somente escalas nominais e ordinais.

Nominal	Os valores de medição são categorias sem implicação de ordem entre as mesmas. Por exemplo, a classificação de defeitos por tipo.
Ordinal	Os valores de medição são pontuações ordenadas. Por exemplo, a atribuição de defeitos a um nível de criticidade.
Intervalar	Os valores de medição têm distâncias iguais correspondendo a quantidades iguais do atributo. Por exemplo, a complexidade ciclomática tem um valor mínimo de 1, mas cada incremento representa um caminho adicional. Complexidade ciclomática é uma métrica de software usada para indicar a complexidade de um programa de computador. Mede a quantidade de caminhos de execução independentes a partir de um código fonte. O valor de zero não é possível.
Proporcional	Os valores de medição têm distâncias iguais correspondendo a quantidades iguais do atributo onde o valor de zero corresponde a nada do atributo. Por exemplo, o tamanho de um módulo de software em termos de LOC é uma escala proporcional porque o valor de zero corresponde a nenhuma linha de código e cada incremento adicional representa a quantidade igual de código.

Tabela 2: Tipos comuns de escala (ISO, 2007)

Muitas vezes uma unidade é associada com uma escala. Uma **unidade de medição** é uma quantidade específica, definida e adotada por convenção com quais outras quantidades do mesmo tipo são comparadas para expressar a sua magnitude relativa a esta quantidade. Exemplos são horas, semanas, U$ dólar, etc.

Uma **medida derivada** é definida como uma função de dois ou mais valores de medidas-base. Esta função de medição é um algoritmo ou cálculo realizado para combinar duas ou mais

medidas base. A escala e unidade de uma medida derivada dependem das escalas e unidades das medidas base da qual é composta e da forma como são combinadas por meio da função.

Um **indicador** é uma medida que fornece uma estimativa ou avaliação de atributos especificados e derivados de um modelo com relação às necessidades de informação. Os indicadores são a base para análise e tomada de decisão. O modelo (de análise) é um algoritmo ou cálculo combinando uma ou mais medidas base e/ou derivadas com critérios de decisão associados. Estes critérios de decisão são limites ou alvos utilizados para determinar a necessidade para ação ou investigação ou para descrever o nível de confiança para um resultado dado. A escala e método de medição também afetam a escolha de técnicas de análise ou modelos utilizados para produzir indicadores. (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 20)

2.2.2 Abordagens de Medição

Há duas referências que descrevem como a medição pode ser realizada. Esta seção tem como objetivo apresentá-las.

2.2.2.1 Goal/Question/Metric (GQM)

O *Goal/Question/Metric* (GQM) (BASILI; V. R., WEISS; D. M., 1984) é uma abordagem sistemática para fazer integração dos objetivos aos modelos dos processos de software, produtos e perspectivas de qualidade de interesse com base nas necessidades de informação do projeto e da organização. (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 25)

Essa abordagem foi desenvolvida na Universidade de Maryland, com cooperação do *Software Engineering Laboratory* da NASA, como uma abordagem orientada a objetivos para medição de produtos e processos na engenharia de software. (BASILI; V. R., WEISS; D. M., 1984)

GQM suporta a definição *top-down* de um programa de medição e a análise e interpretação *bottom-up* dos dados coletados para atender aos objetivos de medição. Em um programa de medição GQM, a tarefa de análise é precisamente e explicitamente especificada pelo objetivo de medição. Os objetivos são definidos numa forma operacional e rastreável através de um conjunto de perguntas quantificáveis. Estas perguntas são utilizadas para extrair a informação apropriada de modelos que representam dimensões do objetivo. Este refinamento é documentado num plano GQM, o qual fornece uma justificativa para a seleção das medidas subjacentes. Os dados coletados são analisados e interpretados numa forma bottom-up no contexto do objetivo de medição e as perguntas, considerando as limitações e suposições subjacentes a cada medida. (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 26)

A estrutura hierárquica, chamada de plano GQM, inclui três níveis (BASILI; V. R., et al, 1994):

- **Nível conceitual (Objetivo):** Um objetivo é definido para um objeto, para um propósito especifico com relação a um modelo de qualidade de um ponto de vista dentro de um contexto especifico.
- **Nível operacional (Pergunta):** Um conjunto de perguntas é utilizado para definir como atender a um objetivo específico. O objetivo de medição é caracterizado por perguntas que dizem respeito ao enfoque de qualidade específico do ponto de vista dentro do contexto específico.
- **Nível quantitativo (Medida):** Um conjunto de medidas é definido para descrever o mapeamento do sistema empírico ao sistema formal no que diz respeito às perguntas.

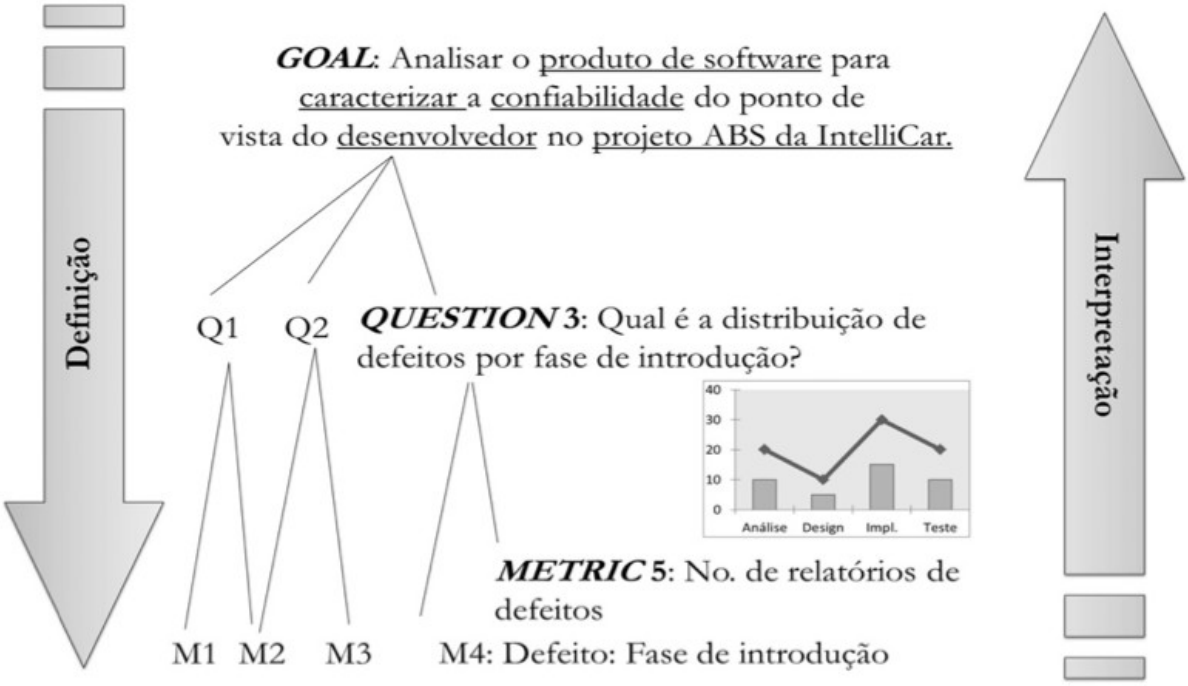


Figura 4: Modelo de informação no GQM (WANGENHEIM; C., et al, 1998)

Assim, o GQM ajuda a refinar aspectos de qualidade que são abstratos no escopo de software e na derivação de mapeamentos entre o mundo empírico e o modelo formal. GQM pode ser aplicado para medir qualquer tipo de produto, processo ou projeto de software. Ele pode visar todos os tipos de finalidades e focar em qualquer aspecto de qualidade definido a partir de qualquer perspectiva em qualquer contexto. (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 27)

O processo de medição usado pelo GQM se baseia no Paradigma de Melhoria de Qualidade (QIP – *Quality Improvement Paradigm*) (BASILI; V. R. et al, 1994), incluindo o

planejamento, execução do programa de medição e a captura de experiências, formando um ciclo, apresentado na Figura 5.

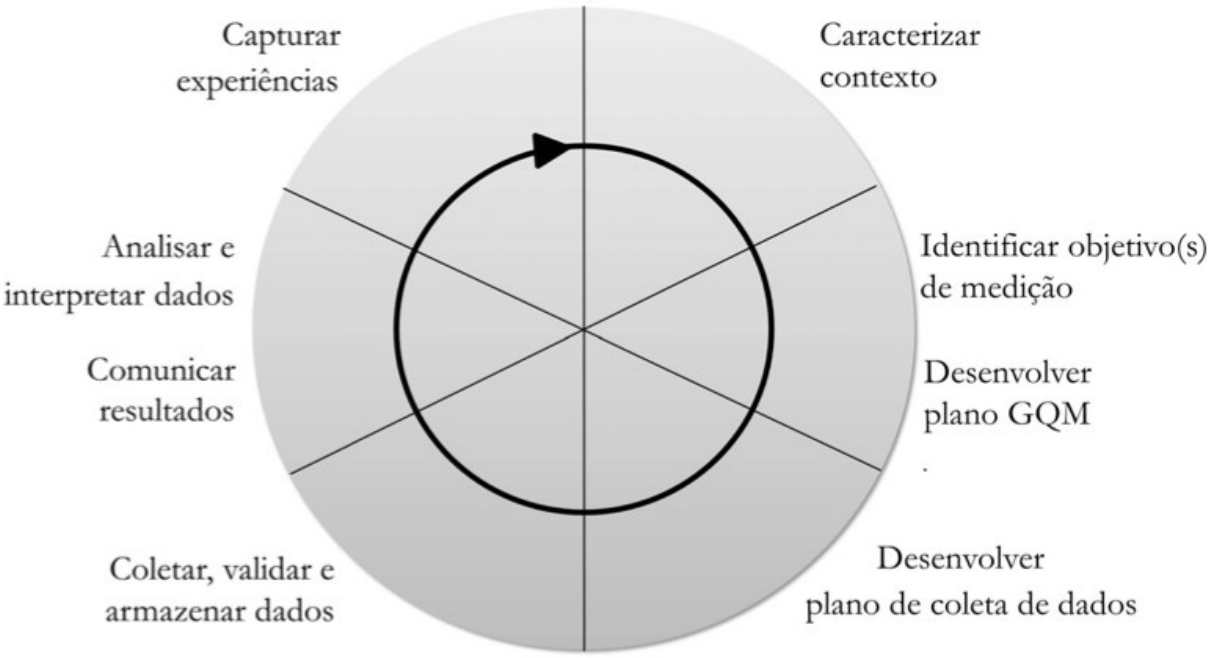


Figura 5: Ciclo do processo de medição GQM (GRESSE; C. et al, 1995)

Inicialmente, um estudo prévio é realizado para caracterizar o contexto e encontrar modelos e experiências relevantes. Com base nessa contextualização, objetivo(s) a serem atingidos pelo programa de medição são especificados, definindo precisamente:

- objeto,
- propósito,
- enfoque de qualidade,
- ponto de vista e
- contexto da medição.

A partir do objetivo, um conjunto de medidas relevantes é derivado via perguntas e modelos, sendo documentado em um plano GQM. Um plano de coleta de dados é definido pela integração das medidas definidas, determinando quando, como, e por quem os dados podem ser coletados, armazenados e verificados.

Durante a execução do programa de medição, os dados são coletados de acordo com o plano de coleta de dados. Os dados coletados são analisados e interpretados, considerando os objetivos GQM definidos no plano GQM. Ao final, os resultados da análise e interpretação e outras experiências são capturados para a sua reutilização em outros programas de medição.

Um princípio importante do GQM é o envolvimento desde o início e ativo da equipe de projeto e outros interessados, tanto na definição, quanto na execução do programa de medição. Este envolvimento ajuda em prevenir resistência contra medição porque permite que

os envolvidos participem da definição e assim também direcionem o uso dos resultados da medição. Além disto, ajuda para melhorar a validade dos resultados por ter interpretações de pessoas que são especialistas no contexto específico do objeto e no contexto que está sendo analisado. (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 27-28)

2.2.2.2 Pratical Software and System Measurement (PSM)

A abordagem PSM - Practical Software and Systems Measurement (Department of Defense, 2003) descreve um processo de medição guiado por informação que foca nos objetivos técnicos e do negócio de uma organização. A abordagem PSM foi patrocinada pelo DoD – *Department of Defense* (Departamento de Defesa)/ EUA e pelo US *Army* (Exército dos EUA) e é baseada em práticas de medição do DoD, governo dos EUA e indústria. Ela foi a base e, conseqüentemente, está compatível com a norma ISO/IEC 15939: *Software Engineering - Software Measurement Process*. O PSM fornece orientações, incluindo exemplos de indicadores e medidas, lições aprendidas, estudos de casos, etc. O PSM descreve como definir e implementar um programa de medição para suportar as necessidades de informação de organizações de aquisição e fornecimento de software. É uma abordagem que pode ser utilizada para a definição e implementação de um processo eficaz de medição para projetos de software e de sistemas. O objetivo do PSM é fornecer aos gerentes técnicos e de projetos informações quantitativas que são necessárias para a tomada de decisões que impactam no custo, no cronograma, e nos objetivos técnicos de desempenho do projeto. (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 30)

Princípios de medição

O processo de medição PSM é baseado em um conjunto de princípios de medição. Estes princípios representam as melhores práticas da medição e fazem do processo de medição PSM uma ferramenta eficaz de gerenciamento (Department of Defense, 2003):

1. Usar questões e objetivos para guiar os requisitos de medição.
2. Definir e coletar medidas baseando-se nos processos técnicos e de gerenciamento.
3. Coletar e analisar dados em um nível de detalhamento suficiente para identificar e isolar problemas.
4. Implementar uma capacidade de análise independente.
5. Usar um processo sistemático de análise para rastrear as medidas para as decisões.
6. Interpretar os resultados da medição no contexto de outra informação de projeto.

7. Integrar a medição ao processo de gerenciamento do projeto em todo o ciclo de vida.
8. Usar o processo de medição como uma base para comunicações objetivas.
9. Focar inicialmente em análises de nível de projeto.

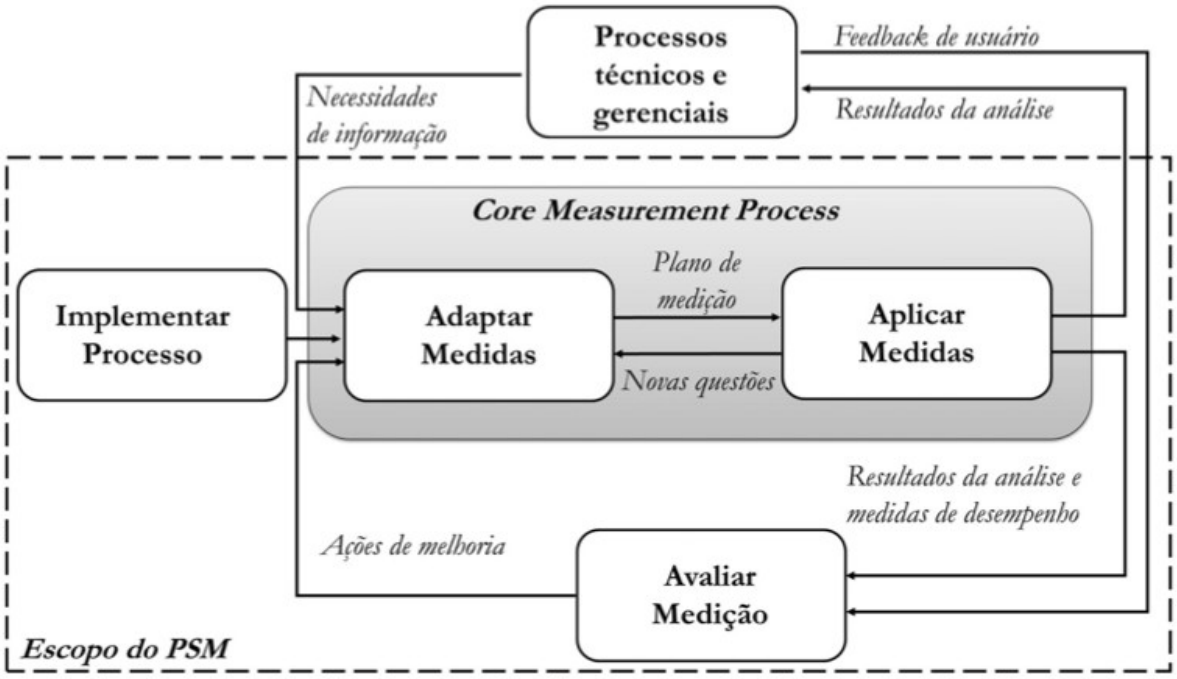


Figura 6: Processo de Medição do PSM (WANGENHEIM; C., WANGENHEIM; A., LINO; J., 2012, p. 32)

3 Estado da Arte e Especificação da Ferramenta

Para analisar o estado da arte, foi procurado em repositórios como o *Github* e área de plugins no website do *Redmine* por palavras-chave relacionadas com o tipo de ferramenta. As palavras-chave foram: *redmine, git, linhas, lines, contagem, counting e etc.* Foram feitas pesquisas utilizando combinações dessas palavras-chave sem sucesso.

Uma ferramenta para melhorar a integração com o GIT foi encontrada, porém sem nenhuma relação com Medição e Análise e inspeção de código. Servia apenas para automatizar a forma com que o Redmine atualiza os repositórios cadastrados quando há modificações. (PHLEGX SYSTEMS, 2013)

Na Universidade Federal de Santa Catarina, foi desenvolvida uma ferramenta para auxílio no processo de inspeção, que foi concebida em forma de plugin para Redmine. (FARIAS; DOS SANTOS, 2013) Porém a ferramenta não conta com a funcionalidade de contagem de linhas.

Há atualmente um número considerável de ferramentas que fazem contagem de linhas. Alguns exemplos são: LocMetrics, CCCC, USC CodeCount, USC COCOMO, CLOO, SLOCCount, SourceMonitor, LOCC e etc. (Disponível em: http://www.locmetrics.com/alternatives.html).

Essas são ferramentas stand-alone, apesar de serem completas em seu escopo, contam apenas linhas de códigos de linguagens específicas e não fazem integração com o GIT.

Algumas vantagens de se ter uma ferramenta para realizar contagem de linhas em forma de plugins para Redmine são observadas. Primeiramente torna-se possível o relacionamento de *branches* com *Issues*. O Redmine possui integração com o GIT, tornando possível o cálculo de diff entre versões de um mesmo arquivo, sendo independente de linguagens de programação. O Redmine é uma ferramenta *open-source*, com uma alta capacidade de customização através do desenvolvimento de *plugins*, possui uma comunidade grande e que participa de forma ativa ajudando desenvolvedores com dúvidas, com uma ótima documentação disponível e em constante desenvolvimento.

3.1 Redmine e Plugins

O Redmine é uma aplicação Web open-source para gerenciamento de projetos, escrito utilizando o framework Ruby on Rails. Ele permite a criação de múltiplos projetos, tem um sistema de monitoramento de tarefas flexível, possui gráfico de Gantt e campos para estimativa de prazo e tempo. A ferramenta desenvolvida no presente trabalho será em forma de um plugin

para o Redmine e utilizará algumas das informações que fazem parte do conjunto básico do mesmo, como o prazo e estimativa em horas de conclusão de uma tarefa.

3.2 Especificação da Ferramenta

3.2.1 Requisitos Funcionais

A Tabela 4 enumera os requisitos funcionais da ferramenta.

Requisito
A ferramenta deverá realizar a contagem de linhas de um ramo, não contabilizando linhas em branco nem linhas removidas
Um projeto deverá associar um ou mais repositórios de código, sendo que devem ser da mesma ferramenta de versionamento.
Um *ticket* deverá ser associado a um ou mais *branches.*
O plugin deverá informar quantas linhas de código estão sendo trabalhadas no ticket.
O plugin de inspeção, com base na quantidade de código e na média de preparação para a inspeção, deverá sugerir o tempo de preparação para cada reunião.
Deverá ser adicionado as seguintes colunas à lista de colunas para filtros: Tempo estimado, Linhas estimada, Linhas Escritas, Porcentagem de horas realizadas, Porcentagem de linhas realizadas, Produtividade, Horas Projetadas e Linhas Projetadas.
A ferramenta permitirá que o usuário configure por projeto as seguintes variáveis: Média de preparação de Inspeção, Média de duração da inspeção, Média de retrabalho e Média de produtividade.
A ferramenta permitirá que o usuário configure um fator de ajuste referente ao tipo da tarefa do Redmine e à complexidade escolhida para tal tarefa.
Um campo personalizado nomeado **Complexidade** será criado, que terá os seguintes valores: **Pequena**, **Média** e **Alta**

Tabela 4: Requisitos Funcionais

3.2.2 Requisitos Não Funcionais

A Tabela 5 enumera os requisitos funcionais da ferramenta.

Requisito
Será desenvolvido em formato de plugin para o Redmine
Fará integração com o Git, utilizando a gem rugged

Tabela 5: Requisitos não funcionais

3.2.3 Diagrama Entidade-Relacionamento

O Redmine é uma ferramenta grande e complexa, portanto, possui uma grande quantidade de tabelas e relacionamentos.

Para concepção da ferramenta, algumas tabelas e relacionamentos foram criados. As tabelas são representadas por classes chamadas **Model**, que serão descritas a seguir.

A Figura 7 mostra o modelo ER com as tabelas da nova configuração do Redmine após a instalação do plugin:

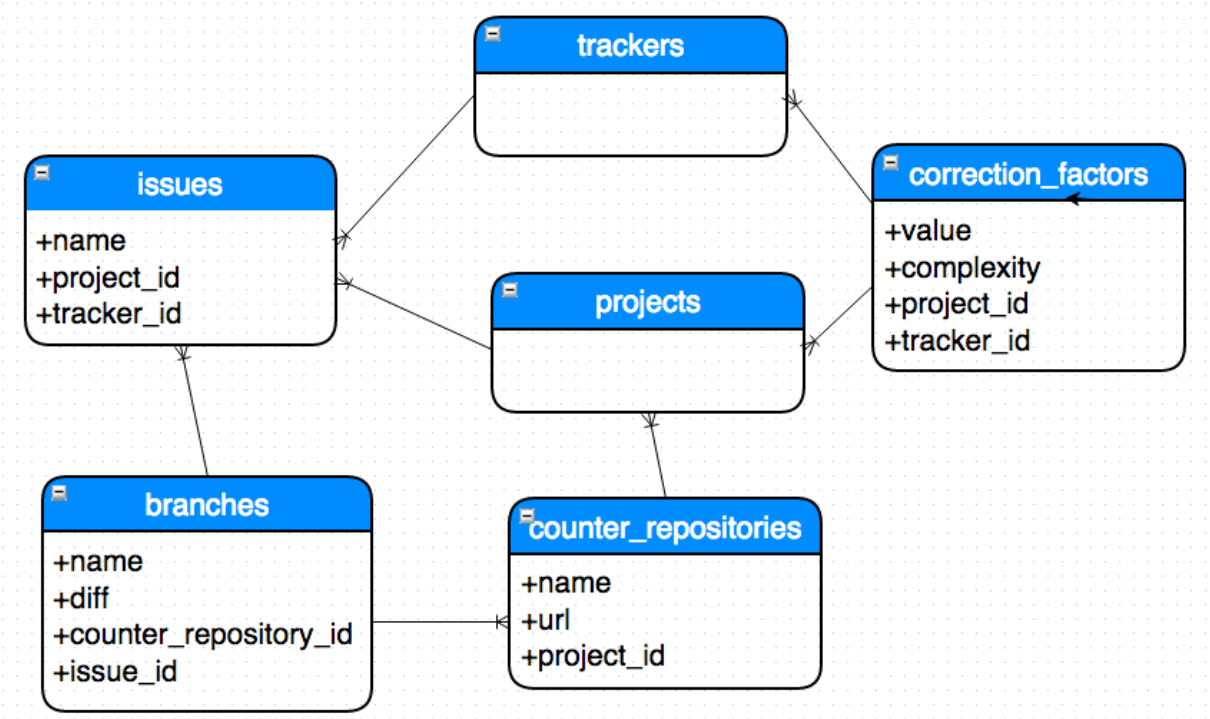


Figura 7 – Modelo ER

3.2.3.1 Counter_Repository

Representa um repositório, foi criado para possibilitar que o usuário adicione repositórios do git ao projeto do Redmine. Possui os campos:

- **name** – Nome do repositório
- **url** – Caminho para o repositório local
- **project_id** – id do projeto na tabela projects

```
require 'rugged'

class CounterRepository < ActiveRecord::Base
  unloadable

  attr_accessible :name, :url, :project

  belongs_to :project

  def branches
    repo = Rugged::Repository.new(url)
    repo.branches
  end
end
```

Figura 8 – Arquivo counter_repository.rb

3.2.3.2 Branch

Representa um branch do repositório Git. O usuário pode adicionar quantos branches quiser à tarefa.

- **name** – Nome do branch
- **counter_repository_id** – id do repositório na tabela counter_repositories
- **issue_id** – id da tarefa na tabela issues
- **diff** - diferença de linhas entre o branch e seu respectivo master no repositório

```ruby
require 'rugged'

class Branch < ActiveRecord::Base

  unloadable

  attr_accessible :name, :counter_repository, :issue, :diff

  belongs_to :issue
  belongs_to :counter_repository


  def get_diff
    count = 0
    repo = Rugged::Repository.new(counter_repository.url)
    diff = repo.diff(name, 'master')

    diff.each_patch do |patch|
      patch.hunks.each do |hunk|
        hunk.lines.each do |line|
          count = count + 1 if (line.new_lineno == -1 && !line.content.blank?)
          count = count - 1 if line.old_lineno == -1
        end
      end
    end
    count
  end

end
```

Figura 9 – Arquivo branch.rb

3.2.3.3 Correction_Factor

Representa o fator de ajuste utilizado para corrigir as estimativas geradas pela ferramenta, configurado de acordo com cada tipo de tarefa possível no projeto e a complexidade. Possui os campos **value**, **tracker_id**, **project_id** e **complexity**.

- **complexity** – valor da complexidade que é validada para aceitar apenas os valores “Low”, “Medium” e “High”
- **project_id** – id do projeto na tabela projects
- **tracker_id** – id do tipo de tarefa, na tabela trackers
- **value** – o valor do fator de ajuste

```ruby
class CorrectionFactor < ActiveRecord::Base
  unloadable
  attr_accessible :complexity, :project, :tracker, :value

  belongs_to :project
  belongs_to :tracker

  validates :complexity, inclusion: { in: %w(Low Medium High) }
end
```

Figura 10 – Arquivo correction_factor.rb

Como descrito acima, alguns relacionamentos foram criados para tornar possível o funcionamento adequado da ferramenta, a seguir será descrito as modificações necessárias para adequar o código do Redmine à levar em consideração esses relacionamentos.

Para modificar o comportamento original do Redmine, e adicionar os relacionamentos com as novas tabelas criadas, foi necessário criar Patches que sobrescrevem os arquivos para adicionar novo código.

3.2.3.4 Issue

```ruby
module IssuePatch
  def self.included(base) # :nodoc:
    base.extend(ClassMethods)
    base.send(:include, InstanceMethods)


    # Exectue this code at the class level (not instance level)
    base.class_eval do
      has_many :branches
    end
  end

  module ClassMethods
  end
  module InstanceMethods
  end
end
```

Figura 11 – Arquivo issue_patch.rb

3.2.3.5 Project

O usuário pode configurar por projeto, para cada tipo de tarefa e para cada complexidade um fator de ajuste. Para isso um relacionamento entre o novo model criado Correction_Factor e a tabela projects é necessário. O código a seguir descreve esse novo relacionamento através do método *has_many*.

```ruby
module ProjectPatch
  def self.included(base) # :nodoc:
    base.send(:include, InstanceMethods)

    # Exectue this code at the class level (not instance level)
    base.class_eval do
      has_many :correction_factors
    end
  end

  module InstanceMethods
  end
end
```

Figura 12 – Arquivo project_patch.rb

Alguns campos também foram adicionados diretamente à tabela projects, são eles:

Média de preparação de Inspeção, Média de duração da inspeção, Média de retrabalho e Média de produtividade. Para criá-los na tabela, um arquivo de migração foi gerado:

```ruby
class AddMonitoringFieldsToProject < ActiveRecord::Migration
  def change
    add_column :projects, :inspection_preparation_average, :integer
    add_column :projects, :inspection_average, :integer
    add_column :projects, :rework_average, :integer
    add_column :projects, :productivity_average, :integer
  end
end
```

Figura 13 – Arquivo 003_add_monitoring_fields_to_project.rb

3.2.4 Casos de Uso

A principal função da ferramenta é utilizar a contagem de linhas e estimativas cadastradas pelo usuário para calcular campos que serão utilizados na geração de relatórios, que aproveita a estrutura do Redmine. A seguir um diagrama de casos de uso com as funcionalidades básicas da ferramenta:

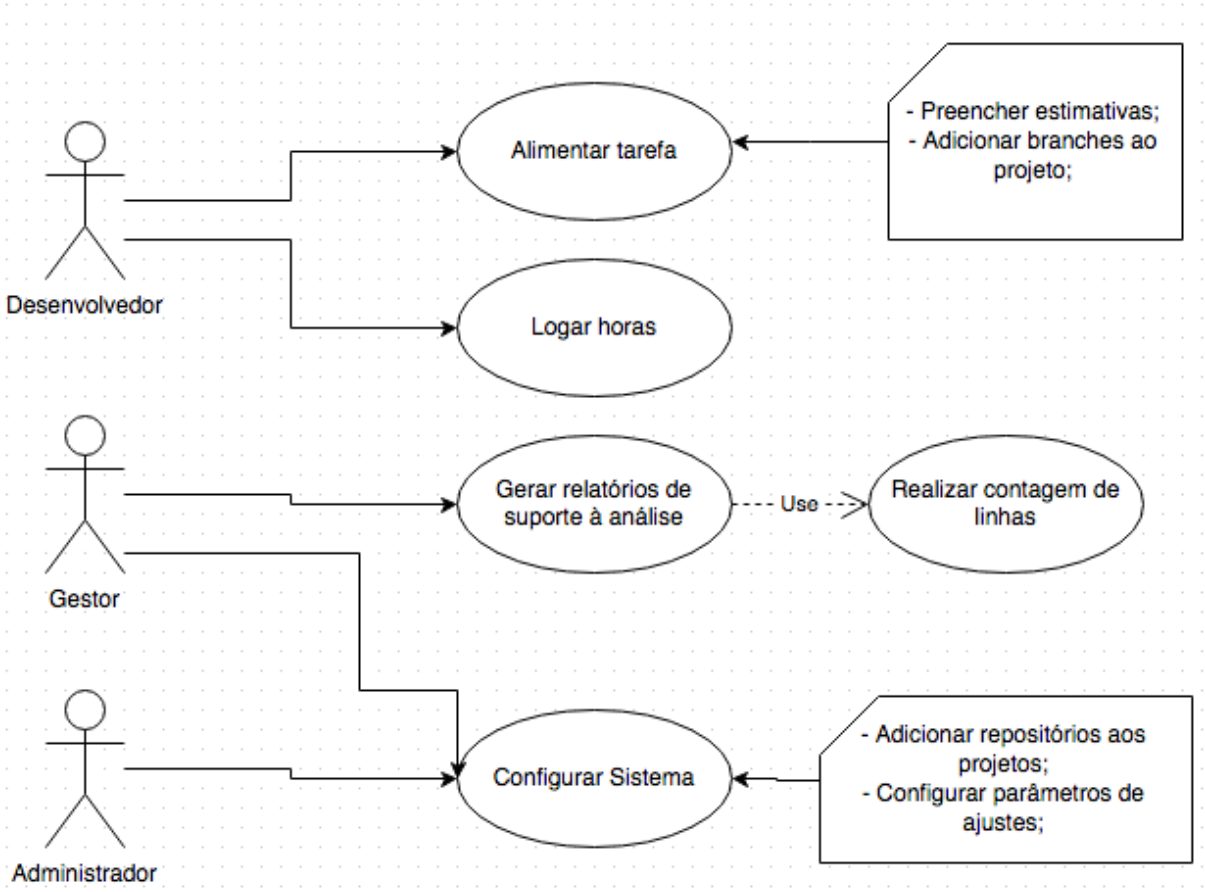


Figura 14 – Diagrama de Casos de Uso

3.3 Métricas

A abordagem escolhida para se realizar a medição foi o GQM. Ele foi escolhido pois define modelos de medição em três níveis: Conceitual (Objetivos), Operacional (Questões) e Quantitativo (Métricas). Dessa forma podemos partir de objetivos e chegar nas métricas que serão usadas na ferramenta. Com base no que é descrito na abordagem os, objetivos, questões e métricas são definidos, com isso o quadro a seguir é gerado:

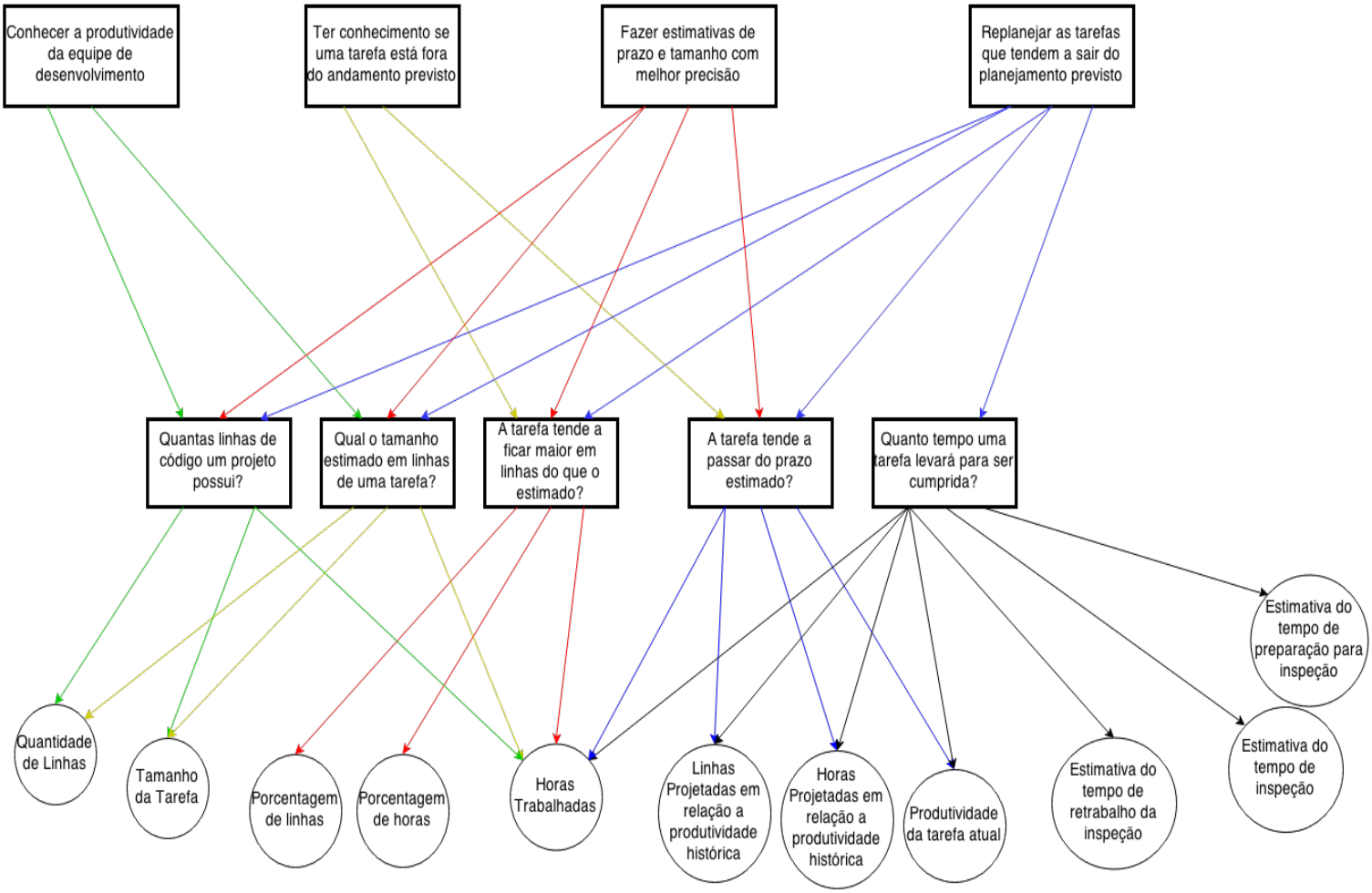


Figura 15 – Quadro com objetivos, questões e métricas

Nos capítulos a seguir serão detalhadas as métrica apresentadas no GQM do projeto e como são calculadas.

3.3.1 Linhas

Quantidade de linhas desenvolvidas em uma tarefa. Métrica necessária para ter o conhecimento do tamanho da tarefa. Definido pela soma de diffs de todos os branches cadastrados àquela tarefa.

Apenas linhas adicionadas são levadas em consideração pela ferramenta. Uma tarefa de refatoração, por exemplo, teria uma quantidade pequena de linhas apesar de ter uma complexidade maior que uma correção de defeito. Para esse tipo de situação existem os campos de Complexidade e Tipo da Tarefa, a serem cadastrados para cada tarefa. Com isso, a quantidades de linhas tem um peso diferenciado para cada tarefa.

3.3.2 Porcentagem de Linhas

Porcentagem total desenvolvida em linhas. Com essa métrica é possível saber qual é a porcentagem em relação ao estimado e o desenvolvido para uma tarefa. Ao final do

desenvolvimento, sabe-se se uma tarefa foi estimada corretamente ou não. Definido pela fórmula:

$$\frac{\text{Total de Linhas}}{\text{Linhas Estimadas}} * 100$$

3.3.3 Porcentagem de Horas

Porcentagem de horas gasta. Com essa métrica sabemos se um projeto está perto de expirar o número de horas de acordo com o estimado ou se ultrapassou o estimado e verificar se a estimativa foi correta ou incorreta. Utiliza o número de horas estimado e o número de horas registradas no sistema, como mostra a fórmula:

$$\frac{\text{Total de Horas}}{\text{Horas Estimadas}} * 100$$

3.3.4 Produtividade

Conhecer a produtividade permite que se faça uma análise detalhada em conjunto com outros fatores para que se possa melhorá-la. Calculado baseado no total de linhas desenvolvidas e o número de horas registradas no sistema.

$$\frac{\text{Total de Linhas}}{\text{Total de Horas}}$$

3.3.5 Linhas Projetadas

A quantidade de linhas projetadas de acordo com o andamento da tarefa. Calculada utilizando o total de linhas desenvolvidas e a porcentagem de conclusão registrada pelo usuário. Como base nessa métrica o gestor tem conhecimento se a tarefa ficará com uma diferença de tamanho do estimado.

$$\frac{\text{Total de Linhas} * 100}{\text{Linhas Estimadas}}$$

3.3.6 Horas Projetadas

A quantidade de horas projetadas de acordo com o andamento da tarefa. Calculada utilizando o total de horas registradas e a quantidade de horas estimada. Essa métrica permite que se tenha uma ideia do valor de horas que o projeto levará para ser executado, o que pode significar um erro de estimativa ou um atraso no prazo.

$$\frac{\text{Total de Horas}}{\text{Horas Estimadas}} * 100$$

3.3.7 Tamanho da Tarefa

Estimativa do tamanho que a tarefa terá em linhas. Calculado utilizando o total de horas estimado para a tarefa e a produtividade média.

$$\text{Horas Estimadas} * \text{Produtividade Média}$$

3.3.8 Tempo de preparação para inspeção

O tempo estimado que será levado para se fazer uma preparação para a inspeção da tarefa. Calculada utilizando o tamanho da tarefa e a média de preparação para a inspeção.

$$\frac{\text{Tamanho da Tarefa}}{\text{Média de preparação para inspeção}}$$

3.3.9 Tempo de inspeção

O tempo estimado que será levado para se fazer uma inspeção da tarefa. Calculada utilizando o tamanho da tarefa e a média de tempo para a inspeção.

$$\frac{\text{Tamanho da Tarefa}}{\text{Média para inspeção}}$$

3.3.10 Tempo de retrabalho da inspeção

O tempo estimado que será levado para se resolver os problemas descobertos em uma inspeção. Calculada utilizando o tamanho da tarefa e a média de tempo de retrabalho para a inspeção.

$$\frac{\text{Tamanho da Tarefa}}{\text{Média para retrabalho}}$$

3.4 Apresentação da Ferramenta

Para a ferramenta iniciar o seu funcionamento é necessário que algumas configurações sejam realizadas, como será apresentado no capítulo a seguir.

Em seguida será apresentado o seu funcionamento.

3.4.1 Configuração

A configuração da ferramenta é iniciada pela ligação do projeto, definido no Redmine, com um repositório local do GIT, que é um Sistema de Versionamento de Código (http://git-scm.com/).

Após a especificação do repositório os ramos (*branches*) que farão parte da codificação dessa tarefa deverão ser declarados.

Finalmente a configuração estará completa quando os parâmetros para o projeto, isto é, para todas as novas tarefas envolvidas no grupo de um projeto do Redmine, forem definidos.

3.4.1.1 Adicionando Repositórios

No projeto será realizado o cadastro dos repositórios locais, através da aba configurações (Figura 16), que poderão ser utilizados nas tarefas, como pode ser visto na Figura 17.

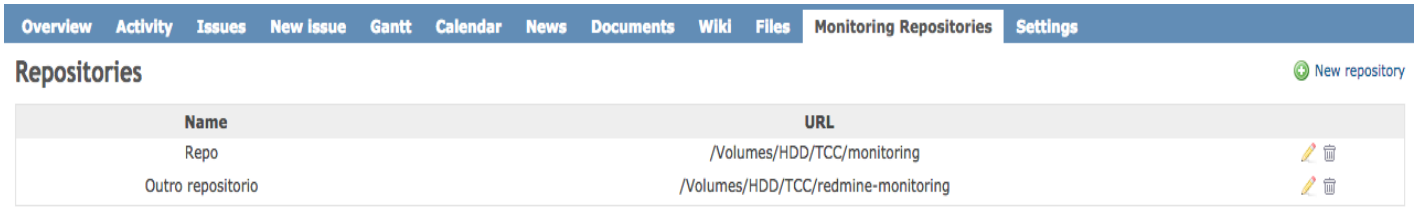


Figura 16 – Destaque da aba para adicionar repositórios

O usuário insere o nome do repositório e sua localização, que deve ser o caminho para a pasta do repositório local. A ferramenta não aceita repositórios remotos.

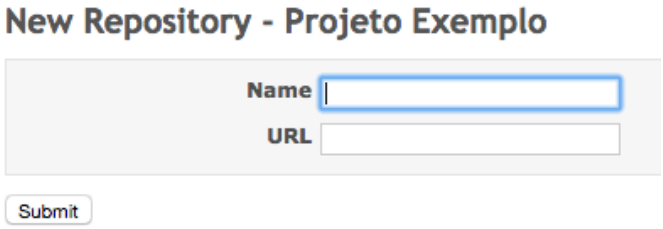


Figura 17 - Cadastro de repositórios do projeto.

O motivo de ser possível cadastrar mais de um repositório em um projeto do Redmine é a real necessidade de um projeto possuir diversos repositórios. Um exemplo disso pode ser a utilização de ferramentas auxiliares (plugins) criadas por terceiros ou mesmo criadas especificamente para o projeto.

Cada ferramenta auxiliar será mantida em um repositório específico e o próprio Git se encarregará de fazer a ligação dos sub-repositórios através de “submodule”. Disponível em: <http://git-scm.com/book/en/v2/Git-Tools-Submodules> Acesso em: 15 out. 2014.

Em uma tarefa pode ser necessário alterar, além do repositório principal, também os repositórios das ferramentas, para poder atingir o objetivo da alteração solicitada na tarefa.

3.4.1.2 Adicionando Branches

Para fazer a contagem de linhas de um arquivo é necessário associar um ou mais branches à uma tarefa, dessa forma as linhas contadas são entre os arquivos dos ramos informados e seus respectivos ramos principais.

A Figura 18 destaca como os branchs são adicionados e removidos à tarefa e como são apresentados.

Figura 18 - Tela de criação da tarefa, com novo link para escolher branches.

A Figura 19 mostra o formulário de escolha de branches, separando-os por repositórios que estão em negrito e a escolha é feita através de checkboxes:

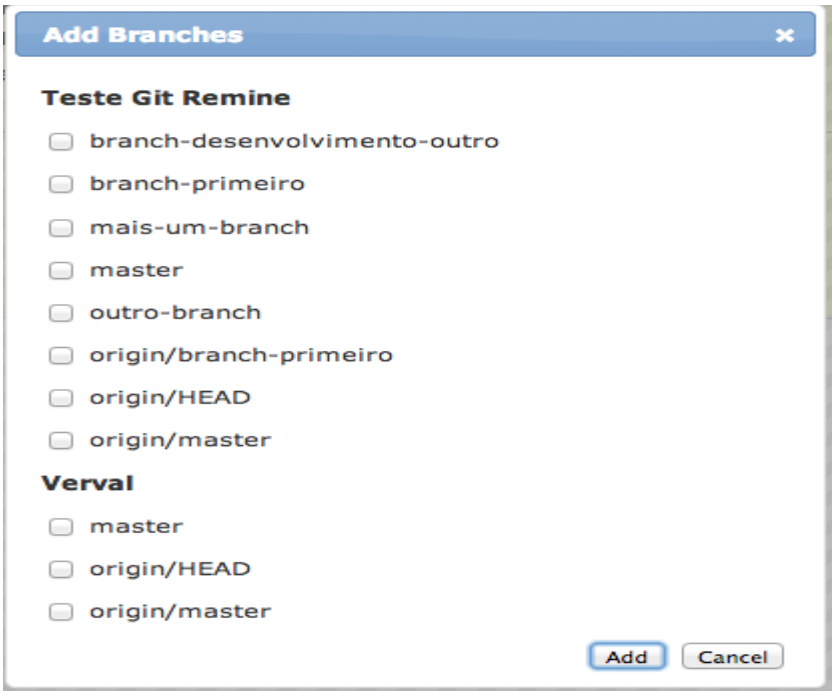


Figura 19 - Tela de escolha dos branches.

3.4.1.3 Configurando Parâmetros do Projeto

Os parâmetros utilizados nos cálculos das métricas são configuráveis por projeto, como mostra a Figura 20:

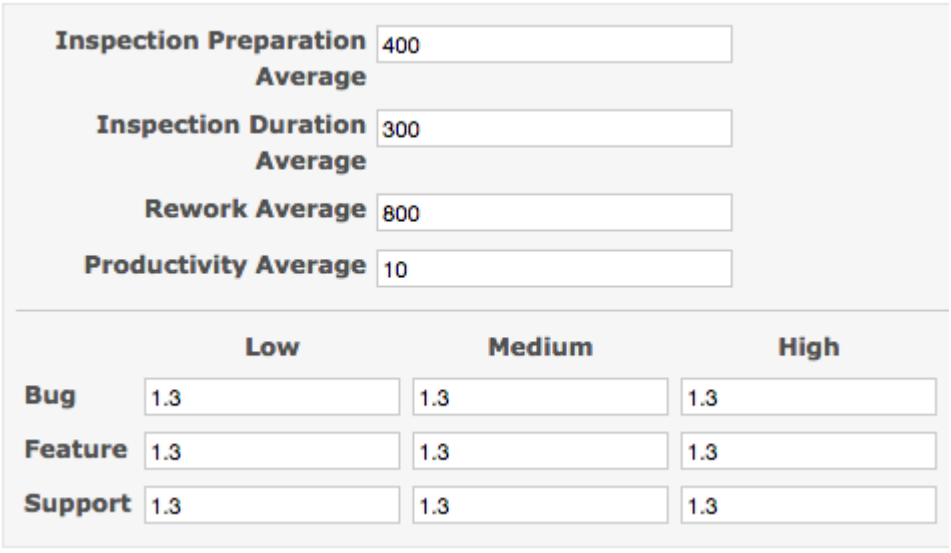


Figura 20 – Configuração de parâmetros

A tabela para configurar os fatores de correção é criada a partir de um campo customizado automaticamente criado pela ferramenta chamado **Complexidade**, com três valores (Baixa, Média e Alta) e todos os tipos de tarefas que estão criados no sistema. A produtividade de um

time de desenvolvimento está diretamente ligada a coisas como complexidade e tipo de tarefas. Uma correção de bug, por exemplo, não possui a mesma complexidade que uma nova funcionalidade em um sistema, por isso a criação de um fator de ajuste para cada tipo de tarefa.

A complexidade servirá para poder, se possível, classificar a produtividade de tarefas com as mesmas características. Dessa forma os relatórios serão agrupados por complexidade e suas respectivas produtividades.

Com o tempo os fatores de ajuste apresentarão o desvio entre o planejado e o realizado, por complexidade, nos relatórios da ferramenta. Periodicamente, conforme definido no plano de qualidade do projeto, será realizado um recadastro no fator de ajuste na ferramenta (Figura 20), com base nos relatórios apresentados sobre esse desvio, fazendo com que essa divergência entre o estimado e o realizado tenha uma tendência de ser estabilizada com o tempo.

3.4.2 Relatórios e Métricas

3.4.2.1 Métricas

A métrica básica da ferramenta é a contagem de linhas de uma tarefa. Elas são contadas com o resultado de um comando *git diff* entre os ramos atuais, definido na tarefa, e os ramos mestre dos repositórios realizado pela ferramenta. O resultado é analisado para se determinar a contagem. Linhas adicionadas em branco não são consideradas, assim como as linhas que foram excluídas durante o desenvolvimento..

```
diff --git a/Gemfile b/Gemfile
new file mode 100644
index 0000000..ea9869f
--- /dev/null
+++ b/Gemfile
@@ -0,0 +1,3 @@
+source 'https://rubygems.org'
+
+gem 'rugged', git: 'git://github.com/libgit2/rugged.git', branch: 'development', submodules: true
\ No newline at end of file
```

Figura 21 – Exemplo do resultado de um comando *git diff*

Após a inserção dos branches à tarefa, a diferença do número de linhas entre o branch e o **master** é apresentada como mostra a Figura 22:

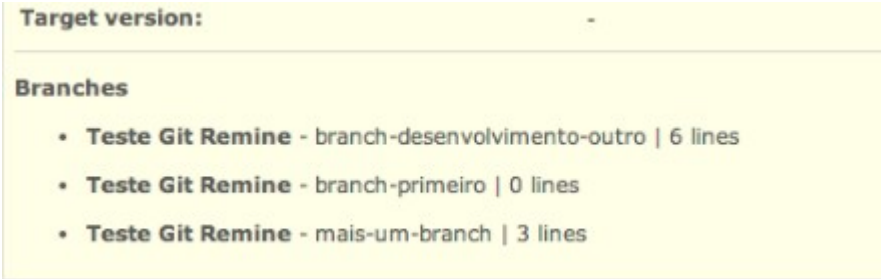


Figura 22: Apresentação dos branches e seus respectivos número de linhas

Algumas outras informações são calculadas e apresentadas na tarefa, são elas o tamanho estimado da tarefa em linhas e as estimativas de: tempo de preparação da inspeção, tempo da inspeção em si e o tempo do retrabalho, como é destacado na Figura 23:

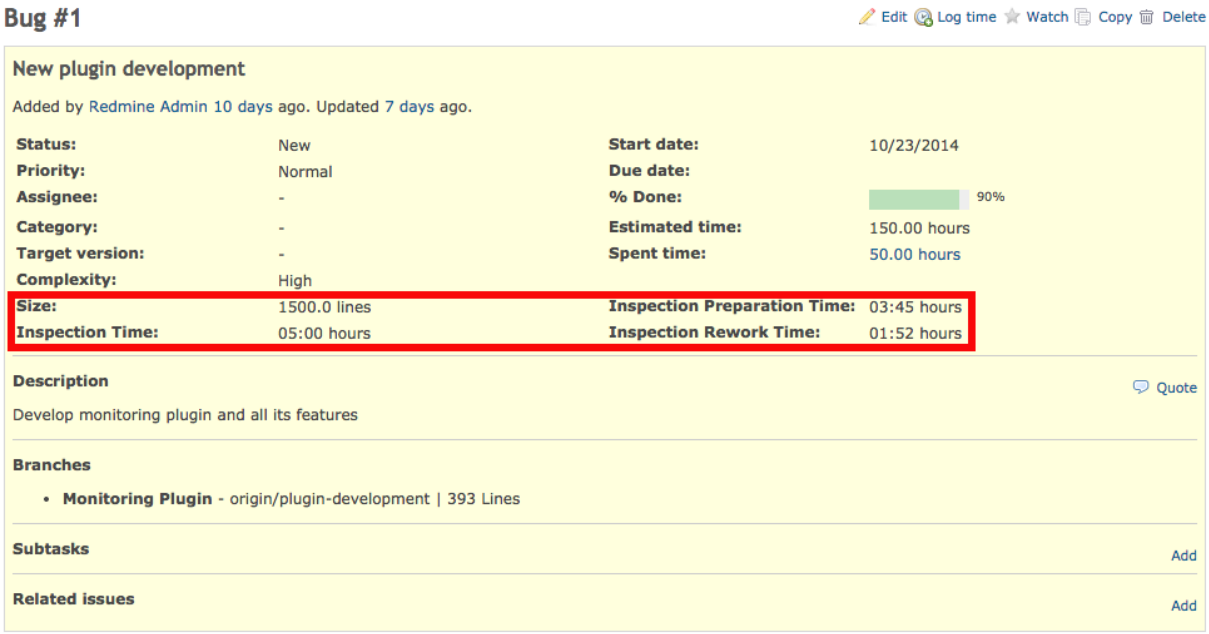


Figura 23 – Destaque das informações calculadas pela ferramenta e informadas à tarefa

Os valores calculados dos tempos de inspeção, e seus afins, são apresentados na tarefa para auxiliar na estimativa da tarefa.

O motivo de serem apresentados esses valores é a decisão, da equipe de qualidade ou do grupo gestor, de que o projeto utilizará essa metodologia de trabalho no seu processo de desenvolvimento de software. Com isso deverão ser contabilizadas as atividades de: preparação, a inspeção em si e o retrabalho após a inspeção. Como essas atividades tomam tempo devem ser contabilizadas na estimativa, por isso serão apresentadas, para o gestor verificar se a estimativa apresentada pela equipe de desenvolvimento é compatível com as informações apresentadas.

3.4.2.2 Relatórios

O Redmine possui uma funcionalidade que permite filtrar as tarefas de acordo com uma série de regras e então a lista de tarefas pode ser usada para montar relatórios.

É possível escolher para quais tipos de usuários o relatório estará visível e também se estará disponível em todos os projetos ou não. É possível também agrupar os resultados por alguma coluna. É possível filtrar os resultados de várias formas, por exemplo: por data, por

tipo de tarefa, por estado e por projeto. Basicamente todas as colunas disponíveis podem ser usadas para filtros. Organizar a lista em ordem ascendente ou descendente usando as colunas também é possível.

Para montar os relatórios é necessário escolher as colunas que irão compor o mesmo e os filtros que serão usados, como destaca a Figura 24.

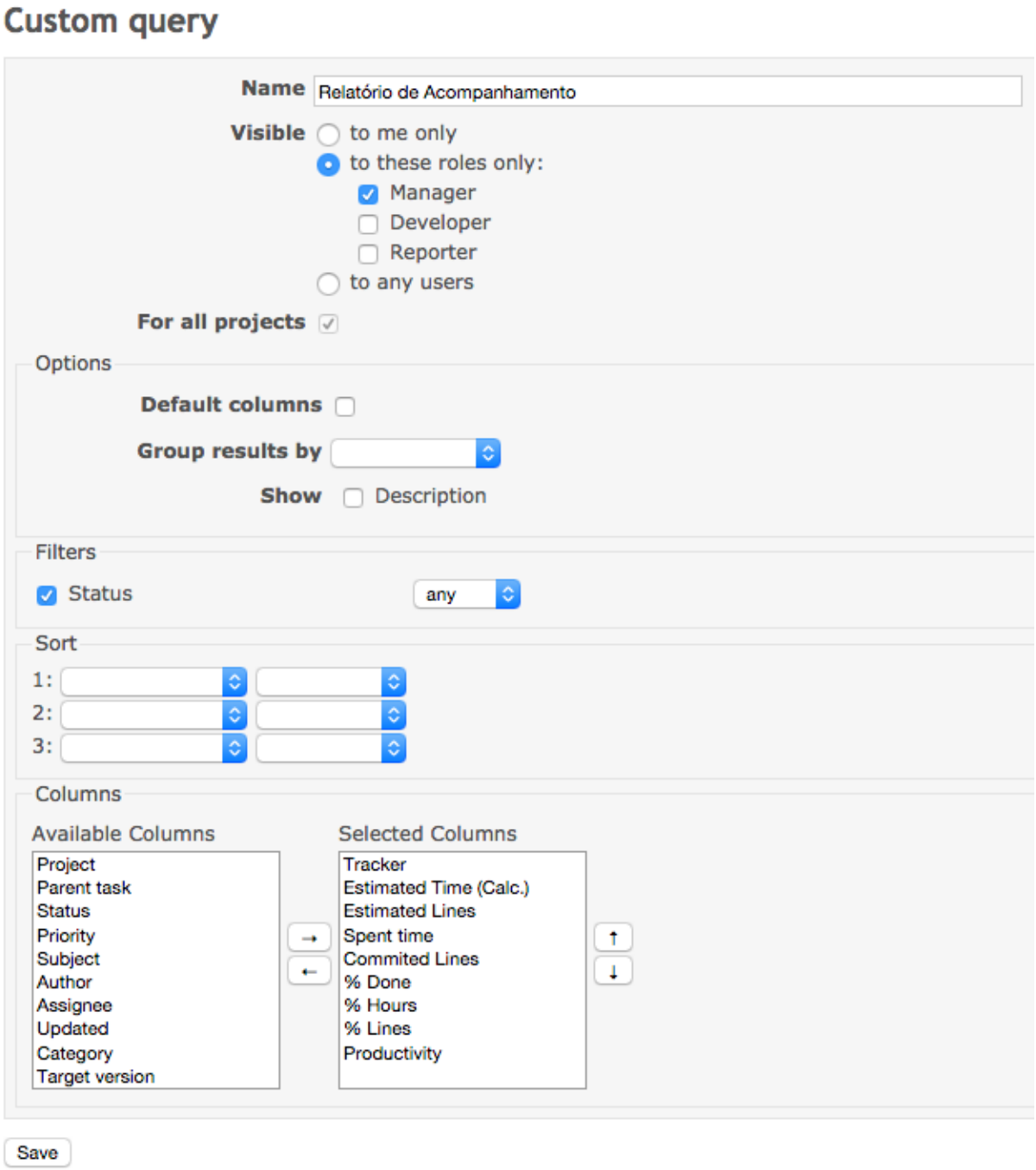


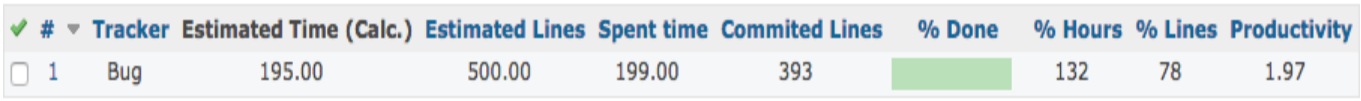
Figura 24 – Tela para criação de relatórios

A ferramenta adiciona todas as métricas descritas à estrutura de colunas para montar filtros do Redmine, permitindo que crie-se uma série de relatórios, de acordo com a

necessidade do gerente. Com o objetivo de facilitar o uso da ferramenta, três relatórios são criados automaticamente:

- **Acompanhamento**

Relatório que envolve todas as tarefas criadas no projeto. Nele há informações básicas sobre a tarefa como o tipo da mesma e o estado. Além disso, alguns dados que são preenchidos pela a equipe de desenvolvimento como a quantidade de linhas, a quantidade de horas usadas na tarefa e a porcentagem de completude da tarefa. E por fim, os dados calculados pela ferramenta como o tempo estimado calculado para o término da tarefa, a quantidade de linhas estimadas, as linhas desenvolvidas, a porcentagem de horas, a porcentagem de linhas e a produtividade. O objetivo do relatório de acompanhamento é verificar se as estimativas estão corretas, se os prazos estão sendo cumpridos, se a produtividade está de acordo e dar uma série de opções de análise e conclusão para o gestor.

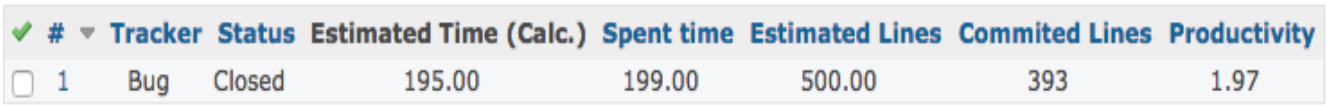


#	Tracker	Estimated Time (Calc.)	Estimated Lines	Spent time	Commited Lines	% Done	% Hours	% Lines	Productivity
1	Bug	195.00	500.00	199.00	393		132	78	1.97

Figura 25 – Relatório de acompanhamento

- **Batimento**

O relatório de batimento engloba as tarefas que já foram finalizadas, desde a data de criação do projeto. Nele estão campos básicos como o tipo da tarefa e seu estado. Dados adicionados pela equipe de desenvolvimento como a quantidade de horas gastas na tarefa e a quantidade de linhas estimadas. E para completar, os dados calculados pela ferramenta como o tempo em horas para completar a tarefa, a quantidade de linhas desenvolvidas e a produtividade. O objetivo é verificar a produtividade da equipe, podendo ser feito uma série de análises por complexidade, tipo de tarefa ou então um intervalo de data, por exemplo, e também se as estimativas estão corretas, caso não estejam uma análise mais aprofundada pode ser feita.



#	Tracker	Status	Estimated Time (Calc.)	Spent time	Estimated Lines	Commited Lines	Productivity
1	Bug	Closed	195.00	199.00	500.00	393	1.97

Figura 26 – Relatório de batimento

- **Projeção**

Envolve as tarefas em aberto. Este relatório contém os campos básicos nome do projeto e tipo da tarefa. Dados que são inseridos na tarefa pela equipe de desenvolvimento como a quantidade de linhas estimadas e a quantidade de horas gastas. E dados calculados como a quantidade de horas calculadas, as linhas desenvolvidas e a produtividade. E finalmente o que o diferencia do relatório de acompanhamento, os campos de projeção de horas e de linhas. Em tempo real é possível verificar de acordo com o valor preenchido pelo desenvolvedor quanto ao total feito para a tarefa, se a estimativa de tamanho e tempo estão corretas, fornecendo assim subsídios para o gestor chegar a uma conclusão, podendo assim entrar em contato com a equipe de desenvolvimento para verificar se alguma informação está antiga, incompleta ou incorreta e se será necessário extensão do prazo ou refazer as estimativas.

#	Tracker	Estimated Time (Calc.)	Estimated Lines	Spent time	Commited Lines	Projected Hours	Projected Lines	Productivity
9	Bug	390.00	10000.00	200.00	1490	200.00	1490.00	7.45

Figura 27 – Relatório de projeção

Todos os relatórios criados, podem ser editados conforme a vontade dos administradores e gestores, seja acrescentando e removendo colunas, adicionando filtros e etc.

4 Resultados

Com o objetivo de testar o funcionamento da ferramenta e obter resultados, dados simulados foram inseridos. Uma vez que os dados estão inseridos, é possível gerar relatórios e fazer análises.

A Figura 28 a seguir mostra o relatório de acompanhamento gerado:

#	Tracker	Estimated Time (Calc.)	Estimated Lines	Spent time	Commited Lines	% Done	% Hours	% Lines	Productivity
9	Bug	390.00	3900.00	200.00	1505		66	38	7.53
8	Bug	39.00	390.00	0	301		0	77	0
7	Bug	130.00	1300.00	50.00	1204		50	92	24.08
6	Bug	26.00	260.00	10.00	0		50	0	0
5	Bug	39.00	390.00	5.00	98		16	25	19.60
4	Bug	65.00	650.00	20.00	198		40	30	9.90
3	Bug	26.00	260.00	15.00	201		75	77	13.40

Figura 28 – Relatório de Acompanhamento

A partir dos resultados do relatório, é possível aplicar o GQM e observar algumas características.

Por exemplo, para a tarefa 3 foi estimado 260 linhas de código e o tempo estimado é de 26 horas, porém de acordo com o campo **% Done**, dado informado pela equipe de desenvolvimento, observa-se que a tarefa está em 50% e que já foram usadas 20 horas (Spent Time). Com base nisso, o gestor do projeto pode chegar à conclusão de que as estimativas foram incorretas e que a tarefa irá atrasar, podendo assim, tomar medidas necessárias para corrigir o problema, como por exemplo replanejar a tarefa ou pedir uma extensão de prazo para o cliente. Outra possibilidade para essa tarefa é que, conversando com a equipe de desenvolvimento sobre essa situação específica, o gestor tem conhecimento que a parte inicial da tarefa tomou muito tempo e que as horas restantes serão suficientes para a entrega da tarefa. De qualquer forma um alerta foi levantado para essa tarefa, que aparentemente saiu do andamento normal.

Já na tarefa 4, foi estimado 65 horas de desenvolvimento e com 650 linhas, porém em 20 horas de desenvolvimento (Spent time), foi desenvolvido 200 linhas e a tarefa está 80% completada, de acordo com o que foi colocado pelo desenvolvedor. O gestor pode concluir a partir desses dados, que a tarefa será completada antes das 65 horas estimadas, portanto antes do prazo. Um desenvolvimento completado antes do prazo pode ser resultado de uma estimativa incorreta, como também pode ser resultado de um desenvolvimento descuidado, podendo gerar um número exagerado de erros. Uma análise posterior mais detalhada, então,

poderá ser feita para se descobrir o que houve nesse caso em específico. O que pode ter ocorrido também, é um erro no preenchimento das informações por parte de equipe de desenvolvimento, inserindo assim dados incorretos quanto ao registro de horas e/ou a porcentagem completada.

A tarefa 5, se for comparado o estimado com o realizado, nota-se uma discrepância muito grande nos números. Nesse caso, o gestor poderá conversar com a equipe de desenvolvimento para descobrir o que aconteceu, como por exemplo um esquecimento por parte do desenvolvedor, que não preencheu as suas horas em algum dia. O gestor, com base nos dados do alerta pode tomar decisões, como por exemplo, fazer um replanejamento da tarefa em questão, ou então comunicar à equipe de desenvolvimento que atualize os dados no sistema, se for o caso, ou ainda solicitar que as alterações feitas até então sejam enviadas ao(s) repositório(s), para que a contagem de linhas se dê corretamente.

Na tarefa 6, observa-se que apesar de estar no relatório que o desenvolvedor registrou 10 horas no sistema, a tarefa está em 0% e nenhuma linha foi desenvolvida. Um caso desse tipo é comum indicar que houve uma etapa de análise mais longa, portanto não sendo gerado nenhum código durante a mesma. E quando o desenvolvimento de fato for iniciado a produtividade é grande, mantendo assim a estimativa correta. Ou então, novamente houve um descuido por parte da equipe de desenvolvimento que não atualizou os dados no sistema e/ou não enviou as alterações no código para o(s) repositório(s). De qualquer forma um estudo pode ser feito para descobrir o que causou o desvio da normalidade e com base nos resultados, uma decisão pode ser tomada.

Utilizando os mesmos dados, foi gerado um relatório de projeção, que calcula de acordo com os dados estimados, as projeções de linhas e horas da tarefa em questão.

#	Tracker	Estimated Time (Calc.)	Estimated Lines	Spent time	Commited Lines	Projected Hours	Projected Lines	Productivity
9	Bug	390.00	3900.00	200.00	1505	200.00	1505.00	7.53
8	Bug	39.00	390.00	0	301	0	602.00	0
7	Bug	130.00	1300.00	50.00	1204	166.67	4013.33	24.08
6	Bug	26.00	260.00	10.00	0	0	0	0
5	Bug	39.00	390.00	5.00	98	50.00	980.00	19.60
4	Bug	65.00	650.00	20.00	198	25.00	247.50	9.90
3	Bug	26.00	260.00	15.00	201	30.00	402.00	13.40

Figura 29 – Relatório de Projeção

Com base nesses dados o gestor tem uma ideia se a tarefa irá passar do prazo estimado e/ou quanto tempo ela levará para ser cumprida, podendo tomar decisões e replanejar a tarefa, se necessário.

Esses são apenas alguns exemplos de casos que podem ser detectados pelo gestor utilizando esses relatórios. Com base nos dados gerados pelo relatório o gestor terá o conhecimento da produtividade da equipe, poderá fazer estimativas de prazo e tamanho com maior precisão para tarefas futuras e poderá replanejar tarefas que tendem a sair do andamento planejado. E quanto mais tarefas, projetos, tipos de tarefas e outras informações estiverem cadastradas, maior são as possibilidades de geração de relatório e mais ricos serão os resultados.

Com o tempo de uso da ferramenta acontece um aumento no histórico da produtividade da equipe é gerado, podendo ser filtrada por datas, projetos e qualquer campo disponível no Redmine.

5 Considerações finais

O objetivo desse trabalho era projetar e implementar uma ferramenta em forma de plugin do Redmine, que através da contagem de linhas de código derivasse uma série de métricas que pudessem ser apresentadas para análise de decisão para a equipe gestora. Após a implementação da contagem de linhas, uma série de outras métricas puderam ser geradas. Essas métricas foram usadas para compor relatórios, que podem ser gerados da forma como a equipe achar melhor para analisá-los.

Analisando os relatórios, a equipe tem o conhecimento das tendências e de possíveis desvios no planejamento das tarefas, abrindo uma série de opções de decisão a serem tomadas, podendo assim, com antecedência, tomar medidas como por exemplo: replanejamento das tarefas e alinhamento com a equipe de desenvolvimento.

Conhecendo-se a produtividade da equipe de forma objetiva, como apresentada nesse trabalho, alterações no processo de desenvolvimento de software podem ser acompanhadas e comparadas. Caso haja a necessidade de inclusão de uma fase de análise, implementação e execução de testes unitários, qual o impacto dessas atividades na produtividade do desenvolvimento? Nesse caso, pode-se realizar um piloto dos testes em um conjunto de tarefas e medir a produtividade para avaliar o impacto. Em seguida é possível comparar a nova produtividade, contendo o desenvolvimento dos testes, com a produtividade antiga, sem os testes. Com base nessas informações pode-se avaliar o impacto na produtividade e analisar a inclusão ou não de uma nova fase de testes.

Após o uso contínuo da ferramenta, os parâmetros configurados poderão ser ajustados com os próprios dados apresentados nos relatórios, dessa forma as informações apresentadas nos relatórios com as novas configurações tendem a ficam cada vez mais precisos, como é o caso da própria produtividade. Depois de um período de trabalho da equipe, se for verificado que há uma divergência entre a produtividade definida como padrão e a apresentada pela equipe, pode ser alterada na configuração do sistema. Isso impacta diretamente nas projeções e estimativas das tarefas.

Possíveis trabalhos futuros envolvem a adequação das métricas para um processo ágil, como o SCRUM, onde a produtividade passa a ser medida não mais por tarefa mas por alguma unidade mais representativa, como o *sprint* ou o *history*, entre outras melhorias na ferramenta em si. Na parte da implementação, pode-se atualizar o plugin para suportar a última versão do Redmine e também traduzi-la para o Português.

Referências bibliográficas

C. Gresse von Wangenheim, A. von Wangenheim, J. I. Lino. Medição de Software – Guia Prático. Bookess, ISBN 9788580453362, Janeiro 2012. 265p.

CMMI para Desenvolvimento, Versão 1.2 CMMI-DEV, V1.2 CMU/SEI-2006-TR-008 ESC-TR-2006-008 Melhorando processos para obter melhores produtos Equipe do Produto CMMI - Agosto de 2006, Tradução: André Villas Boas e José Marcos Gonçalves

G. Gordon Schulmeyer. Handbook of Software Quality Assurance , 1999, 4 ed., Prentice Hall.

WIKIPÉDIA, CMMI, Disponível em: http://pt.wikipedia.org/wiki/CMMI, Data de acesso: 20/06/2013

ISDBrasil, Disponível em: http://www.isdbrasil.com.br/, 20/06/2013

Bruno Marota, CMMI para concursos, Disponível em: http://brunomarota.blogspot.com.br/2012/04/cmmi-para-concursos-parte-3.html, Data de acesso: 21/06/2013

Phlegx Systems, Redmine Gitolite Hook, Disponível em: http://github.com/phlegx/redmine_gitolite_hook, Data de acesso: 07/07/2013

FAGAN, M.E. Design and Code Inspection to Reduce Errors in Program Development, IBM Systems Journal, vol. 15, no. 3, pp. 182-211, 1976

Farias, Efraim dos Santos. Desenvolvimento de um plugin no Redmine para auxílio no processo de Inspeção, UFSC, 2013

F. S. Roberts. Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences. Encyclopedia of Mathematics and its Applications, Addison-Wesley, 1979.

N. Fenton. Software Measurement: A Necessary Scientific Basis. IEEE Transactions on Software Engineering, vol. 20, No. 3, March 1994.

International Organization for Standarization. ISO/IEC 15939 Systems and software engineering -- Measurement process, 2007.

V. R. Basili, D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, IEEETransactions on Software Engineering, pp. 728-738, Vol. SE-10, no. 6, November 1984.

V. R. Basili et al. Goal/Question/Metric Approach. In J. Marciniak (ed.), Encyclopedia of Software Engineering, volume 1. John Wiley & Sons, 1994.

V. R. Basili et al. Experience Factory. In John J. Marciniak (ed.), Encyclopedia of Software Engineering, vol.1. John Wiley & Sons, 1994.

C. Gresse von Wangenheim et al. Melhoramento de Software Baseado em Mensuração - Como Aplicar GQM na Pratica?. Material de tutorial IX CITS - Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba, Brasil, 1998.

C. Gresse et al. A Process Model for GQM-Based Measurement. Technical Report STTI-95-04-E, Software-Technologie-Transfer- Initiative Kaiserslautern, Alemanha, 1995.

Department of Defense and US Army. Practical Software and Systems Measurement: A Foundation for Objective Project Management, v. 4.0. Practical Software & Systems Measurement, EUA, 2003 (http://www.psmsc.com).

LocMetrics, Disponível em: http://www.locmetrics.com/, Data de acesso: 07/07/2013

K. Schwaber, J. Sutherland. Scrum Guide, Julho 2013.