Qualidade de Código em Software Livre

Plano de Medições - 1/2014

Matheus Souza Fernandes - 11/0017765
Fagner Rodrigues - 09/0112750

2 de julho de 2014

Sumário

Lista de Figuras ii
Lista de Tabelas iii
1 Introdução 1
1 .1 Goal - Question - Metric 1
1 .2 Modelo de Qualidade de Software 1
1 .3 Software Livre e Qualidade de Código 2
1 .4 Radar Parlamentar 3
2 Definição dos Objetivos da Medição 4
3 Questões a serem respondidas 5
4 Métricas de Código-Fonte 6
4 .1 M1: LOC - *Lines of Code* 6
4 .2 M2: AMLOC - *Average Method LOC* 6
4 .3 M3: PODC - *Percentage Of Duplicated Code* 7
4 .4 M4: ACCM - *Average Cyclomatic Complexity per Method* 8
4 .5 M5: ACC - *Afferent Connections per Class* 8
4 .6 M6: AWP - *Accordance With PEP8* 9
4 .7 M7: TC - *Test Coverage* 9
4 .8 M8: DOCL - *Density of Commented Lines* 10
4 .9 M9: MI - *Maintainability Index* 10
5 Coleta de Dados 12
6 Resultados e Análises 12
6 .1 Dashboard 12
6 .2 Hotspots 14
6 .3 Árvore de Dependência 15
7 Considerações Finais 16
Glossary 17
Referências Bibliográficas 19

Lista de Figuras

1 GQM - Exemplo 1
2 ISO/IEC 9126 2
3 Cobertura de Código por Módulo 12
4 Dashboard - SonarQube 13
5 Densidade de Linhas Comentadas 13
6 Porcentagem de Código Duplicado 13
7 Complexidade Ciclomática 14
8 Cobertura de Código 14
9 Hotspots 14
10 Árvore de Dependência 15

Lista de Tabelas

1 GQM - Objetivo 4
2 M1: Linhas de Código 6
3 M2: Linhas de Código por Método 7
4 M3: Porcentagem de Código Duplicado 7
5 M4: Complexidade Ciclomática por Método 8
6 M5: Conectividade por Classe 9
7 M6: Padronização de Código 9
8 M7: Cobertura de Código 10
9 M8: Densidade de Linhas Comentadas 10
10 M9: Índice de Manutenibilidade 11

1 Introdução

1 .1 Goal - Question - Metric

A medição de software é algo imprecindível quando se trata de melhorias de processos. Sem ela não conseguimos avaliar *qual* aspecto do processo e *quanto* é passível de melhorias. Mas apenas medir sem nenhum objetivo também não vantajoso, já que medição é um processo caro e trabalhoso. Visando resolver esse problema foi criado o GQM, um método simples para planejar as medições de forma que elas sejam baseadas em objetivos específicos de medição. Dessa forma, são definidas somente as medições realmente úteis ao projeto.

O QGM é divido em três níveis: conteitual, operacional e quantitativo [1].

- **Conceitual:** onde são definidos os objetivos da medição em termos de ponto de vista, ambiente e atributos de qualidade. (GOAL)
- **Operacional:** onde os objetivos, definidos no nível conceitual, são refinados em um conjunto de perguntas que, respondidas, conseguem atingir os objetivos. (QUESTION)
- **Quantitativo:** onde, para cada pergunta, são definidas as métricas relevantes para responder à pergunta. (METRIC)

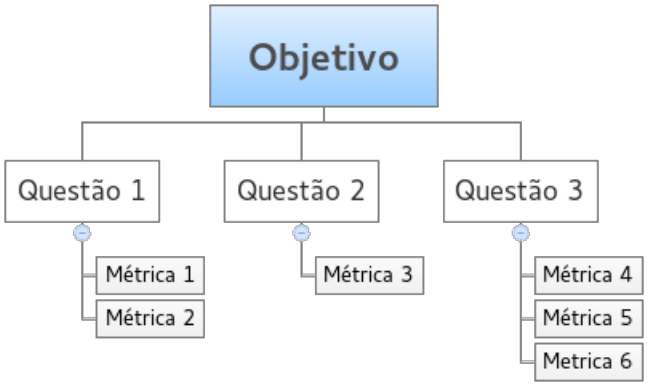

Figura 1: GQM - Exemplo

1 .2 Modelo de Qualidade de Software

Neste projeto serão tratados apenas os aspectos de qualidade de código. Estes são tratados dentro da ISO/IEC 9126, que define a qualidade de software em seis categorias: Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenibilidade e Portabilidade. O foco deste projeto será na manutenibilidade do software.

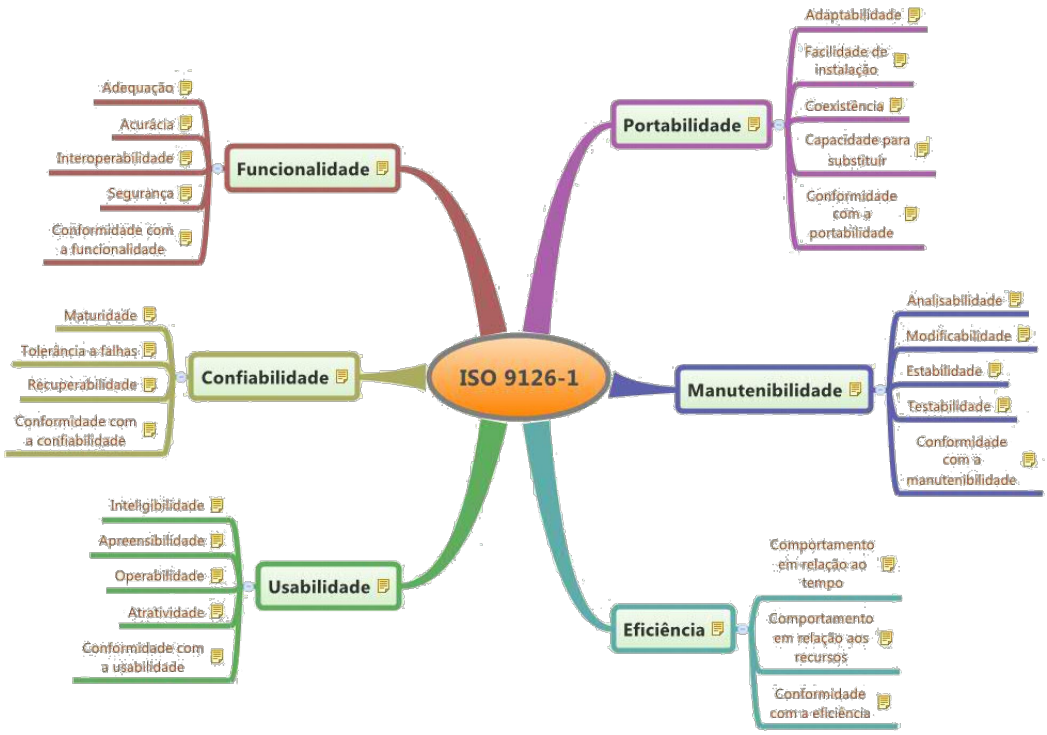

Figura 2: ISO/IEC 9126

Manutenibilidade de Software

De acordo com a ISO-9126, Manutenibilidade é a capacidade, ou facilidade, de um software ser modificado, incluindo tanto as melhorias ou extensões de funcionalidade, quanto as correções de defeitos, falhas ou erros. A Manutenibilidade pode ser subdivida em:

- **Analisabilidade:** Identifica a facilidade em se diagnosticar eventuais problemas e identificar as causas das deficiências ou falhas, ou seja, a facilidade em se encontrar falhas no sistema, quando elas ocorrerem.
- **Modificabilidade:** Caracteriza a facilidade com que o comportamento do software pode ser modificado, o quão fácil ele é de modificar. Sistemas com baixo nível de qualidade de código são muito mais difíceis de modificar, tanto para adicionar novas funcionalidades quanto para alterar as já existentes sem influenciar o comportamento do resto da aplicação.
- **Estabilidade:** Avalia a capacidade do software de evitar efeitos colaterais decorrentes de modificações introduzidas, ou seja, o que garante que o software continuara funcionando corretamente com modificações futuras.
- **Testabilidade:** Representa a capacidade de se testar o sistema modificado, tanto quanto as novas funcionalidades quanto as funcionalidades não afetadas diretamente pela modificação

1 .3 Software Livre e Qualidade de Código

O princípio básico do Software Livre é promover a a liberdade do usuário, sem discriminar quem tem permissão para usar um software e seus limites de uso, baseado na colaboração e num processo de desenvolvimento aberto [2]. Ou seja, software livre é aquele que permite aos usuários usá-lo, estudá-lo, modificá-lo e redistribuí-lo, em geral, sem restrições. O código é, necessáriamente, licensiado sob os termos legais formais que estão de acordo com as definições da Free Software Foundation ou da Open Source Initiative.

O modelo de desenvolvimento de software livre é conhecido por duas principais vantagens:

- Potencial de revisão por diferentes partes.
- Possibilidade de atrair desenvolvedoresao redor do mundo [3].

Em um de seus estudos, Santos Jr. [4] definiu um modelo teórico para *atratividade* de projetos de software livre, onde a atratividade também é influenciada por atributos de código fonte, como tamanho e complexidade estrutural observadas através de métricas de código.

Embora as métricas analisadas nesse projeto não indiquem completamente se um projeto de software livre é um projeto bem sucedido, elas podem oferecer indícios para ajudar a alcançar ou manter o "sucesso"do projeto. Além disso, a maior parte das contribuições recebidas são em torno do seu código-fonte, que é o principal artefato gerado e gerenciado pela sua comunidade [2].

1 .4 Radar Parlamentar

Radar Parlamentar é um software livre que determina "semelhanças"entre partidos políticos baseado na análise matemática dos dados de votações de projetos de lei na casa legislativa. Essas semelhanças são apresentadas em um gráfico bi-dimensional, em que círculos representam partidos e a distância entre esses círculos representa o quão parecido esses partidos votam.

Foi escrito em Python, utilizando o framework Django, e é uma aplicação web. O Radar Parlamentar é mantido pelo PoliGNU - grupo de estudos de software livre da Poli-USP. Além de alguns membros do PoliGNU, outros contribuidores se juntaram ao projeto com os mais diversos tipos de perfis e contribuições, inclusive mas não somente: desenvolvimento, design, documentação, algoritmos e ciência política.

Uma das parcerias firmadas para o desenvolvimento do Radar foi realizada com o professor doutor Paulo Meirelles, professor do curso de Engenharia de Software da Universidade de Brasília (UnB). Nos dois semestres de 2013 e no primeiro de 2014 a disciplina de Manutenção e Evolução de Software, ministrada pelo prof. Paulo, contou com um grupo de alunos dedicados à manutenção e evolução do Radar Parlamentar.

2 Definição dos Objetivos da Medição

Os estudos realizados na primeira etapa do projeto serviram de base para a definição dos objetivos desse plano de medições:

Analisar	Radar Parlamentar
Com o propósito de	Avaliar
Com respeito a	Qualidade de Código
Do ponto de vista da	Equipe de Desenvolvimento
No contexto de	Software Livre

Tabela 1: GQM - Objetivo

Juntamente com o objetivo, definimos, também duas hipóteses:

H1: A atratividade de um projeto de software livre é inversamente proporcional à complexidade de seu código.

Quanto maior a complexidade do código, menos atrativo se torna o projeto. Isso se dá por conta da dificuldade de compreensão do código, o que leva à um aumento do esforço de manutenção e fica cada vez mais difícil de atrair novos mesmbros e usuários. Com o passar do tempo, isso pode resultar em uma redução considerável de equipe que, por sua vez, diminui a capacidade de adicionar novas funcionalidades ao projeto.

H2: Projetos de software livre maiores tendem a ser mais atrativos

Geralmente, o tamanho do software é um relfexo do esforço e trabalho de seus colaboradores. Portanto, projetos maiores tendem a ser mais atrativos, já que oferecem mais oportunidades de contribuição e reconhecimento.

3 Questões a serem respondidas

Afim de alcançar o objetivo definido anteriormente, foram propostas as seguintes questões:

Q1: Qual o tamanho do software?

Essa pergunta tem o objetivo de verificar a veracidade da hipótese H2, se, realmente, o tamanho de um software torna o projeto mais atrativo.

Q2: Qual a complexidade do software?

A hipótese H1 nos diz que um software mais complexo possui uma menor atratividade na comunidade de software livre. Respondendo a essa questão, podemos analisar a complexidade do código e, por sua vez, a atratividade de acordo com a complexidade.

Q3: Qual o índice de Manutenibilidade?

Para juntar as respostas das perguntas anteriores, precisamos saber o índice de manutenibilidade do software em questão, para, enfim, avaliar a atratividade do projeto em relação ao tamanho e complexidade.

4 Métricas de Código-Fonte

Monitorar a qualidade do software é fundamental, independente da metodologia utilizada para o desenvolvimento. Várias características de um bom software são reflexos do código-fonte e, algumas vezes, exclusivos dele. Portanto, as métricas de código-fonte podem complementar outras abordagens de monitoramento da qualidade de software. Por exemplo, um código compilado pode ser analisado, mas características como organização e legibilidade são perdidas [2].

Para poder melhorar ou incrementar uma implementação, é necessário que o programador leia e, principalmente, entenda o código. Por isso, manter um código claro e simples é muito importante e as métricas de código-fonte podem ajudar nesse sentido.

Visando responder às perguntas definidas em *Questões a serem respondidas*, foi definido um conjunto de métricas de código, que será explicado a seguir.

4 .1 M1: LOC - *Lines of Code*

O Número de Linhas de Código é a medida mais comum utilizada quando se quer medir o tamanho de um software. Mas são contadas apenas as linhas de código que são executadas, ou seja, comentários e linhas em branco não entram na contagem.

Objetivo da Medição	Obter o número de linhas de código que o código possui.
Fórmula	Linhas de Código Executável
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a contagem das das linhas de código para análise.
Análise	**Responsável:** Analista **Procedimentos:** Verificar o tamanho do software.

Tabela 2: M1: Linhas de Código

4 .2 M2: AMLOC - *Average Method LOC*

É a média de linhas de código por método, uma métrica derivada da métrica anterior. Essa medida verifica se o código é bem dividido entre os métodos. Quanto maior o tamanho do método, mais coisa ele faz, o que significa que será mais dificil de ler e entender. Os intervalos sugeridos são [2]:

- **Bom:** Até 10 linhas/método.
- **Regular:** De 10 à 13 linhas/método.
- **Ruim:** Mais de 13 linhas/método.

Objetivo da Medição	Obter o número médio de linhas de código que cada método possui.
Fórmula	$\frac{LOC}{N^o\ de\ Métodos} * 100$
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a contagem das linhas de código que possui no método para análise.
Análise	**Responsável:** Analista **Procedimentos:** Verificar o tamanho do método para categorização.

Tabela 3: M2: Linhas de Código por Método

4 .3 M3: PODC - *Percentage Of Duplicated Code*

Mede a quantidade de código duplicado no projeto. Um código que possui muito código duplicado pode consumir mais tempo de leitura e entendimento, já que é mais código a ser lido, porém, sem necessidade.

Objetivo da Medição	Obter o número de código que se repetem, ou seja, código que possui a mesma ação.
Fórmula	$CD = \frac{QLD}{QTL} * 100$ CD - Código duplicado. QLD - Quantidade de linhas duplicadas. QTL - Quantidade total de Linhas.
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a contagem das funções duplicadas no código para análise.
Análise	**Responsável:** Analista **Procedimentos:** Verificar a quantidade de funções duplicadas e categorizar o código. do software.

Tabela 4: M3: Porcentagem de Código Duplicado

4 .4 M4: ACCM - *Average Cyclomatic Complexity per Method*

Mede a complexidade do software e pode ser representada como um grafo de fluxo de controle, onde os nós representam uma ou mais instruções e os arcos orientados indicam o sentido.

Objetivo da Medição	Obter a complexidade do código, medindo a quantidade de caminhos de de execução independentes a partir do código.
Fórmula	$$CP = (QS - QN) + QCC$$ CP - Complexidade ciclomática. QS - Quantidades de setas. QN - Quantidade de nós. QCC - Quantidade de componentes conectados.
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a medição com a feramenta para este fim no código para análise.
Análise	**Responsável:** Analista **Procedimentos:** Verificar os resultados obtidos da análise e quantificar a complexidade existente do código.

Tabela 5: M4: Complexidade Ciclomática por Método

4 .5 M5: ACC - *Afferent Connections per Class*

Mede a conectividade de cada classe. Se o valor for muito grande, uma alteração na classe pode causar diversos efeitos colaterais no código, tornando a manutenção do código mais dificil. Os intervalos sugeridos são [2]:

- **Bom:** Até 2 classes.
- **Regular:** De 2 à 20 classes.
- **Ruim:** Mais de 20 classes.

Objetivo da Medição	Obter a quantidade de classes conectadas a uma classe fornecedra, ou seja, medir o acoplamento dessa classe.
Fórmula	Quantidade de chamadas para métodos de fora da classe
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a medição com a feramenta para este fim no código para análise.
Análise	**Responsável:** Analista **Procedimentos:** Verificar os resultados obtidos da análise e quantificar a quantidade de classes acopladas código.

Tabela 6: M5: Conectividade por Classe

4 .6 M6: AWP - *Accordance With PEP8*

Avalia o padrão do código do projeto de acordo com a PEP8 (*Style Guide for Python Code*). Um código padronizado torna a manutenção muito mais simples e facilita a leitura e a compreensão do código.

Objetivo da Medição	Avaliar o padrão de estilo do código, verificando se o mesmo segue o padrão de estilo determinado.
Fórmula	Quantidade de itens fora do padrão PEP8
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a medição com a feramenta para este fim no código para análise.
Análise	**Responsável:** Analista **Procedimentos:** Verificar os resultados obtidos da análise e verificar o quanto o códogo está detro do padrão de estilo.

Tabela 7: M6: Padronização de Código

4 .7 M7: TC - *Test Coverage*

Define a cobertura de código oferecida pela suíte de teste do projeto. Quanto maior for a cobertura, menor é a chance de uma alteração causar algum tipo de efeito colateral no código.

Objetivo da Medição	Obter a cobertura de código por testes, ou seja, o quanto seu código está sendo testado por código de teste.
Fórmula	$\frac{Linhas\ de\ Código\ Testado}{LOC} * 100$
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a quantificação da porcentagem de cobertura de código.
Análise	**Responsável:** Analista **Procedimentos:** Verificar os resultados obtidos da análise e avaliar o quanto o código está coberto por testes válidos.

Tabela 8: M7: Cobertura de Código

4 .8 M8: DOCL - *Density of Commented Lines*

A quantidade de linhas comentadas pode auxiliar no entendimento do código, mas isso depende da qualidade do comentário, pois, as vezes, um comentário muito grande pode ser desnecessário.

Objetivo da Medição	Obter a quantidade de comentários existentes no código.
Fórmula	$\frac{Quantidade\ de\ Linhas\ Comentadas}{LOC}$
Escala da Medição	Absoluta
Coleta	**Responsável**: Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a quantificação de comentários existentes no código.
Análise	**Responsável:** Analista **Procedimentos:** Verificar os resultados obtidos da quantificação de comnetários e avaliar se são comentários pertinentes.

Tabela 9: M8: Densidade de Linhas Comentadas

4 .9 M9: MI - *Maintainability Index*

É o índice de manutenibilidade. O objetivo dessa métrica é medir quantitativamente a manutenibilidade de um sistema.

Objetivo da Medição	Obter o esforço necessário para modificar o software.
Fórmula	$171 - 5.2ln(V) - 0.23V(g) - 16.2 * ln(LOC) - 50sin(\sqrt{2.4 * CM})$ (1)
Escala da Medição	Absoluta
Coleta	**Responsável**: Analista e Desenvolvedores **Periodicidade**: Durante o desenvolvimento. **Procedimentos**: Realizar a quantificação do esforço para modificação no código.
Análise	**Responsável:** Analista **Procedimentos:** Verificar os resultados obtidos da quantificação do esforço e avaliar a manutenibilidade do software.

Tabela 10: M9: Índice de Manutenibilidade

5 Coleta de Dados

A extração das métricas definidas nesse documento foi realizada através das bibliotecas Coverage e PyLint do Python e disponibilizadas, via web, através do SonarQube. As medições estão disponíveis em: analiseradar.cloudapp.net .

O processo de coleta de dados se deu nas seguintes etapas:

1. Atualização do repositório local do projeto.
2. Aplicação dos testes unitários.
3. Aplicação dos testes de interface.
4. Cálculo de cobertura.
5. Análise do Código, utilizando PyLint.
6. Subir a análise para o Sonar.

Esse procedimento é executado a uma vez ao dia, todos os dias, para manter as medições sempre atualizadas.

6 Resultados e Análises

Logo ao acessar o servidor com o Sonar, vemos o gráfico abaixo, que mostra a cobertura de código separada por módulos. O tamanho dos quadrados é proporcional à quantidade de linhas de código em cada módulo e as cores são proporcionais à cobertura de código do módulo, quanto mais verde, maior a cobertura.

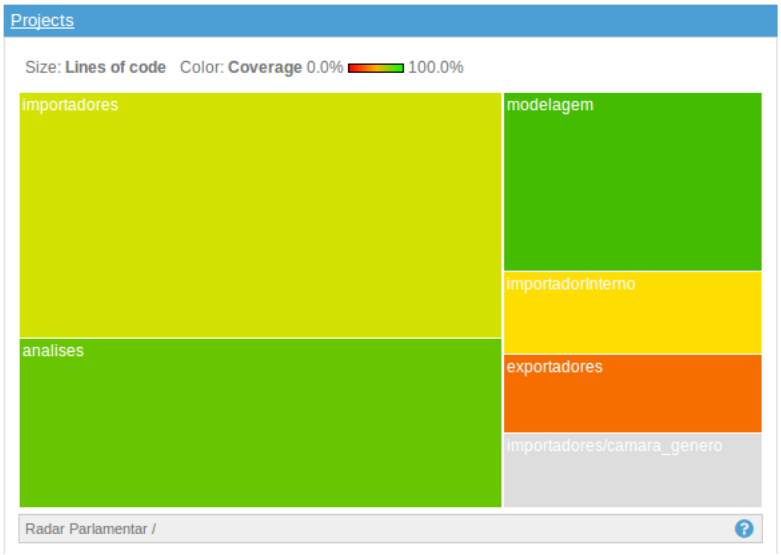

Figura 3: Cobertura de Código por Módulo

6 .1 Dashboard

Ao acessar o dashboard do Sonar, temos um painel com as principais medições realizadas, comparadas com a última análise realizada. São elas:

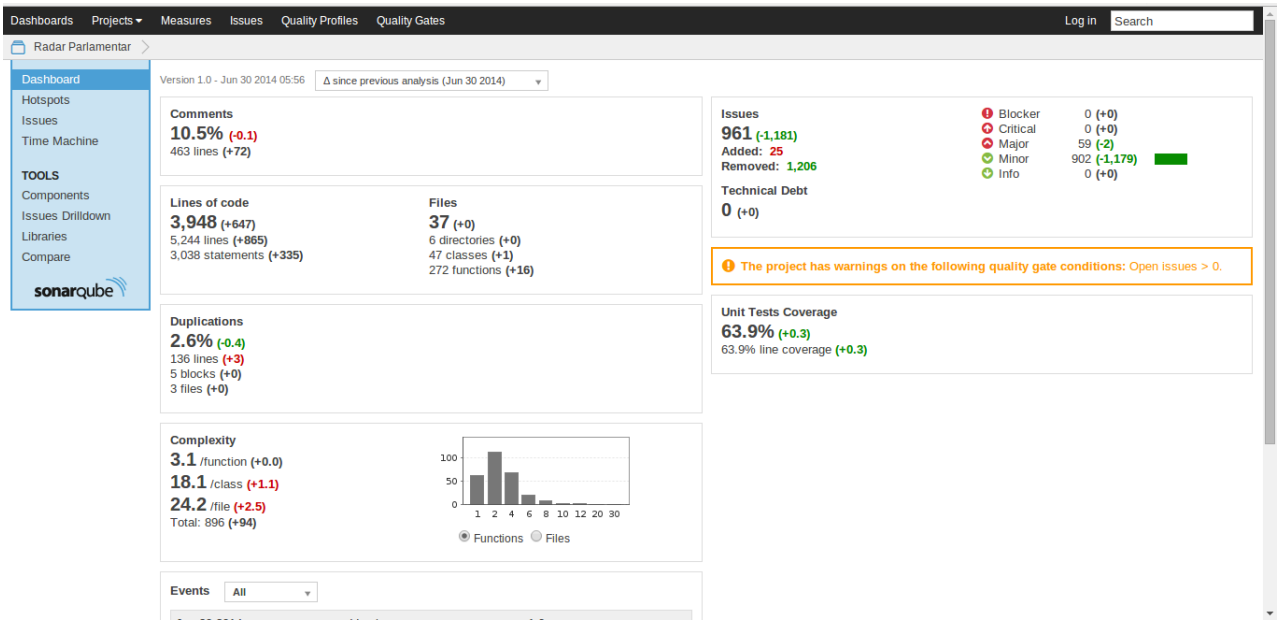

Figura 4: Dashboard - SonarQube

Os valores que se encontram dentro dos parênteses, nas medidas, são a variação da medida em relação à última análise. Alguns desses valores ficam verdes, outros ficam vermelhos e outros permanecem cinza. Isso acontece porque a forma de avaliar algumas métricas são diferentes, por exemplo: a redução da duplicação de código é algo bom para o código, por isso, mesmo sendo um valor negativo, o valor é verde.

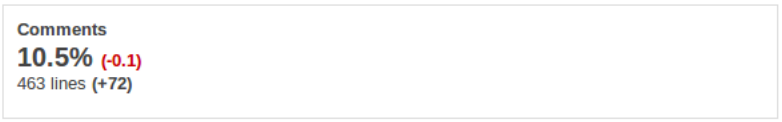

Figura 5: Densidade de Linhas Comentadas

Pode-se verificar o valor gerado de código duplicado ou seja, códigos que exercem a mesma ação. Assim evidencia-se a necessidade de rastrear os códigos que estejam duplicados para que sejam refatorados e posteriormente seja feita nova análise com a ferramenta.

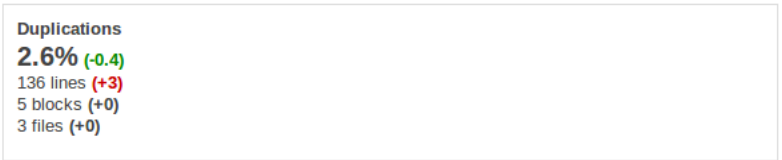

Figura 6: Porcentagem de Código Duplicado

Em relação a complexidade ciclomática, abaixo podemos verificar o resultado obtido colhido pela ferramenta, que mede nesse caso a quantidade de caminhos de execução independentes, quando a quantidade de fluxos é grande o código aumenta sua complexidade. Abaixo foram registrados os valores de complexidades para funções do código, pelas classes e por arquivos.

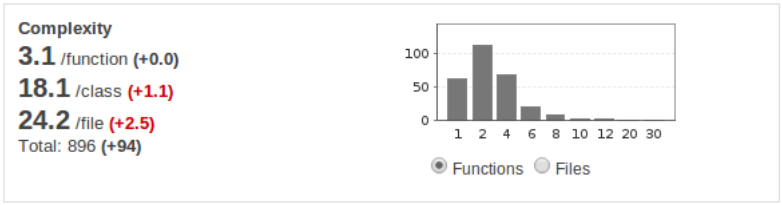

Figura 7: Complexidade Ciclomática

Para esta medida, foram medidos a quantidade de abrangência dos testes unitários sobre o código. O teste é importante para o código para prevenção de futuros bugs e erros que podem ocasionar ao usuário pelo uso do software, assim é necessário medir a abrangência desses testes para que também quando seja necessário uma manutenção o impacto e esforço seja menor. abaixo tem-se o registro da quantidade de cobertura de teste no código.

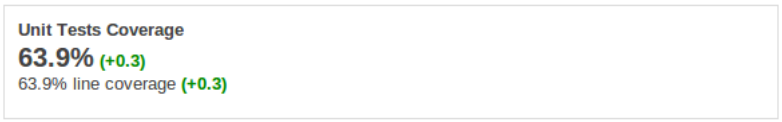

Figura 8: Cobertura de Código

6 .2 Hotspots

Ao acessar o hostpots do sonar podemos verificar um nível de rastreabilidade para a métrica desejada, no caso já localiza em que classe está determinada métrica pretendida, por exemplo para linhas duplicadas ele aponta a quantidade de linhas duplicadas e em quais classes estão. Também fornece outras medidas rastreáveis como complexidade, linhas reveladas, complexidade por função. E ainda fornece as regras mais violadas e recursos mais violados numa escala.

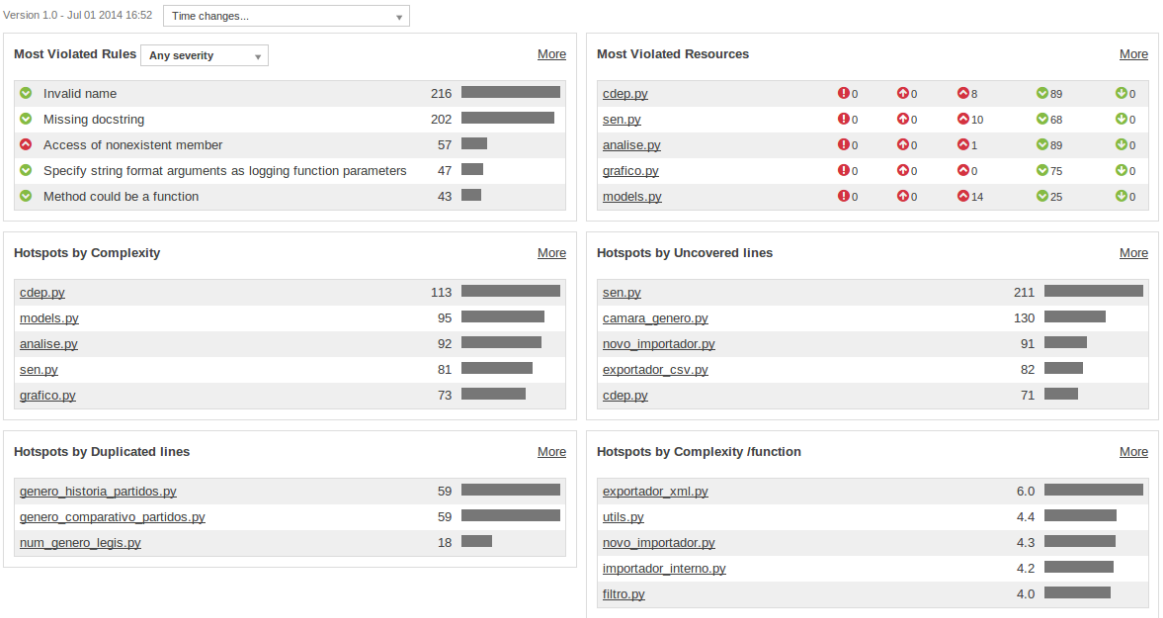

Figura 9: Hotspots

6 .3 Árvore de Dependência

Nesta sessão podemos verificar os dados de acoplamento das classes do software radar parlamentar, verificamos a quantidade de acoplamento de classes podendo então classificá-las em ruim, regular ou bom de acordo com o número de de classes acopladas a outra, podendo também rastrear a classe e fazer a refatoração para diminuir o acoplamento.

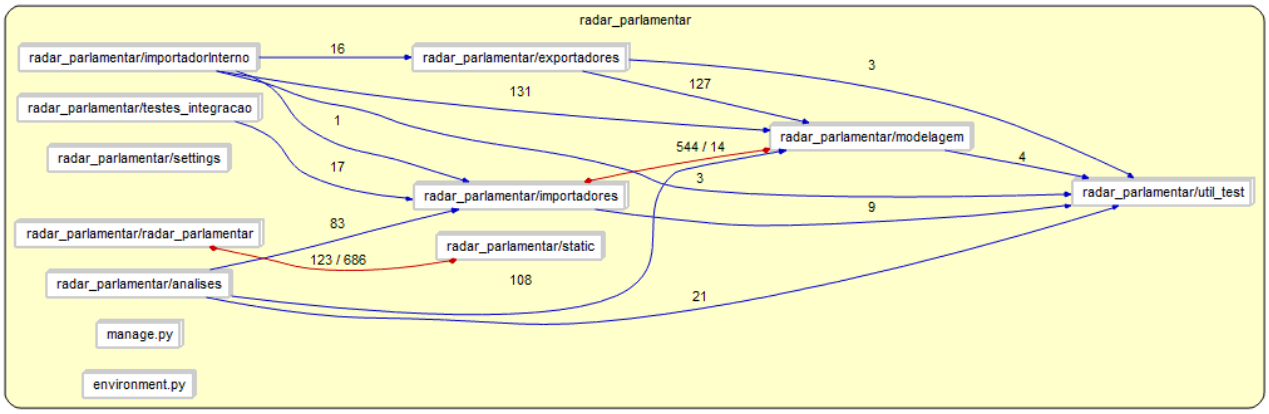

Figura 10: Árvore de Dependência

7 Considerações Finais

Métricas são fundamentais para a melhoria de um software e principalmente para que satisfaça o objetivo pretendido quando visado, pois é justamente nele que pode-se ganhar notoriamente grande evidência na melhora de uma métrica com o intuito de sempre alcançá-Ío. Também podemos notar a importância de se estabelecer métricas bem definidas, e fazer melhorias da mesma conforme o avanço e estabelecimento de novos objetivos, abstraindo uma nova concepção das métricas e por consequencia novas análises e mais ricas em conteúdo. Por fim podemos perceber que com relação aos dados das métricas colhidos é possível diminuir o impacto negativo que um código não quantificado metricamente e por consequencia sem qualidade gera para a manutenção de um software, ou seja, software com código de má qualidade em que não foram estabelecidadas métricas para sua administração para justamente rastrear problemas, geram um custo e um impacto negativo muito alto para sua manutenção.

As métricas coletadas neste projeto nos mostram um software com aproximadamente 4000 linhas de código, divididas em 37 arquivos, 47 classes e 272 métodos, que, por sua vez, têm uma complexidade média de 3.1, além de uma cobertura de 64% do código. Por outro lado, vemos também que, por meio de contribuições dos alunos da disciplina de Manutenção e Evolução de Software da UnB, uma grande quantidade de erros relacionados à padrões de código foram solucionados e, por consequencia disso, outras métricas ficaram melhores, como a duplicação de código, que melhorou consideravelmente.

Apesar de não conseguir atingir o objetivo proposto na primeira etapa do trabalho (comprovar que a qualidade do código pode afetar diretamente a atratividade do projeto), o grupo, ainda assim, se sentiu gratificado, pois, mesmo não alcançando algumas métricas, a experiência e a colaboração com o Radar Parlamentar foram bastante positivas.

Glossary

A | C | F | I | O | P | S

A

Atratividade

Atratividade é a capacidade de trazer usuários e desenvolvedores para um projeto. Um projeto de software livre é tão atraente quanto a sua capacidade de atrair potenciais usuários e programadores que, depois de um período como usuários passivos, passam a participar das tarefas de melhoria do projeto [4].

C

Coverage

Utilizado para medir a cobertura de código, geralmente durante a execução dos testes. Ele utiliza as ferramentas de análise de código nativas do Python para avaliar quais linhas são executáveis e quais foram executadas..

F

Free Software Foundation

A Free Software Foundation (FSF, Fundação para o Software Livre) é uma organização sem fins lucrativos, fundada em 04 de Outubro de 1985 por Richard Stallman e que se dedica a eliminação de restrições sobre a cópia, redistribuição, estudo e modificação de programas de computadores – bandeiras do movimento do software livre, em essência. Faz isso promovendo o desenvolvimento e o uso de software livre em todas as áreas da computação mas, particularmente, ajudando a desenvolver o sistema operacional GNU e suas ferramentas.

I

ISO/IEC 9126

ISO/IEC 9126 é uma norma ISO para qualidade de produto de software, que se enquadra no modelo de qualidade das normas da família 9000. A norma brasileira correspondente é a NBR ISO/IEC 9126.

O

Open Source Initiative

A Open Source Initiative (OSI) - Iniciativa pelo código aberto - é uma organização dedicada a promover o software de código aberto ou software livre. Ela foi criada para incentivar uma aproximação de entidades comerciais com o software livre. Sua atuação principal é a de certificar quais licenças se enquadram como licenças de software livre, e promovem a divulgação do software livre e suas vantagens tecnológicas e econômicas.

P

PyLint

A biblioteca PyLint é uma importante ferramenta no contexto de qualidade de código em Python, pois com ele é possível obter métricas de padronização de código, detecta erros e duplicações de código..

S

SonarQube

Sonar é um projeto *open source*, distibuído sob a licença LGPL v3. É uma ferramenta de análise estática de código, com suporte a diversas linguagens, uma delas é o Python, que também é a linguagem utilizada no Radar Parlamentar.

.

Referências Bibliográficas

[1] BASILI, G. C. V. R. The goal question metric approach. Institute for Advanced Computer Studies, 2002.

[2] MEIRELLES, P. R. M. Software livre. In: *Monitoramento de métricas de código-fonte em projetos de software livre*. [S.l.: s.n.], 2013.

[3] MICHLMAYR FRANCIS HUNT, D. P. M. Quality pratices and problems in free software projects. *First International Conference on Open Source System*, 2005.

[4] SANTOS, C. J. Attractiveness of free and open source software projects. *Proceedings of the 18th European Conference on Information Systems (ECIS)*, 2010.