Instituto Federal do Espírito Santo

# Projeto Capoeira

Briane Bianca

Gustavo Grimaldi Capello

Leticia Schulthais Senna

# Sumário

1 HISTÓRICO DO DOCUMENTO 2

2 INTRODUÇÃO 2

3 APRESENTAÇÃO 3

3.1 MINIMUNDO 3

3.2 TIPO DE USUÁRIO 3

3.3 DIAGRAMA DE CLASES 4

3.4 DESCRIÇÃO MVC 4

3.5 DESCRIÇÃO PADRÕES DE PROJETO 5

3.6 REFATORAÇÃO E AVALIAÇÃO DE QUALIDADE DE CÓDIGO (SONAR) 7

4 COMANDOS (COMPILAR, EXECUTAR E TESTAR) 10

5 REFERÊNCIAS 11

# 1 HISTÓRICO DO DOCUMENTO

| Versão | Responsável | Modificação | Data de Modificação |
|---|---|---|---|
| 1.0 | Elton Couto Rizzo | Criação do documento. | 23/06/2015 |
| 2.0 | Leticia Schulthais Senna | Inserção dos dados | 03/10/2015 |
| 3.0 | Briane Bianca, Gustavo Grimaldi e Leticia Senna | Atualização e inserção de dados | 18/10/2015 |
| 4.0 | Briane Bianca, Gustavo Grimaldi e Leticia Senna | Atualização e inserção de dados | 20/11/2015 |
| 5.0 | Briane Bianca, Gustavo Grimaldi e Leticia Senna | Atualização e inserção de dados | 15/01/2016 |

# 2 INTRODUÇÃO

Projeto Capoeira é um sistema para Professores de Capoeira. Cujo visa substituir as inscrições feitas em fichas impressas para cadastro on-line (acabar com fichas impressas, reduzindo volume, e facilitando a busca/armazenamento dos dados). O Projeto Capoeira é um produto administrativo, para monitorar inscrições, relação de alunos, controle de histórico de alunos, e atividades da capoeira. Que propõe dar agilidade ao processo de ingresso no projeto, facilitar o armazenamento das informações referentes ao que compõem o projeto (alunos, eventos e etc).

# 3 APRESENTAÇÃO

O projeto será inicialmente desenvolvido na linguagem Python, usando a interface gráfica do Django e conectando-se com o banco de dados Sqlite3.

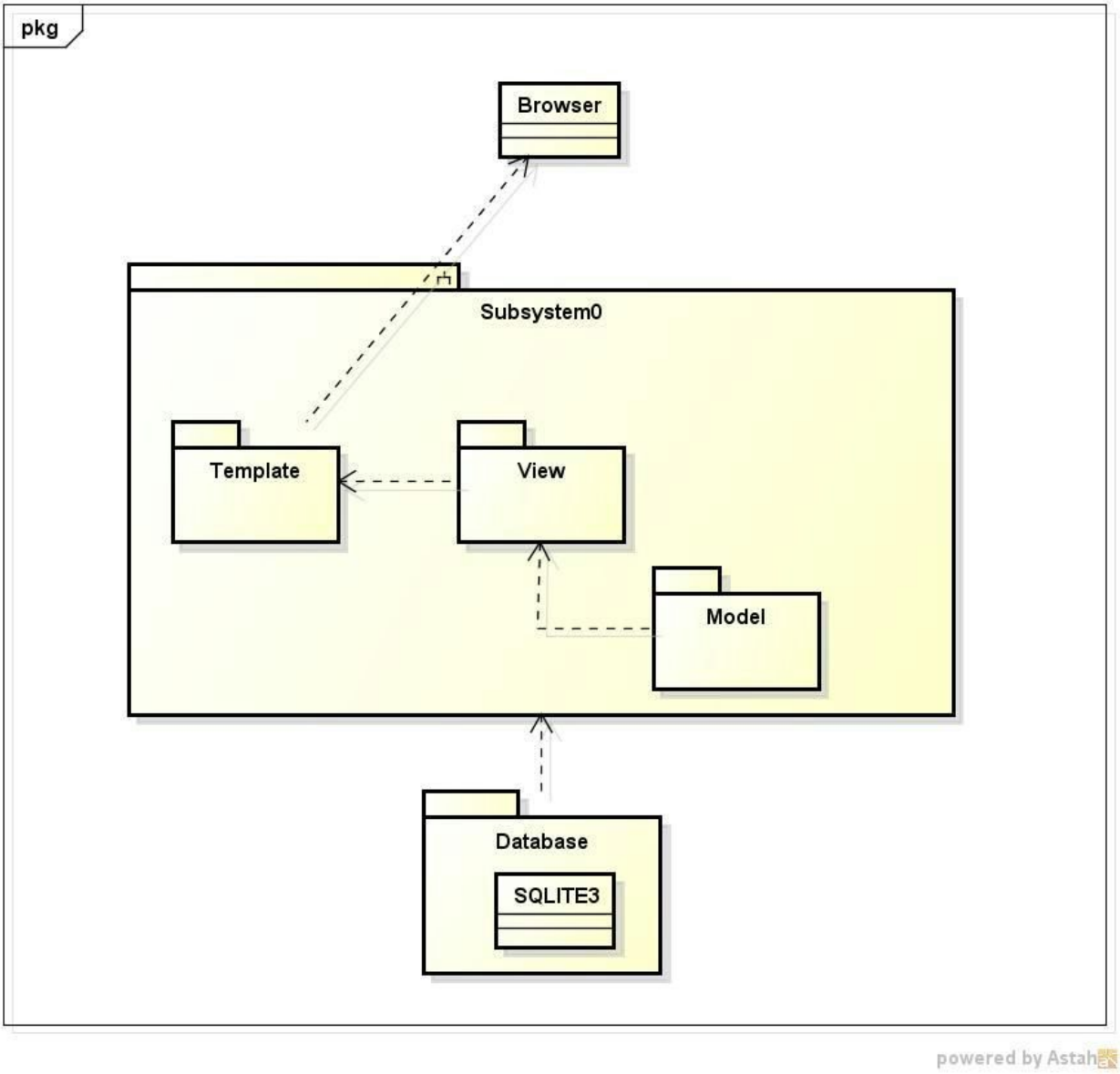
## 3.1 **MINIMUNDO**

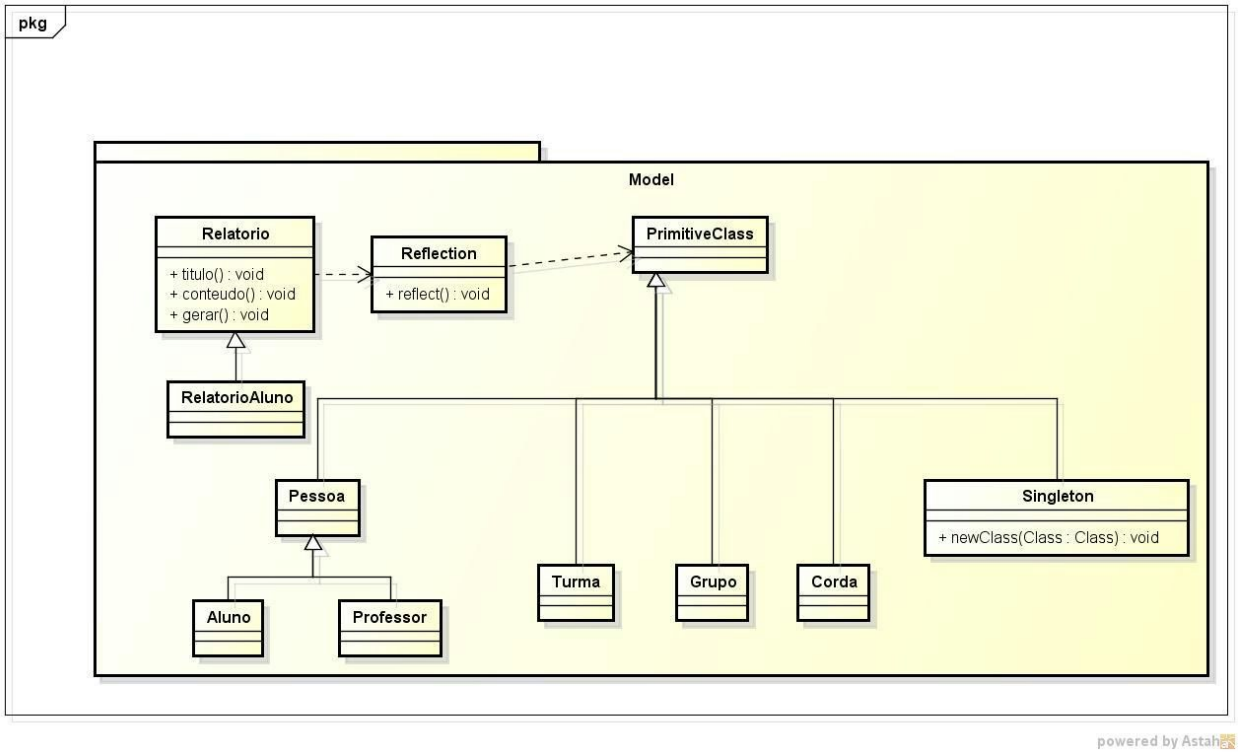
O projeto capoeira é um sistema que armazena digitalmente os dados de alunos, professores, local, entre outras informações relacionadas à prática da capoeira. O sistema deve permitir o cadastro de novos alunos (Nome, RG, Filiação, Data de Nascimento, Endereço, Telefone, Profissão, Grau de Escolaridade, Corda), Professores (Nome, RG, Data de Nascimento, Endereço, Telefone, Profissão, Grau de Escolaridade, Corda), Turmas (Nome, Turno, Horário, Dia da semana), Local (Logradouro, Numero, Bairro, Cidade, Complemento), Corda (Cor), Exame (Data do Exame, Horário, Mestre examinador, Local da Cerimônia, Turma), Grupo (Nome, Endereço, Sequência das cordas). Além disso, o sistema deve gerar um relatório de inscrições ao término de cada de período de inscrição, deve permitir somente um determinado número de inscrições por turma, formando um cadastro de reserva com as inscrições que ultrapassarem o limite.

## 3.2 **TIPO DE USUÁRIO**

Em geral as pessoas que irão utilizar o sistema possuem um grau de escolaridade suficiente, estando fora de níveis extremos, como por exemplo: o analfabetismo, o que impediria ou dificultaria a utilização do sistema.  Deixando claro que o projeto abrange a todos sem qualquer tipo de distinção e que qualquer dificuldade que possa surgir em relação à acessibilidade, seja ela de qualquer forma, será solucionada da maneira mais agradável para ambas as partes.

## 3.3 **DIAGRAMA DE CLASSES**






## 3.4 DESCRIÇÃO MVC

- Modelo-visao-controle (MVC – model-view-controller) é um padrão arquitetural de software para implementação da interface do usuário.
- Porém, como estamos utilizando o Django neste projeto ele usa as nomenclaturas para as camadas, diferente do modelo MVC. Para o modelo o Django usa o model, para view ele usa template (MTV), mas define o controller como view. E tem também as generic 'views' que são controllers genéricos para situações comuns como máster/detail, delete, update. (Fonte: http://pyman.blogspot.com.br/2007/04/o-mvc-o-mtv-e-o-django.html).
- No model se encontram todas as classes principais do sistema.
- Como o Django possui suas classes de configurações de ambiente, ele possui uma classe settings.py que conecta com o banco de dados e com as telas de comunicação com o usuário, e ele mesmo faz o controle através da classe admin.py.

## 3.5 **DESCRIÇÃO PADRÕES DE PROJETO**

- **Primeira etapa de padrões de projeto:** (Fábrica Abstrata, Método Fábrica, Singleton, Builder e Protótipo)

Utilizamos o singleton será utilizado em conjunto com o Decorator. Eles serão utilizados para o login de usuário, o singleton em cada classe principal, para q seja gerada apenas uma instância do objeto, e o decorator para cobrir a herança das classes. O método fabrica é a classe fábrica, que vai retornar um clone do objeto.

O Protótipo não será utilizado pois utilizaremos a herança nas classes.

Não utilização do padrão Fábrica Abstrata: Este padrão fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Como estamos utilizando o Django e este é um projeto simples não vimos a necessidade da aplicação desse padrão, pois o mesmo aumentaria o numero de classes e de complexidade do projeto.

Não utilização do padrão Builder: Este padrão tem como objetivo separar a construção de um objeto complexo da sua representação de modo que o mesmo processo de construção possa criar diferentes representações. Deve ser usado em situações onde a construção do objeto deve ser independente das partes que o compõem, de modo que seja possível a existência de diferentes representações para o objeto construído. Visto que em nosso modelo não existe classe que se encaixe nesse padrão, já que todas as classes do modelo tem comportamento semelhante, onde sua construção é dinâmica. Então a maioria das partes, que compõem os objetos, é informada pelo usuário dinamicamente, o que torna inviável a utilização do padrão Builder.

- **Segunda etapa de padrões de projeto:** (Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy).

O flyweight não será utilizado, pois os objetos do sistema são simples, sendo assim a equipe não viu a necessidade em implementar este padrão.

**Active Record:** esse padrão foi aplicado na hora de chamar o banco de dados do Django. O padrão determina que a interface de um certo objeto deve incluir funções de CRUD e propriedades que correspondam, de certa forma, diretamente às colunas do banco de dados associados.

Não há necessidade da utilização do adapter, pois não temos classes a serem inseridas no projeto que não se encaixe com as outras classes.

O Decorator foi justificado anteriormente, na primeira etapa, juntamente com o singleton.

Como temos um projeto simples e objetivo não implementamos o padrão bridge, pois não temos grandes incrementações de objetos havendo assim a necessidade de desacoplar a abstração de suas implementações. Nem o padrão composite será utilizado pelo mesmo motivo, projeto simples com objetos pequenos, não havendo necessidade de maior complexidade na implementação desse padrão.

Como estamos utilizando o Django, o mesmo possui o padrão template view, que fornece e controle dos templates e HTML, semelhante ao padrão facade, porém não possui uma inteface comum a todas as classes.

Não utilizamos o padrão Proxy, pois a equipe não viu a necessidade de um “substituto” ser exibido até a ação seja finalizada, como sugere este padrão. O Django possui esse padrão como se fosse um template que estende por herança até que a ação termine.

- **Terceira etapa de padrões de projeto:** (Interpreter, Template Method, Cadeia de Responsabilidade, Comando, Iterator, Mediador, Memento, Observer, State, Strategy, Visitor)

O Template Method:

O template foi construido para gerar um relatório base em que o metodo “gerar” retorna uma sequencia de execução de metodos que resulta na criaçao de um relatório, quem herdar esta classe, só sobrescreve os metodos que o criar chama.

Fabrica:

O fabrica retorna o tipo de classe que eu requisitei (e não a instancia dela) para que eu possa fazer consultas com o django para gerar o relatório a fabrica seria substituida pelo decorator, porém ao chegar em casa não consegui executar o projeto por um erro no sqlite (perdoe me) por isso uso uma das ultimas versoes comitadas que nao contem o reflection, que retornaria a classe solicitada para fazer as consultas:

```
class Reflection(models.Model):
    @staticmethod
    def reflect(class_nome):
        factory = [class_nome]
        path = "post.models"+class_nome
        _temp=__import__(path,fromlist=factory)
        classe = getattr(_temp, class_nome)
        classe=classe()
        return classe
```

Singleton + Decorator(?):

De acordo com algumas publicacões no projeto o Singleton foi implementado com o decorator, recebendo uma classe como paramentro e “decorando-a” com o padrão singleton sendo possível apenas instancia-la uma vez. Porem retenho o que falei em

sala de aula.O decorador não retornaria a mesma classe, eu iria averiguar ao chegar em casa se apontavam para mesma area de memoria.

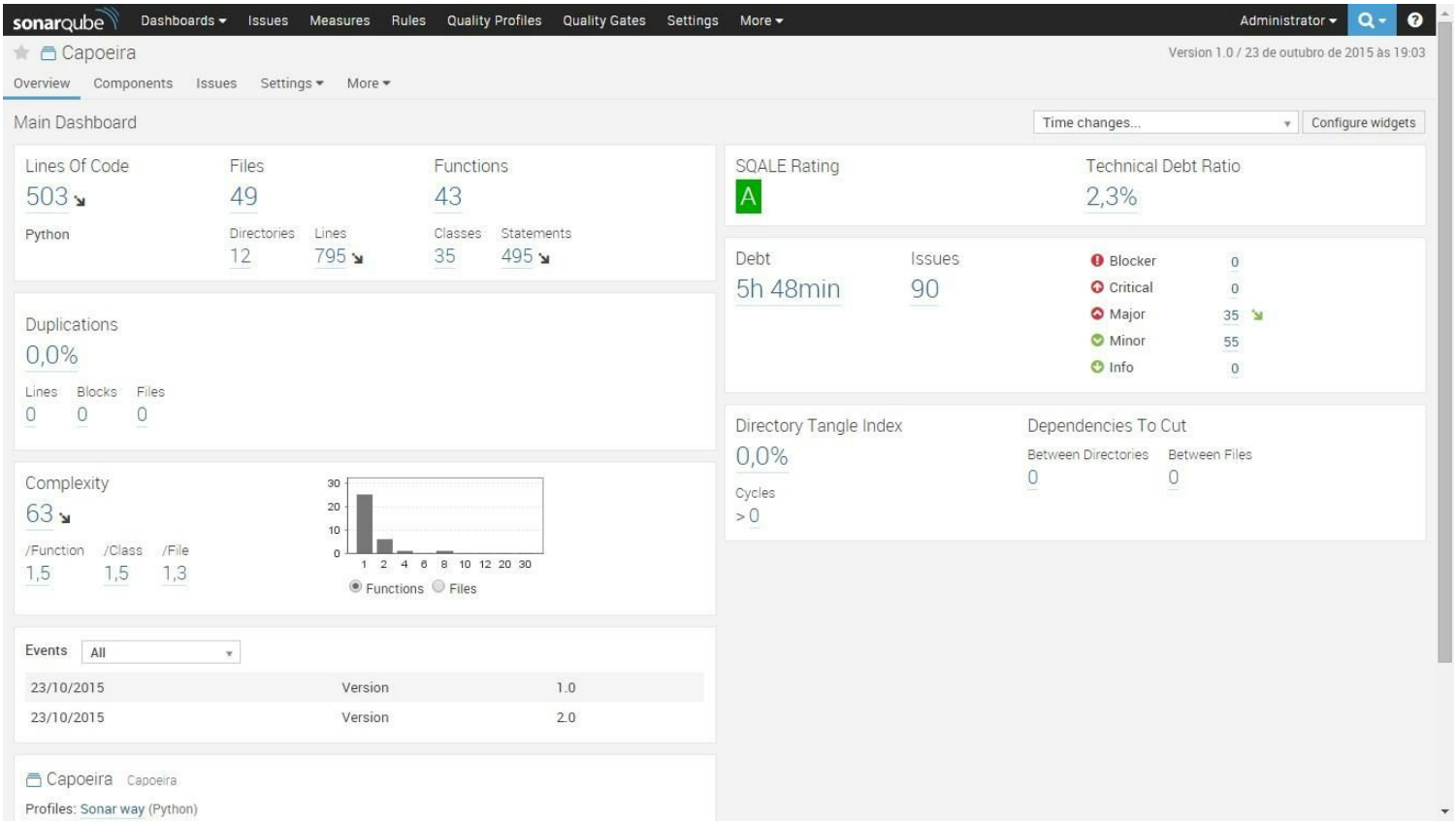
p.s. a proposta ideal para o singleton seria:

```
#Singleton pattern

class Singleton(type):
    def __init__(cls, name, bases, dict):
        super(Singleton, cls).__init__(name, bases, dict)
        cls.instance = None


    def __call__(cls,*args,**kw):
        if cls.instance is None:
            cls.instance = super(Singleton, cls).__call__(*args, **kw)
        return cls.instance

class Fabrica(object):
    __metaclass__ = Singleton
```

A equipe não viu a necessidade de implementar os outros padrões. Pelo fato do sistema ser simples, como já dito anteriormente, não se viu a necessidade de aumentar a complexidade do sistema, apenas para a adição dos padrões, podendo acarretar a instabilidade do sistema, a dificuldade da manutenção e as duplicações do mesmo.

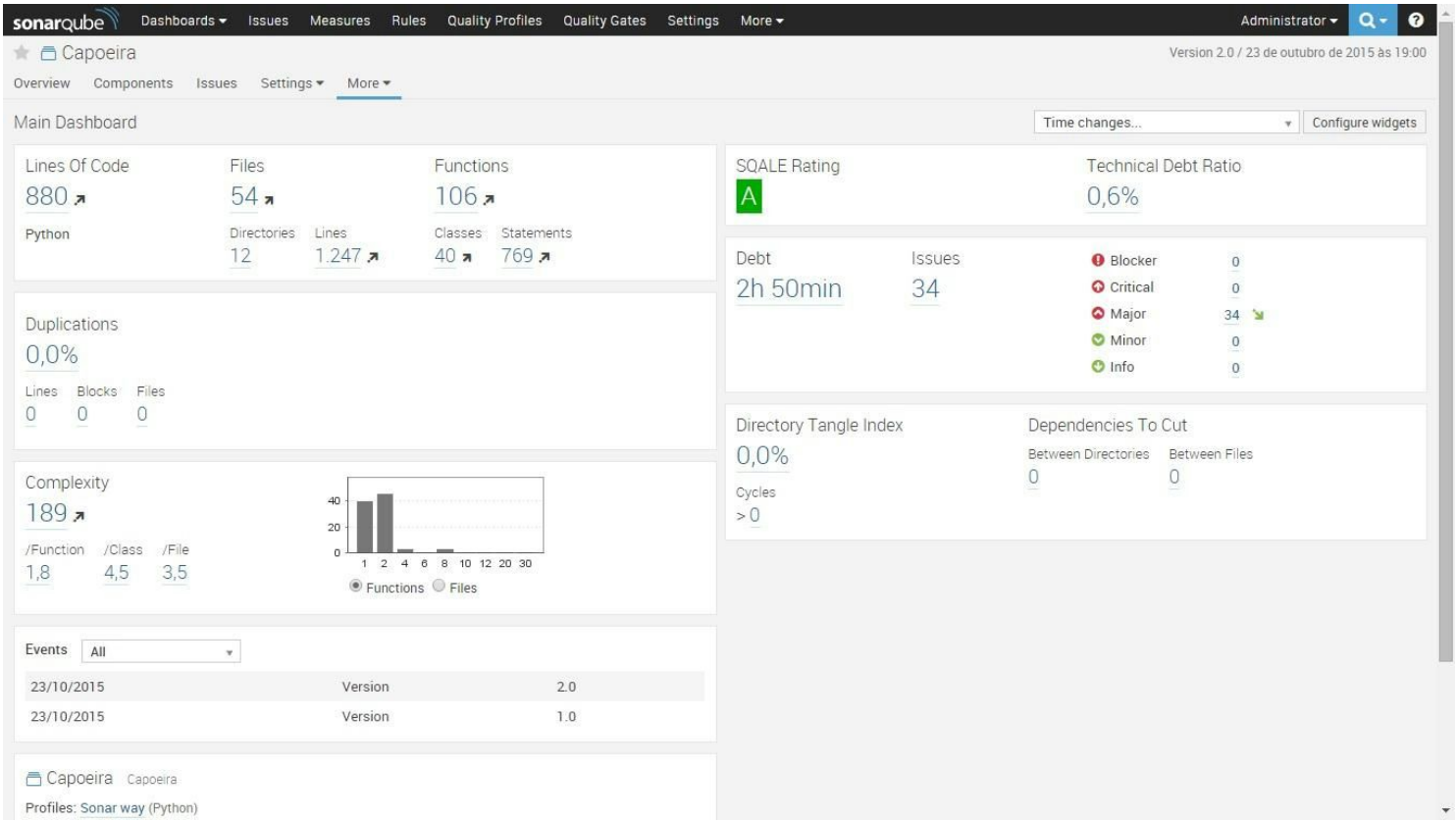
## 3.6 REFATORAÇÃO E AVALIAÇÃO DE QUALIDADE DE CÓDIGO (SONAR)

- Versão 1:




A primeira versão do projeto, bem simples, sem padrões de projeto.

- Versão 2:




Os padrões ajudaram principalmente na organização do código. Ajudaram a garantir funcionalidades do sistema, buscando eficiência e eficácia. Podemos observar uma redução de débitos técnicos no sistema, o que significa menos instabilidade.

A pesar de o Sonar indicar um aumento na complexidade observamos uma redução no tempo do débito e nos problemas. Com a ajuda do mesmo conseguimos reduzir os problemas maiores, zerar os menores e acertar o código onde poderia causar futuros problemas ou erros de execução.

O padrão Singleton melhorou o sistema pelo fato de, além de garantir a integridade do banco de dados, controla a abertura e fechamento do banco. Se

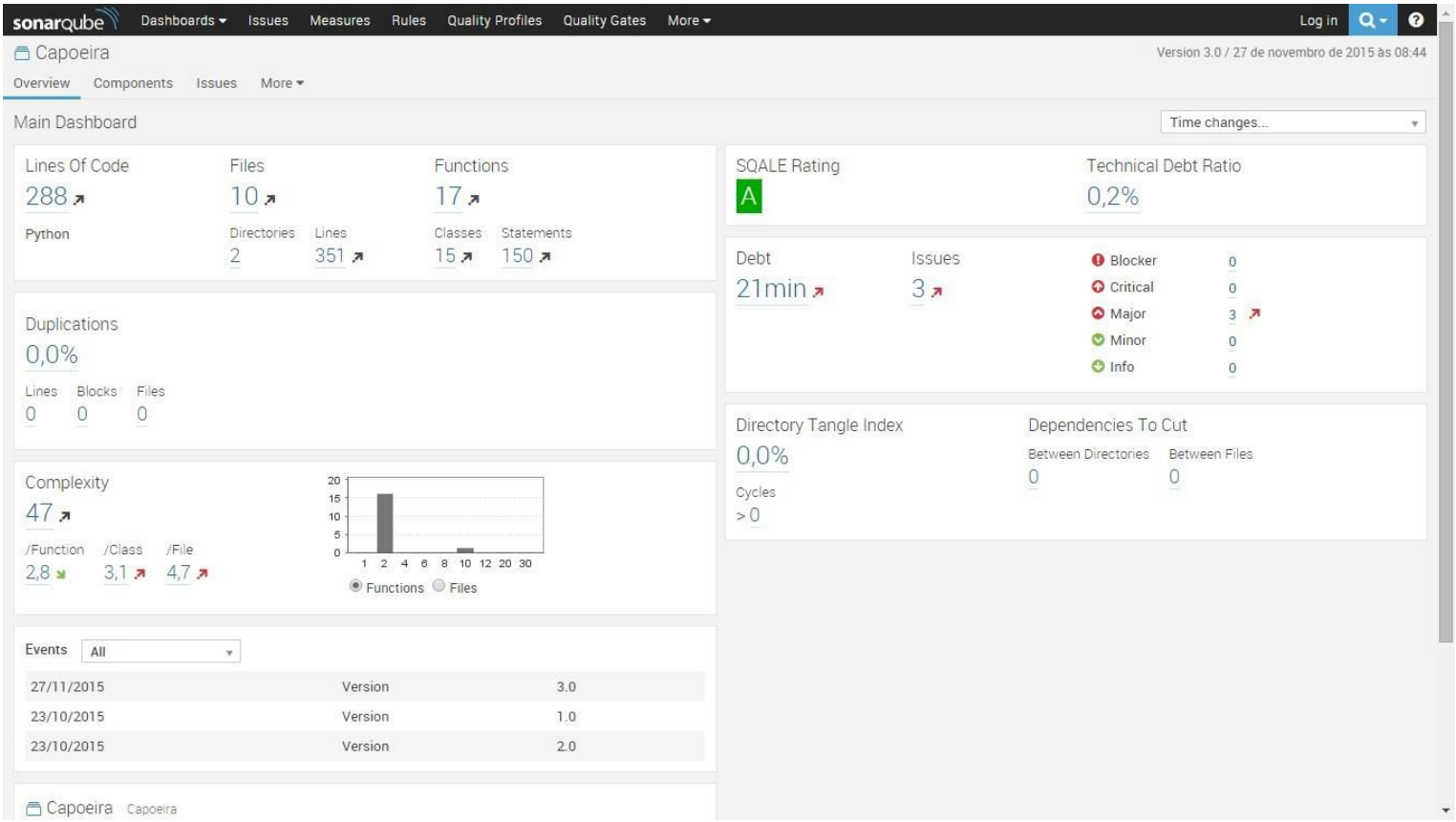
não fosse por ele, em casos de processos concorrentes e com duas instâncias da classe conexão, poderia causar problemas na persistência dos dados.

A Fábrica abstraiu o processo de construção dos objetos, trazendo modularidade para o código, pois caso precise ser efetuado alguma modificação futura não impactará muito.

A Fábrica Abstrata serve como molde de fábricas de linguagens diferentes de banco, no caso do projeto foi utilizado apenas o SQLite, mas se quiséssemos adicionar outro banco isso facilitaria.

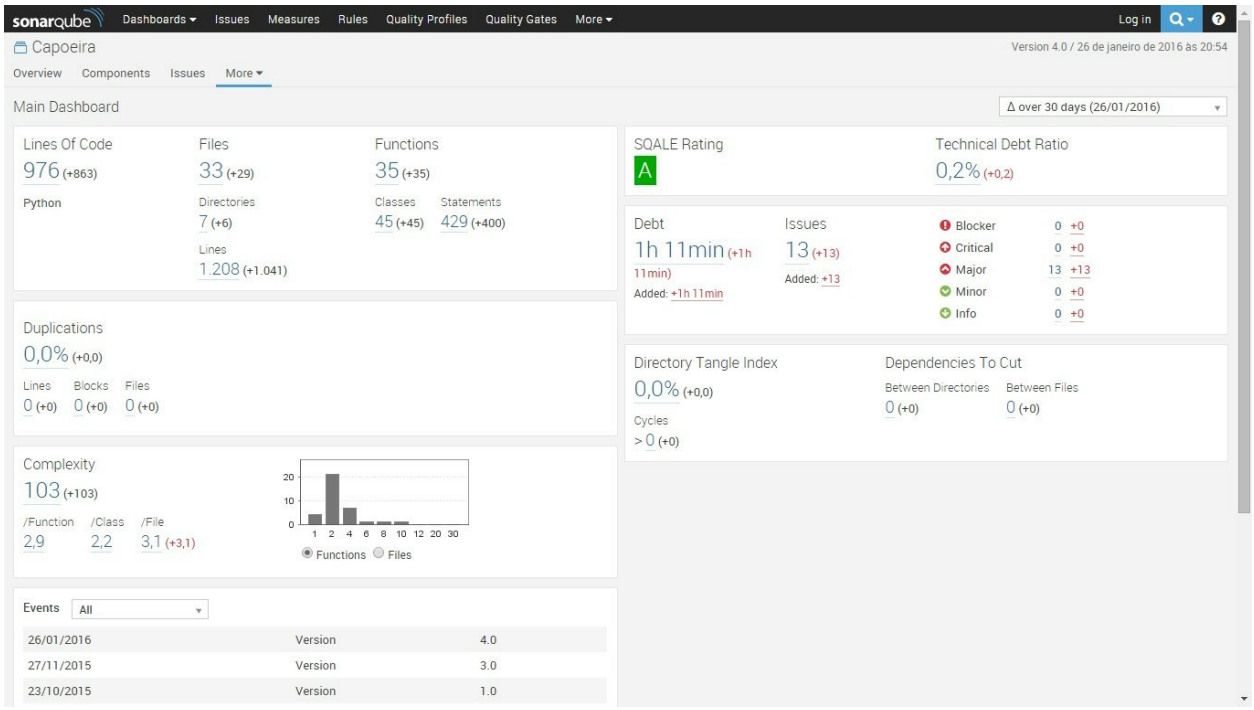
O Sonar ajudou a visualizar os erros que mais eram de vícios de linguagem, pois no curso programamos muito tempo com Java, e os maiores erros eram nomes de variáveis que usávamos do mesmo jeito que em Java. Os demais erros não puderam ser corrigidos, pois envolviam a criação de métodos abstratos, mesmo modificando a criação desses métodos e adicionando o @abstractmethod o sonar pedia que o método fosse estático.

- Versão 3: (Usando Django)




Com o sistema mais simples, poucos padrões, o Django reduz muito código, tornando assim o sistema menor e mais limpo.

- Versão 4: (Usando Django)




# 4 COMANDOS (EXECUTAR)

ACESSAR O TERMINAL E ENTRAR NA PASTA PYTHONCLUB DO DOCUMENTO E EXECUTAR O COMANDO "SOURCE BIN/ACTIVATE" APÓS ISSO ENTRAR NO SUBDIRETORIO CAPOEIRA E EXECUTAR "PYTHON MANAGE.PY MAKEMIGRATIONS" AGUARDAR E ASSIM QUE POSSÍVEL DIGITAR "PYTHON MANAGE.PY MIGRATE", APÓS ISSO CRIAR UM USUARIO PRO SISTEMA "PYTHON MANAGE.PY CREATESUPERUSER" E INSERIR NOME E SENHA (E-MAIL NÃO OBRIGATORIO. O DJANGO FORNECE MUITOS RECURSOS PARA UM SISTEMA WEB, INCLUSIVE GERENCIA DE USUARIOS, PORÉM NÃO FOI POSSÍVEL CRIAR UMA PAGINA DE LOGIN, PEÇO DESCULPAS, ENTÃO O COMANDO A SER EXECUTADO AGORA É O "PYTHON MANAGE.PY RUNSERVER" E ACESSAR HTTP://127.0.0.1:8000/ ADMIN/ E FAZER LOGIN COM O USUARIO. PARA ACESSARA PAGINA É NECESSARIO ENTRAR NO HTTP://127.0.0.1:8000/ POST/ (A RAIZ DO PROJETO ERA RESERVADA PARA O LOGIN, POREM PELO IMPREVISTO COM O BANCO NA MAQUINA LINUX NÃO CONSEGUI RODAR O PROJETO)

## Referências:

- Reflection no Django: http://chase-seibert.github.io/blog/2014/04/23/python-imp-examples.html

- Padrões de projeto no Django: Django Design Patterns ans Best Practices (Arun Ravindran)

- http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python