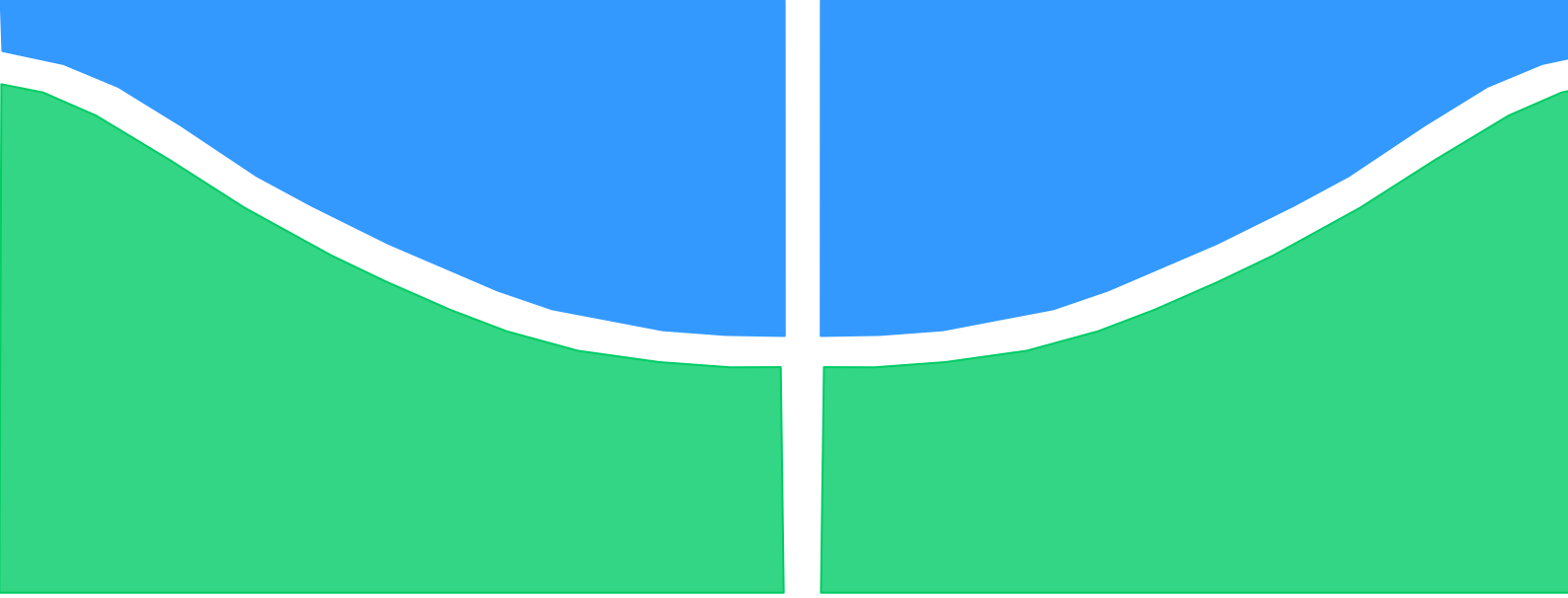


**Universidade de Brasília - UnB**
**Faculdade UnB Gama - FGA**
**Curso de Engenharia de Software**

# MONITORAÇÃO DA QUALIDADE DE PRODUTO NAS CONTRATAÇÕES DE SOLUÇÕES DE TI DA ADMINISTRAÇÃO PÚBLICA FEDERAL

**Autora: Kamilla Holanda Crozara**
**Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Jr.**

**Brasília, DF**
**2014**

**KAMILLA HOLANDA CROZARA**

# MONITORAÇÃO DA QUALIDADE DE PRODUTO DAS CONTRATAÇÕES DE SOLUÇÕES DE TI DA ADMINISTRAÇÃO PÚBLICA FEDERAL

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

Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Jr.

**Brasília, DF**
**2014**

Holanda Crozara, Kamilla.
Monitoração da qualidade de produto nas contratações de soluções de TI da Administração Pública Federal / Kamilla Holanda Crozara. Brasília: UnB, 2014. 103 p. : il. ; 29,5 cm.

Monografia (Graduação) – Universidade de Brasília Faculdade do Gama, Brasília, 2013. Orientação: Luiz Carlos Miyadaira Ribeiro Jr.

1. Contratações. 2. Qualidade de software. 3. Palavra chave3 I. Dr. Luiz Carlos Miyadaira Ribeiro Jr. II. Monitoração da qualidade de produto nas contratações de soluções de TI da Administração Pública Federal.

CDU Classificação


**MONITORAÇÃO DA QUALIDADE DE PRODUTOS NAS CONTRATAÇÕES DE SOLUÇÕES DE TI DA ADMINISTRAÇÃO PÚBLICA FEDERAL**

**Kamilla Holanda Crozara**

Monografia submetida como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software da Faculdade UnB Gama - FGA, da Universidade de Brasília, em 25/06/2014 apresentada e aprovada pela banca examinadora abaixo assinada:

**Prof. Dr: Luiz Carlos Miyadaira Ribeiro Jr., UnB/ FGA**
Orientador

**Prof. Dra.: Rejane M. da Costa Figueiredo, UnB/ FGA**
Membro Convidado

**Prof. Dr.: Sérgio de A. Andrade de Freitas, UnB/ FGA**
Membro Convidado

Brasília, DF
2014

## AGRADECIMENTOS

Agradeço primeiramente aos meu pais, Marluce Holanda Cavalcante e Emerson Crozara pelo dom da vida e por todo o apoio e dedicação despendidos à mim, especialmente durante o árduo período de graduação durante o qual enfrentei os maiores desafios. Agradeço especialmente à minha mãe por todo o incentivo, por ter sido meu porto seguro, meu exemplo e meu consolo nos momentos difíceis. Agradeço também a toda a minha família, em especial à minha avó Maria Bernadete de Holanda por ter me apoiado ao longo do curso, especialmente durante os primeiros anos. Sem vocês essa jornada não teria sido possível.

Agradeço ao meu orientador professor Dr. Luiz Carlos Miyadaira Ribeiro Júnior, com o qual tenho trabalhando desde o terceiro semestre, pela confiança em mim depositada, pela paciência, pelo incentivo, pelo apoio, pelas discussões propostas e principalmente por ter estendido o meu aprendizado a respeito do desenvolvimento de software muito além dos limites da sala de aula.

Agradeço a todos os professores da Faculdade do Gama (FGA) por toda a dedicação e determinação ao enfrentar os desafios da criação de um novo campus. Agradeço em especial à professora Dra. Rejane Maria da Costa Figueiredo que foi minha incentivadora na busca por novos desafios além dos propostos em sala de aula. Agradeço também por todos os seus esforços para a fundação do curso de Engenharia de Software da Universidade de Brasília, o primeiro no nível de graduação do Brasil, que hoje disponibiliza educação de alto nível a centenas de estudantes. Poucos sabem das suas batalhas para a fundação de um curso de qualidade alinhado aos padrões de referência para a Engenharia de Software.

Agradeço à Hannah Presley dos Santos por ter me acompanhado durante todas as etapas da minha graduação, por todo o companheirismo, amor e paciência (sobretudo nos dias que antecederam a entrega deste trabalho), por compartilhar dos meus sonhos e por ser fonte de inspiração para a construção de um grande futuro.

Agradeço aos meus amigos e companheiros de Engenharia de Software, em especial ao Eduardo Pinto Barbosa, ao Charles Daniel de Oliveira, ao Rodrigo Siqueira de Melo e ao Maylon Felix de Brito por todo o apoio e companheirismo durante os estudos, entregas de trabalho e madrugadas em claro.

Por fim, agradeço também à equipe do Centro de Qualidade e Testes de Software por proporcionar tantos momentos de aprendizado e mostrar tanta motivação ao enfrentar os desafios propostos.

"Mesmo que o futuro lhe pareça distante, ele está começando neste exato momento". (Mattie J.T. Stepanek)

## RESUMO

O setor público é cada vez mais dependente das contratações de serviços de TI. A contratação permite que a administração concentre seus esforços nas atividades de gestão e não nas tarefas executivas, executando assim suas atividades primárias de forma mais adequada. As contratações de TI envolvem recursos públicos significativos e esforços de diversas unidades administrativas e devem ser bem concebidas, executadas e gerenciadas para que as necessidades dos órgãos e entidades sejam atendidas. A verificação da qualidade do código-fonte do produto resultante das contratações é um dos fatores chave para o sucesso das contratações de serviços, como o de fábrica de software, visto que é necessária a verificação dos critérios de aceitação definidos em contrato. O objetivo deste trabalho foi definir técnicas para um processo de verificação de software aplicado ao contexto da Administração Pública Federal. A pesquisa foi descritiva, com apoio da técnica de estudo de caso. Os critérios de qualidade adotados estão embasados na ISO/IEC 25000. A ferramenta SonarQube foi selecionada para a manutenção dos critérios de qualidade de código e para a análise do código-fonte dos produtos resultantes das contratações. Foi criado um Perfil de Qualidade com a definição de critérios de qualidade para a avaliação dos produtos de software resultantes das contratações. Como trabalhos futuros, objetiva-se a validação e refinamento dos critérios de qualidade selecionados e também a criação de plug-ins para a ferramenta SonarQube para integrar a análise dos critérios de qualidade ao painel da ferramenta SonarQube. E também a estruturação de critérios de aceitação para editais baseados nos critérios de qualidade definidos para o SonarQube.

**Palavras-chave:** Contratação de Fábrica de Software. Qualidade de Software. Análise estática de código fonte.

## ABSTRACT

The public sector is increasingly dependent on the outsourcing of IT services. The outsourcing allows the Federal Public Administration to focus its efforts on management activities and not in executive tasks, thus performing their primary activities more appropriately. IT outsourcing involves significant public resources and efforts of various administrative units and should be well designed, executed and managed so that the needs of agencies and entities are met. The quality check of the source code of the product resulting from the IT outsourcing is one of the key factors for the success of services agreements, such as software factory, since the verification of acceptance criteria defined in the contract is required. The goal of this work was to define techniques for software verification process applied to the context of the Federal Public Administration. The research was descriptive with support of the technical case study. The quality criterias adopted are based in ISO/IEC 25000. The SonarQube was selected for maintanance of quality criterias and to analyze the source code of products resulting from outsourcing. As future work, we aim at validating and refining the quality criterias and the creation of plug-ins for SonarQube tool to integrate the analysis of the quality criterias to the SonarQube dashboard.

**Keywords:** Outsourcing. Software Quality. Static analysis.

# LISTA DE FIGURAS

# LISTA DE TABELAS

## LISTA DE ABREVIATURAS

| | |
|---|---|
| **APF** | Administração Pública Federal |
| **CobIT** | Control Objectives for Information and Related Technology |
| **DISIS** | Divisão de Sistemas |
| **MCT** | Ministério da Ciência e Tecnologia |
| **PBQP** | Programa Brasileiro da Qualidade e Produtividade em Software |
| **PCSTI** | Processo de Contratação de Serviços de TI |
| **PDTI** | Plano Diretor de Tecnologia da Informação |
| **SATE** | Static Analysis Tool Exposition |
| **SEPIN** | Secretaria de Política de Informática do MCT |
| **SLTI** | Secretaria de Logística e Tecnologia da Informação |
| **SquaRE** | Software Quality Requirements and Evaluation |
| **TCU** | Tribunal de Contas da União |
| **TI** | Tecnologia da Informação |

# SUMÁRIO

# 1. INTRODUÇÃO

Neste capítulo apresenta-se o contexto do trabalho, os problemas identificados, os objetivos , as justificativas e a metodologia adotada.

## 1.1. CONTEXTO

A atividade de contratação no Brasil não pode mais ser tratada somente como uma abordagem que visa exclusivamente à redução de custos. No cenário atual, as contratações são parte integrante da estratégia das organizações, principalmente das organizações da Administração Pública Federal (APF), e tornou-se uma abordagem estratégica que adiciona valor ao negócio e amplia a melhoria dos processos internos e dos serviços prestados (PRADO; CRISTOFOLI, 2012). Nos últimos anos a APF passou a ter uma maior preocupação em relação às contratações de software realizadas. Em um período de dezesseis anos (1995 a 2010) percebeu-se um número crescente de acórdãos e decisões do Tribunal de Contas da União relacionadas ao tema (CRUZ; ANDRADE; FIGUEIREDO, 2011)

A APF é uma das principais contratantes de serviços de TI no Brasil, contribuindo assim, para o fortalecimento do crescimento desta atividade. Essa decisão está alinhada com o Decreto-Lei nº 200/67, art. 10, § 7º (BRASIL, 1967), que estabelece que a administração deve concentrar seus esforços nas atividades de gestão e não nas tarefas executivas. Essas contratações envolvem recursos públicos significativos e esforços de diversas unidades administrativas, portanto devem ser bem concebidas, executadas e gerenciadas para que as necessidades dos órgãos e entidades sejam atendidas (BRASIL, 2012a). O Decreto nº 2.271/97 definiu que as atividades de informática devem ser preferencialmente contratadas de terceiros. Posteriormente, a Instrução Normativa MP/SLTI Nº 04 de 2008 (BRASIL, 2010b) (atualizada pela IN-MP/SLTI Nº 04 de 2010) regulamentou a execução indireta de serviços de TI, à luz da legislação corrente, como a Lei 8.666 de 1993 (BRASIL, 1993), e da jurisprudência sobre o assunto (CRUZ; ANDRADE; FIGUEIREDO, 2011).

Segundo dados recentes da Secretaria de Logística e Tecnologia da Informação (SLTI) (BRASIL, 2012b), entre janeiro e novembro de 2012 foram realizadas, por órgãos da APF, 3.331 licitações públicas voltadas para a aquisição de bens e serviços de tecnologia da informação (TI). Essas licitações movimentaram

recursos da ordem de R$ 1,6 bilhão, que representaram cerca de 5% do total das aquisições da administração direta, autárquica e fundacional. Desse total de licitações públicas, os bens representaram 86% e os serviços 14% das aquisições de TI. Além disso, as compras de TI por dispensa ou inexigibilidade de licitação movimentaram R$ 1,2 bilhão. O Distrito Federal foi responsável pelas maiores aquisições com um montante de R$ 865,8 milhões que corresponde a 55% do total gasto com TI no referido ano (BRASIL, 2012b).

Existe uma dependência estratégica dos recursos de TI, ou seja, a TI agora é um fator crítico para o desenvolvimento e a manutenção dos órgãos da APF(BRASIL, 2012a). Embora a contratação de serviços de TI tenha papel importante na estratégia organizacional, há muitos riscos que podem frustrar seus resultados e impactar negativamente a governança de TI (HEFLEY; LOESCHE, 2006). Wright (WRIGHT, 2005) analisou vinte e três riscos associados à contratação de serviços de TI e concluiu que os mais relevantes são: (i) riscos de segurança da informação; (ii) riscos de dependência do fornecedor de serviços; e (iii) riscos de disputa legal. No caso da APF, existem também os riscos relacionados ao descumprimento da legislação de licitações e contratos, que poderiam ser, por exemplo: impugnação de procedimento licitatório ou suspensão da assinatura do contrato, causando atraso na contratação (Lei 8.666/1993, art. 41, § 1º); suspensão ou rompimento de contratos considerados ilegais (Lei 8.443/1992, art. 45 (BRASIL, 1992)) perdas orçamentárias; paralisação de projetos importantes calcados em TI; ressarcimento, pelos gestores, de prejuízos quantificados (Lei 8.443/1992, arts. 12 e 19 (BRASIL, 1992)); processo criminal, nos casos previstos na Lei 8.666/1993, arts. 89 a 99 (CRUZ; ANDRADE; FIGUEIREDO, 2011).

Nos últimos anos o Tribunal de Contas da União (TCU) tem efetuado uma série de trabalhos de fiscalização nos quais foram identificados diversos problemas na condução das contratações de soluções e serviços de TI pelos órgãos e entidades da APF, bem como nos respectivos contratos (BRASIL, 2012a). Muitos desses problemas são decorrentes de deficiências no planejamento dessas contratações (BRASIL, 2012a). Segundo Cruz, Andrade e Figueiredo (CRUZ; ANDRADE; FIGUEIREDO, 2011), a contratação na APF é uma das principais responsáveis por inconformidades contratuais e também pelo não aproveitamento dos produtos entregues, com destaque para os sistemas de informação. Os autores

destacam também que tais problemas são causados pela falta de gerenciamento e fiscalização de contratos, o que leva muitas vezes a não conformidade com os objetivos da contratação.

## 1.2. PROBLEMA

Apesar das contratações de serviços de TI serem essenciais para os órgãos da APF, ainda são poucos os estudos a respeito da qualidade do produto de software produzido através desse modelo. Algumas pesquisas vêm sido realizadas pelo Ministério da Ciência e Tecnologia (MCT) e pelo Tribunal de Contas da União (TCU), porém os conceitos de qualidade de produto não são totalmente abrangidos.

O Programa Brasileiro da Qualidade e Produtividade em Software (PBQP-Software) foi lançado originalmente em 1993 e tem como uma de suas tarefas a obtenção de dados e indicadores sobre a evolução da qualidade no setor de software e serviços de TI no Brasil. A última pesquisa foi publicada em 2010 e a população chegou a 2.587 organizações com atividades de software no ano de 2009. A população é constituída por aqueles que mantêm laços estreitos com as entidades atuantes no setor e/ou com o governo, que já possuem certa visibilidade e/ou que, por vontade própria entraram em contato com a Secretaria de Política de Informática do MCT (SEPIN) manifestando interesse em participar da pesquisa. É também composta por empresas de diversos portes e com sedes localizadas em diversas unidades da federação, parte significativa tem a atividade de software como principal fonte de receita (BRASIL, 2010a).

Na edição de 2010, no que se refere à Qualidade dos Produtos de Software, a Pesquisa de Qualidade no Setor de Software Brasileiro buscou identificar como estava a adoção das normas de qualidade de produto publicadas pela ISO/IEC e ABNT. Procurou-se identificar a utilização das Normas NBR ISO/IEC 9126 (ISO/IEC, 2003), NBR ISO/IEC 25000 (ISO/IEC, 2014), NBR ISO/IEC 12119 (ISO/IEC, 1998) e NBR ISO/IEC 14598 (ISO/IEC, 2001) e constatou-se pela pesquisa que tais normas técnicas são pouco utilizadas pelas organizações (Tab. 1).

Ao todo, 273 organizações responderam a esta questão, o que representa 79,6% do total de respondentes da pesquisa e 252 organizações afirmaram não utilizar nenhuma das normas listadas. Isto representa 89% do total de respondentes da questão. A NBR ISO/IEC 9126 (ISO/IEC, 2003), agora substituída pela série

ISO/IEC 25000 (ISO/IEC, 2014), é usada por 5,7% das organizações e ainda é a mais conhecida dentre as normas citadas na questão. É um índice baixo, considerando que a norma NBR ISO 9126 (ISO/IEC, 2003) está disponível no Brasil e foi publicada pela ABNT desde 1996, quando ainda era chamada de NBR 13596. As demais normas relacionadas são utilizadas por uma pequena quantidade de organizações (BRASIL, 2010a).

Tabela 1 - Distribuição das organizações de acordo com a utilização de Normas para requisitos de qualidade de software. Fonte: Pesquisa de Qualidade no Setor de Software Brasileiro 2009 (BRASIL, 2010a).

| **Normas de Requisitos de Qualidade de Produto de *Software*** | **Quantidade de Organizações** | **% das organizações** |
|---|---|---|
| NBR ISO/IEC 25000 | 6 | 2,1 % |
| NBR ISO/IEC 9126 | 16 | 5,7 % |
| NBR ISO/IEC 12119 | 4 | 1,4 % |
| NBR ISO/IEC 14598 | 5 | 1,8 % |
| Não utiliza | 252 | 89,0 % |

*Nota (1): 273 organizações respondentes, de um total de 343 organizações (79,6%). Nota (2): Questão permitia múltiplas respostas.*

O Levantamento de Governança de Tecnologia da Informação (TI) - Ciclo 2012 (BRASIL, 2012c) apresenta dados a respeito do gerenciamento, fiscalização e monitoramento nas contratações dos órgãos da APF. Ao todo, trezentas e trinta e sete instituições responderam ao questionário, as quais, tendo em vista uma melhor avaliação do comportamento dos dados, foram agrupadas nos seguintes segmentos da APF: EXE-Dest, abrangendo as empresas públicas federais e as sociedades de economia mista; EXE-Sisp, abrangendo as instituições que fazem parte do Sistema de Administração dos Recursos de Informação e Informática (SISP); JUD, abrangendo as instituições que fazem parte do Poder Judiciário; LEG, abrangendo as instituições que fazem parte do Poder Legislativo; e MPU, abrangendo as instituições que fazem parte do Ministério Público da União (MPU). Para o presente trabalho foram analisadas as questões referentes às contratações de serviços de TI relacionadas ao monitoramento e controle da contratação (Tab. 2).

Observa-se que, embora no ano de 2012 64% dos órgãos declararam adotar métricas objetivas para a mensuração de resultados do contrato e 89% declararem realizar os pagamentos dos resultados entregues e aceitos em função da mensuração planejada, a pesquisa não trata do tipo de métricas adotadas para a mensuração desses resultados. Além disso, apenas 21% declararam a formalização

do processo de gestão de contratos e 6% declararam que o cumprimento do processo de gestão de contratos é medido e controlado. Diante desse cenário, pode-se perceber que muitos gestores não possuem elementos para medição de qualidade, supervisão e revisão, como recomendado pelo framework CobIT 5 (ITGI, 2012).

Tabela 2 - Resultado da análise das questões referentes às contratações de serviços de TI relacionadas ao monitoramento e controle da contratação. Fonte: Levantamento de Governança de Tecnologia da Informação (TI) - Ciclo 2012 (BRASIL, 2012c)

| **Questão** | **Item** | **Resultado Geral** | |
|---|---|---|---|
| | | **2012** | **2010** |
| 5.7. Em relação às contratações de serviços de TI: | Q57g - são adotadas métricas objetivas para mensuração de resultados do contrato. | 64% | - |
| | Q57h - os pagamentos são feitos em função da mensuração objetiva dos resultados entregues e aceitos. | 89% | - |
| 5.9. Em relação à fase de gestão dos contratos de TI, em qual das descrições abaixo a instituição se encaixa melhor? | 3 - Além do item anterior, o processo de gestão de contratos é formalizado (aprovado e publicado) em norma própria e de cumprimento obrigatório. | 21% | 20% |
| | 4 - Além do item anterior, o cumprimento do processo de gestão de contratos publicado é medido e controlado. | 6% | 7% |

Durante o desenvolvimento da primeira parte deste trabalho, realizou-se entrevistas com órgãos da APF que contavam com uma empresa pública federal, a Embrapa, com um banco público, a Caixa Econômica Federal, e com um membro do poder Executivo, o Ministério das Comunicações. As entrevistas foram realizadas entre janeiro e fevereiro de 2013 (Tab. 3).

Tabela 3 - Resumo em ordem cronológica das entrevistas realizadas

| **Organização** | **Tipo** | **Respondente** | **Tempo médio** |
|---|---|---|---|
| Embrapa | Empresa pública federal | Coordenador de TI | 1 hora e 44 minutos |
| Caixa | Banco público | Coordenador de TI | 1 hora e 52 minutos |
| Ministério das Comunicações | Membro do poder Executivo | Coordenador de TI | 48 minutos |

Os dados das entrevistas foram analisados utilizando o método Grounded Theory criado por Glaser e Strauss em 1967 (GLASER; STRAUSS, 1967). O

resultado da análise utilizando-se o método *Grounded Theory* é a emergência de uma teoria sobre os dados analisados, para esta pesquisa foi escolhida somente uma das questões do questionário aplicado durante as entrevistas. A questão analisada foi a seguinte: “No processo de desenvolvimento de software da contratante, como são avaliados os critérios de qualidade de produto? (como por exemplo, atributos de qualidade interna e externa de software e atributos de qualidade em uso que são características da ISO 9126)”. Como resultado da análise dos dados das entrevistas constatou-se que a verificação do produto é ineficiente nas contratações de TI para desenvolvimento de software realizado pelos órgãos entrevistados. Ou seja, a grande maioria das organizações investigadas não utilizam padrões e normas para a qualidade do produto e muitas vezes essas organizações sequer conseguem avaliar as características de qualidade definidas pelas normas.

Percebe-se que a verificação da qualidade de produto de software nas contratações de TI da APF ainda é limitada. Embora seja recomendado pela Instrução Normativa MP/SLTI Nº04 que sejam estabelecidos critérios de aceitação que abrangem métricas, indicadores e valores mínimos, quando se trata de desenvolvimento, apesar dos casos em que é feito o monitoramento e controle de acordo com o planejamento, as medições utilizadas podem não corresponder ou não serem adequadas para a medição de qualidade de software.

## 1.3. OBJETIVOS

**Objetivo geral**: Definir técnicas, processos e ferramentas que auxiliem na garantia da qualidade de código-fonte do produto de *software* objeto das contratações de serviços de TI.

**Objetivos específicos**:

1. Identificação métricas aplicáveis de verificação da qualidade na APF;
2. Escolha de *softwares* desenvolvidos no contexto da APF para realização de estudo de caso;
3. Desenvolvimento e/ou customização de ferramentas para auxiliar a verificação do produto de software resultante das contratações da APF.

## 1.4. JUSTIFICATIVA

As contratações de serviços de TI envolvem gastos elevados de dinheiro público e alta complexidade de gestão. Deve-se certificar que os serviços contratados são de qualidade e, no caso de manutenção e desenvolvimento de software, deve-se dar grande atenção para a qualidade do produto de software. Os softwares produto das contratações muitas vezes são mantidos pelo órgão contratante ou até mesmo por outra empresa diferente da empresa contratada para o desenvolvimento. Segundo Pressman (2011), a manutenção de software existente pode ser responsável por mais de 60% de todo o esforço despendido por uma organização de desenvolvimento, e a porcentagem continua a crescer à medida que mais software é produzido. Anda segundo Pressman (2011) apenas 20% de todo o trabalho de manutenção é gasto consertando erros, o restante do tempo é gasto adaptando sistemas existentes a modificações no seu ambiente externo, fazendo melhorias solicitadas por usuários e submetendo uma aplicação a reengenharia.

O gestor deve criar elementos para medição de qualidade, supervisão e revisão. É necessário definir, planejar e implementar, controlar e registrar medidas para a fiscalização e o cumprimento da garantia da qualidade (ITGI, 2012). Os gestores devem estar capacitados e habilitados para conduzir novas formas de gestão e processos, bem como realizar acompanhamentos constantes sobre a gestão dos serviços prestados, a fim de garantir bons níveis de qualidade e produtividade (CRISTOFOLI, 2011).

A qualidade do produto de software e o desempenho dos serviços executados na contratação devem ser bem definidos e controlados pelo órgão contratante, sendo essencial que existam práticas apoiadas por normas e padrões para garantir a qualidade do produto que está sendo recebido. Lewis (1999) afirma que é preciso controlar a qualidade dos serviços estabelecendo-se métricas a partir de um padrão de qualidade. Para prevenir que custos, entregas e implementação de funcionalidades fujam do controle o monitoramento dos projetos realizados por contratações deve ser baseado em conhecimento técnico do sistema que está sendo desenvolvido (KUIPERS; VISSER; VRIES, 2007). Assim, é possível garantir a qualidade dos produtos entregues pelos fornecedores de serviços, o bom uso do dinheiro público e o controle das atividades de gestão de contratos. Essas atividades de monitoramento e controle são fundamentais na busca de uma maior eficiência e

um menor custo para o Estado, na melhoria na qualidade dos serviços prestados à sociedade, e também para o aumento da competitividade do país no comércio global.

## 1.5. METODOLOGIA

Quanto ao seu objetivo a pesquisa se classifica como descritiva, pois focou na investigação das técnicas de avaliação da qualidade do produto nas contratações de soluções de TI da Administração Pública Federal. Quanto à sua natureza a pesquisa é considerada aplicada e qualitativa, pois realizou a análise de ferramentas para a monitoração da qualidade de produto de software utilizando softwares coletados no Portal do Software Público e fornecidos por um órgão da APF que tem parceria com a Universidade de Brasília. Quanto aos procedimentos técnicos a pesquisa é considerada bibliográfica e documental e fez também o uso do estudo de caso (GIL, 2008).

A pesquisa bibliográfica foi baseada nas principais bases de conhecimento científico e também nos documentos publicados por órgãos fiscalizadores como o Tribunal de Contas da União que tratam das contratações de serviços de TI na APF. A pesquisa bibliográfica foi realizada com o objetivo de entender como as contratações são realizadas na APF e quais os níveis de controle de qualidade são aplicados. O estudo de caso foi realizado para se obter dados reais a respeito da qualidade dos produtos desenvolvidos na APF que foram obtidos ou não através de contratações de soluções de TI. Além disso, com a realização do estudo de caso foi possível estabelecer parâmetros para a verificação da qualidade do produto de software fruto das contratações.

O plano metodológico adotado foi dividido em seis fases: Planejamento; Análise e Seleção de Ferramentas; Seleção de Software para Análise; Coleta de Dados, Análise e Interpretação dos Dados e Redação dos Resultados .

**Planejamento**: definição dos objetivos da pesquisa, seleção do tipo de metodologia adotada e definição da técnica de coleta de dados;

**Análise e Seleção de Ferramentas**: análise de ferramentas de verificação de código e seleção de ferramentas adequadas às contratações de TI;

**Seleção de Softwares para Análise**: seleção dos softwares a serem analisados pelas ferramentas selecionadas na fase anterior;

**Coleta de Dados**: análise dos softwares selecionados utilizando as ferramentas de verificação de código e armazenamento dos dados das análises em base de dados adequada;

**Análise e Interpretação dos Dados**: comparação dos resultados da verificação do código-fonte dos softwares selecionados e análise dos problemas apontados durante a verificação realizada pela ferramenta selecionada;

**Redação dos Resultados**: descrição do estudo de caso e dos parâmetros selecionados para a verificação de qualidade de código nas contratações de soluções de TI da APF. Esses resultados são apresentados nos capítulos seguintes deste trabalho.

## 1.6. ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis capítulos. Neste capítulo situam-se: o contexto do trabalho, o problema, os objetivos, as justificativas, e a metodologia adotada.

No *Capítulo 2 – Contratação de Fábrica de Software*, são apresentados os conceitos relativos às contratações de serviços de desenvolvimento de software pela Administração Pública Federal. Apresenta-se a Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b), o Modelo de Contratação de Soluções de TI e o Processo de Contratação de Serviços de Tecnologia da Informação para Organizações Públicas Federais.

No *Capítulo 3 – Qualidade de Software*, são apresentados os conceitos relativos à qualidade de software com foco na qualidade do produto de software. É Uma síntese da ISO/IEC 25000, são discutidos conceitos relacionados à verificação de software, à análise estática de código-fonte e em seguida são analisadas as ferramentas que podem ser utilizadas para executar a análise estática de código-fonte.

No *Capítulo 4 – Estudo de caso* é apresentado o estudo de caso realizado utilizando a ferramenta SonarQube para análise estática do código-fonte de projetos do Portal do Software Público Brasileiro e do órgão da APF.

No *Capítulo 5 – Definição do Perfil de Qualidade para as contratações de soluções de TI*, o resultado do estudo de caso foi utilizado para a definição do conjunto de violações de regra que forma o Perfil de Qualidade a ser analisado nas contratações de serviços de desenvolvimento de software no contexto do órgão da APF participante do estudo de caso.

No *Capítulo 6 – Conclusão e Trabalhos Futuros* apresentam-se as conclusões, contribuições obtidas na realização da pesquisa e os trabalhos a serem desenvolvidos futuramente.

# 2. CONTRATAÇÕES DE FÁBRICA DE SOFTWARE

Neste capítulo são apresentados conceitos sobre contratações de serviços de desenvolvimento de software pela Administração Pública Federal. São caracterizados conceitos relativos à contratação de serviços de TI e fábrica de software, apresenta-se também uma síntese Instrução Normativa MP/SLTI Nº04, o Modelo de Contratação de Soluções de TI e o Processo de Contratação de Serviços de TI para Organizações Públicas Federais.

## 2.1. CONTRATAÇÃO

Em seu sentido mais amplo, contratação significa a aquisição de qualquer produto ou serviço de outra empresa. No geral, as empresas que decidem por terceirizar produtos e serviços não podem ou não querem produzi-los por si mesmas (TURBAN; MCLEAN; WETHERBE, 2004). Por sua vez, Beal (2005) descreve que a contratação é a transferência de todas as atividades que não são parte da competência estrutural básica da empresa e tal transferência é feita com o intuito de reduzir custos, aumentar a produtividade da empresa e aumentar a qualidade e produtividade dos serviços. Também para Araújo (2008), a contratação pode ser entendida como o ato de passar adiante a responsabilidade de determinadas atividades ou conjunto delas, permitindo que outras empresas assumam o controle de atividades não essenciais para o negócio.

Deve-se ressaltar que uma empresa contratada terá total responsabilidade sob a atividade que irá executar e que à contratante interessa somente o resultado ou produto objeto do contrato. A contratação é formalizada geralmente por meio de contrato e a natureza jurídica da contratação dependerá de um contrato utilizado ou da combinação de vários deles. O contrato pode ser de prestação de serviços, de empreitada, de locação de serviços, entre outros (MARTINS, 2005). Devido à sua natureza, o contrato pode ser regulado pelo direito civil, comercial ou administrativo e tem como escopo a produção de bens ou a prestação de serviços. No Brasil as contratações públicas são reguladas principalmente pela Lei 8.666/1993 (BRASIL, 1993).

A contratação pode ser uma estratégia de gestão eficiente se aplicada da maneira correta. Para isso, vários aspectos devem ser observados, as contratações devem ser bem concebidas, executadas e gerenciadas, pois demandam mecanismos de controle específicos, envolvem recursos públicos significativos e

também esforços de diversas unidades administrativas (BRASIL, 2012a). Para apoiar as contratações de soluções de TI no âmbito da APF o Tribunal de Contas da União (TCU) apresenta o Guia de boas práticas em contratações de soluções de TI (BRASIL, 2012a) voltado aos órgãos e entidades da APF com orientações referentes aos riscos do planejamento das contratações de soluções de TI baseadas na legislação, na jurisprudência e nas melhores práticas de mercado.

### 2.1.1. Soluções de TI

O inciso IX do art. 2º da Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b) define solução de Tecnologia da Informação como o conjunto de bens e serviços de TI e automação que se integram para o alcance dos resultados pretendidos com a contratação, de modo a atender à necessidade que a desencadeou. Segundo o Guia de boas práticas de contratação de TI (BRASIL, 2012a), no caso da contratação do serviço de um sistema de informação, a solução de TI pode englobar entre outros elementos:

- os softwares do sistema, devidamente documentados e com evidências de que foram testados;
- as bases de dados do sistema, devidamente documentadas;
- o sistema implantado no ambiente de produção do órgão;
- a tecnologia do sistema transferida para a equipe do órgão, que deve ocorrer ao longo de todo o contrato;
- as rotinas de produção do sistema, devidamente documentadas e implantadas no ambiente de produção do órgão;
- as minutas dos normativos que legitimem os atos praticados por intermédio do sistema;
- o sistema de indicadores de desempenho do sistema implantado, que pode incluir as atividades de coleta de dados para gerar os indicadores, fórmula de cálculo de cada indicador e forma de publicação dos indicadores;
- os scripts necessários para prover os atendimentos relativos ao sistema por parte da equipe de atendimento aos usuários;

- a capacitação dos diversos atores envolvidos com o sistema, que pode envolver treinamentos presenciais e a distância;
- o lançamento do sistema no âmbito do órgão ou externamente, para que todos os interessados internos ou externos ao órgão tenham ciência da existência do sistema e das suas principais funcionalidades, de modo que o investimento feito nele gere retorno a esses interessados;
- o serviço contínuo de suporte técnico ao sistema;
- o serviço contínuo de manutenção do sistema.

## 2.2. CONTRATAÇÕES DE SOFTWARE NA ADMINISTRAÇÃO PÚBLICA FEDERAL

Atualmente os serviços de TI na APF têm sido em sua grande maioria prestados por meio de contratações. Isso ocorre devido a determinações legais como as previstas no Decreto-Lei nº 200/6 (BRASIL, 1967), Decreto nº 2.271/97 (BRASIL, 1997) e a Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b) e devido ao modelo de gestão governamental adotado. As contratações de serviços podem ser consideradas um dos principais processos para a consolidação de boa governança (BARBOSA et al., 2006). A norma ABNT NBR ISSO/IEC 38500:2009 (ABNT, 2009) descreve os princípios que devem governar a boa governança de TI, entre esses princípios está o de contratações de TI que quando feitas por razões válidas, baseadas em análise apropriada e contínua e em tomadas de decisões claras e transparentes propiciam o equilíbrio entre benefícios, oportunidades, custos e riscos, de curto e longo prazo.

Deve-se observar que atividades como a implantação de um sistema de informação no ambiente de produção devem ser executadas por servidores públicos. Assim, evita-se a ocorrência de eventos indesejados como acesso não autorizado aos dados em produção e alteração indevida de software ou de dados naquele ambiente. De acordo com o caput do art. 8º da Lei 8.666/1993 (BRASIL, 1993), as contratações devem ser planejadas no todo, entretanto de acordo com o § 1º do art. 23 da Lei 8.666/1993 (BRASIL, 1993), como regra, as contratações têm que ser divididas em tantas parcelas quanto possível, desde que seja técnica e economicamente viável. Portanto, caso seja considerado técnica e economicamente

divisível, cada parcela ou parte da solução poderá corresponder ao objeto de uma licitação separada (BRASIL, 2012a).

A aplicação das normas que regulam as licitações públicas deve ser feita de forma que haja ampliação da disputa entre os interessados, desde que informados no edital e não comprometa o interesse da administração, o princípio da isonomia, a finalidade e a segurança da contratação. Ainda de acordo com a Lei 8.666/1993 (BRASIL, 1993), a celebração de contratos com terceiros na APF deve ser necessariamente precedida de licitação, ressalvadas as hipóteses de dispensa e de inexigibilidade de licitação. Vários fatores devem ser previstos antes da contratação na APF, a Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b) determina que a área demandante justifique, planeje os serviços e avalie a necessidade de acordo com as justificativas e somente após essa avaliação repasse o serviço para terceiros. Portanto, espera-se que essas contratações de TI sejam bem planejadas e controladas pelos servidores públicos responsáveis.

### 2.2.1 Fábrica de Software

Na APF as contratações de TI são realizadas principalmente entre os órgãos da APF e fábricas de software. O conceito de Fábrica de Software foi proposto inicialmente por Bemer (1965) em meados da década de 60. Bemer (1965) considera fábrica de software como uma combinação de ferramentas padronizadas, interface baseada em computadores e uma base de dados com histórico para o controle financeiro e gerencial.

Para a SOFTEX (SOFTEX, 2013) a fábrica de software tem como função transformar o desenvolvimento em um processo padronizado, aumentando assim a produtividade e a eficiência. Segundo Costa (2003) fábrica de software é uma proposta de solução singular para as contratações e se caracteriza por uma estrutura que é complementar à das organizações contratantes, o que amplia a capacidade de atendimento às demandas de serviços de software.

Castor (2004) descreve como característica principal do modelo de Fábrica de Software a criação de um ambiente de desenvolvimento produtivo de produtos de software com qualidade e baixo custo, para isso, são utilizadas técnicas da engenharia de produção em série, buscando a solução de problemas encontrados neste ambiente como a baixa produtividade e alto custo na produção de sistemas.

As fábricas de software buscam a melhoria dos seus processos com o objetivo de otimizar a utilização dos recursos e diminuir os seus custos operacionais.

### 2.2.2. Instrução Normativa MP/SLTI Nº04

As várias falhas e ineficiências de governança apontadas nos acórdãos nº 1.558/2003 e 786/2006, ambos do Plenário, direcionaram, dentre outras coisas, a recomendação à SLTI para expedição de norma sobre o processo de licitação e contratação de serviços de TI. A partir dessas recomendações, a Secretaria de Logística e Tecnologia da Informação (SLTI) publicou a Instrução Normativa MP/SLTI Nº04, de 12 de novembro de 2010 (atualizada em 14/02/2012) (BRASIL, 2010b), que dispõe sobre o processo de contratação de Soluções de Tecnologia da Informação pelos órgãos integrantes do Sistema de Administração dos Recursos de Informação e Informática - SISP do Poder Executivo Federal, contratação que deverá ser precedida de planejamento elaborado em consonância com o Plano Diretor de Tecnologia da Informação (PDTI), o qual deverá estar alinhado ao Plano Estratégico da organização.

A Instrução Normativa MP/SLTI Nº04 possui 32 artigos e está dividida em três Capítulos tratando, o primeiro, sobre as Disposições Gerais, o segundo Do Processo de Contração, e o terceiro Das Disposições Finais. Ainda segundo esta norma, não poderão ser objetos de contratação:

- Todo o conjunto de serviços de TI em um único contrato;
- Mais de uma solução de TI em um único contrato;
- Gestão de processos de TI, incluindo a gestão da segurança da informação.

No Capítulo I pode ser encontradas: a definição dos termos que serão utilizados ao longo do texto da Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b) (artigo 2º); orientações para elaboração do Plano Diretor de Tecnologia da Informação - PDTI (artigos 3º e 4º); e algumas vedações (artigos 5º, 6º e 7º). O Capítulo II está divido em três seções que contemplam todo o procedimento para execução das fases de Planejamento da Contratação, Seleção de Fornecedor e Gerenciamento do Contrato. O Capítulo III, por sua vez, trata Das Disposições Finais indicando: a utilização subsidiária da Instrução Normativa nº 2, de 30 de abril de

2008, (artigo 28); o apoio das Áreas de Compras, Licitações e Contratos nas atividades de contratação (artigo 29); a aplicação da norma em casos de prorrogações contratuais (artigo 30); a data de entrada em vigor da própria Instrução (artigo 31); e a revogação da Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b), de 19 de maio de 2008 (artigo 32).

Especificamente sobre os critérios de acompanhamento e aceite dos produtos, o art. 15 § 2º define que a Estratégia da Contratação será elaborada a partir da Análise de Viabilidade da Contratação e do Plano de Sustentação, contendo no mínimo:

III - Indicação, pela Equipe de Planejamento da Contratação, dos termos contratuais, observado o disposto nos §§ 1º e 2º deste artigo, sem prejuízo do estabelecido na Lei nº 8.666, de 1993, relativos a: (i) fixação de procedimentos e Critérios de Aceitação dos serviços prestados ou bens fornecidos, abrangendo métricas, indicadores e valores mínimos aceitáveis; e (ii) definição de metodologia de avaliação da qualidade e da adequação da Solução de Tecnologia da Informação às especificações funcionais e tecnológicas.

Além disso, para a fase de gerenciamento do contrato a norma prevê no art. 25 § 3º:

III - monitoramento da execução, que consiste em: (i) avaliação da qualidade dos serviços realizados ou dos bens entregues e justificativas, de acordo com os Critérios de Aceitação definidos em contrato, a cargo dos Fiscais Técnico e Requisitante do Contrato;

### 2.2.3. Gestão de contratos de TI

A Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b) destaca que durante o gerenciamento do contrato se deve acompanhar e garantir a adequada prestação dos serviços e o fornecimento dos bens que compõem a solução de TI. Entre essas atividades de gerenciamento de contrato está a avaliação da qualidade dos serviços realizados ou dos bens entregues de acordo com os critérios de aceitação definidos.

O processo de gestão de contratos é essencial, pois segundo descrito por Cruz, Andrade e Figueiredo (2011), a fase gestão do contrato tem atividades com foco no monitoramento e controle dos serviços contratados e durante essa fase, deve-se assegurar que as entregas sejam realizadas dentro dos parâmetros

aceitáveis de desempenho e qualidade, deve-se tratar desvios, realizar o recebimento do objeto (ou partes dele) evidenciando a aceitabilidade da entrega em termos de desempenho e qualidade.

### 2.2.4. Processo de Contratações de Serviços de Tecnologia da Informação para Organizações Públicas (PCSTI)

O Processo de Contratação de Serviços de TI (PCSTI) definido por Cruz, Andrade e Figueiredo (2011) tem o objetivo de obter software e serviços de TI que satisfaçam às necessidades de negócio da organização contratante, alinhada à sua estratégia e à legislação brasileira, cumprindo com os princípios de eficácia, efetividade, economicidade, legalidade e legitimidade dos projetos de TI (CRUZ; ANDRADE; FIGUEIREDO, 2011).

O PCSTI pode ser aplicado na contratação de todos os serviços de TI, tais como desenvolvimento e manutenção de software, serviços de infraestrutura, serviços de *help desk*, consultorias especializadas e outros, e define fases, atividades e tarefas que podem ser adotadas para contratação de qualquer tipo de serviço de TI no setor público (CRUZ; ANDRADE; FIGUEIREDO, 2011).

Para a obtenção de serviços que satisfaçam as necessidades de negócio da organização contratante e atendam à legislação brasileira, é essencial que se tenha foco no controle da qualidade dos serviços de TI. O PCSTI define na fase Gestão do Contrato atividades com foco no monitoramento e controle dos serviços contratados, principalmente as atividades “Realizar o Monitoramento Técnico” e “Executar a atestação técnica”. Durante essas fases, deve-se assegurar que as entregas sejam realizadas dentro dos parâmetros aceitáveis de desempenho e qualidade, deve-se tratar desvios, realizar o recebimento do objeto (ou partes dele) evidenciando a aceitabilidade da entrega em termos de desempenho e qualidade (CRUZ; ANDRADE; FIGUEIREDO, 2011).

# 3. QUALIDADE DE SOFTWARE

Esta seção apresenta conceitos relacionados à Qualidade de Software e sobre as normas técnicas relacionadas. Diante das necessidades intrínsecas da contratação de serviços de TI, principalmente quando se trata do atendimento à legislação vigente, o tema Qualidade de Software é importante, pois fornece subsídios para auxiliar nas atividades de planejamento, acompanhamento e principalmente a aceitação de projetos de software contratados pela APF.

## 3.1. QUALIDADE DE PRODUTO DE SOFTWARE

Vários autores definiram o termo “qualidade” de formas diferentes. Para Phil Crosby (1979), qualidade é o grau de conformidade com os requisitos e para Humphrey (1989) qualidade é quando se atinge níveis excelentes que sejam adequados para o uso. Mais recentemente, o termo “qualidade” foi definido pela norma ISO/IEC 9000 (NBR/ISO, 2005) como a capacidade do produto de estar de acordo com os requisitos. Segundo Pressman (2011), em sentido geral, qualidade de software é a satisfação dos requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo o software desenvolvido profissionalmente. Mas a definição que mais se adequa à proposta do trabalho é a do Institute of Electrical and Eletronics Engineers (IEEE) (1998), que define que qualidade de software é o grau ao qual o software atende uma determinada combinação de atributos que devem ser claramente definidos. Para se avaliar os atributos definidos, um conjunto apropriado de métricas deve ser identificado. O objetivo das métricas de software é fazer avaliações ao longo do ciclo de vida do software para saber se os requisitos de qualidade de software estão sendo atendidos. O uso de métricas de software reduz a subjetividade na avaliação e controle de qualidade de software, fornecendo uma base quantitativa para a tomada de decisões sobre a qualidade do software (IEEE, 1998).

Medições da qualidade interna ou estrutural de software vêm sendo realizadas há um longo tempo na Engenharia de Software. Medidas de qualidade interna de software podem ser encontradas em trabalhos pioneiros desde a década de 70 como os de Halstead (1976), McCabe (1976), Gilb (1977), Boehm (1978) e Jim McCall (1977). Em particular, o trabalho de Boehm (1978) foi o principal precursor do modelo de qualidade definido pela ISSO/IEC 9126 (ISO/IEC, 2003),

substituída pela ISO/IEC 25000 (ISO/IEC, 2014) que é o principal padrão para avaliação da qualidade de produto de software utilizado atualmente.

Diversos modelos de qualidade foram propostos para definir a qualidade de software e os seus atributos. Para este trabalho são considerados a série de normas ISO/IEC 25000 (ISO/IEC, 2014) que trata da definição da qualidade de produto de software.

## 3.2. A SÉRIE ISO/IEC 25000

A série ISO/IEC 25000 – SQuaRE – Software Quality Requirements and Evaluation (Requisitos e Avaliação da Qualidade de Software) surgiu da necessidade de um conjunto harmônico de documentos para a utilização das normas de qualidade de produto. Assim, as séries 9126 (ISO/IEC, 2003) e 14598 (ISO/IEC, 2001) existentes levaram a uma lista de melhorias que foram implementadas na nova série. A série SQuaRE é estruturada de acordo com a seguinte divisão dentro do título *Software product Quality Requirements and Evaluation* (ISO/IEC, 2014):

- ISO/IEC 2500n – Divisão de Gestão da Qualidade;
- ISO/IEC 2501n – Divisão de Modelo de Qualidade;
- ISO/IEC 2502n – Divisão de Medição da Qualidade;
- ISO/IEC 2503n – Divisão de Requisitos de Qualidade; e
- ISO/IEC 2504n – Divisão de Avaliação da Qualidade.

A arquitetura SQuaRE é apresentada na Figura 1. O objetivo da criação do padrão SQuaRE foi a necessidade de uma estrutura logicamente organizada, enriquecida e unificada que abranja os três processos complementares: especificação de requisitos, medição e avaliação. A proposta da norma é auxiliar o desenvolvimento e a aquisição de produtos de software com a especificação e avaliação dos requisitos de qualidade. O padrão estabelece critérios para a especificação dos requisitos da qualidade do produto de software e para sua avaliação. Além disso, a série provê recomendações para medidas de atributos de qualidade do produto de software que podem ser utilizadas por desenvolvedores, adquirentes e avaliadores.


Figura 1 - Arquitetura SQuaRE. Fonte: ISO/IEC SQuaRE SERIES (2014).

A norma descreve um modelo de duas partes para a qualidade de produto de software. A primeira parte trata da qualidade interna e da qualidade externa e a segunda parte trata da qualidade em uso (ISO/IEC, 2014) e serão descritas a seguir:

**Qualidade em uso**: satisfazer as necessidades reais de usuários ao utilizar o produto de software, para atingir metas especificadas em contextos de uso especificados, ou seja, o efeito da utilização do produto, medido com relação às necessidades dos usuários.

**Qualidade externa**: influenciar o comportamento de um sistema, para satisfazer necessidades explícitas e implícitas, quando o sistema que o inclui for utilizado sob condições especificadas, ou seja, o efeito da execução das funções, medido com relação aos requisitos externos.

**Qualidade interna**: atributos estáticos do produto de software para satisfazer necessidades explícitas e implícitas, quando o software for utilizado em condições especificadas, ou seja, o efeito das propriedades de produtos intermediários, medidos com relação aos requisitos internos – projeto e código.

A primeira parte do modelo especifica oito características para qualidade interna e externa (ISO/IEC, 2014) (Fig. 2):

- **Adequação funcional**: o grau ao qual um produto ou sistema provê funções que satisfazem as especificações providas pelos usuários;

- **Eficiência de Performance**: performance relativa à quantidade de recursos usados em condições específicas;
- **Compatibilidade**: o grau no qual um produto, sistema ou componente pode trocar informações com outros produtos, sistemas ou componentes e/ou executa suas funções requeridas, enquanto compartilhando os mesmos recursos de hardware ou o mesmo ambiente;
- **Usabilidade**: o grau no qual um produto ou sistema pode ser usado por usuários específicos para atingir seus objetivos com efetividade, eficiência e satisfação em um contexto específico de uso;
- **Confiabilidade**: o grau no qual um sistema, produto ou componente executa suas funções em um contexto específico para um determinado período de tempo;
- **Segurança**: o grau no qual um produto ou sistema protege as informações e os dados de forma que pessoas ou outros produtos ou sistemas tenham o grau de acesso apropriado de acordo com níveis de autorização específicos;
- **Manutenabilidade**; o grau de efetividade e eficiência com o qual um produto ou sistema pode ser modificado pelos seus mantenedores;
- **Portabilidade**: o grau de efetividade e eficiência com o qual um sistema, produto ou componente pode ser transferido de um hardware, software ou ambiente de uso para outro;

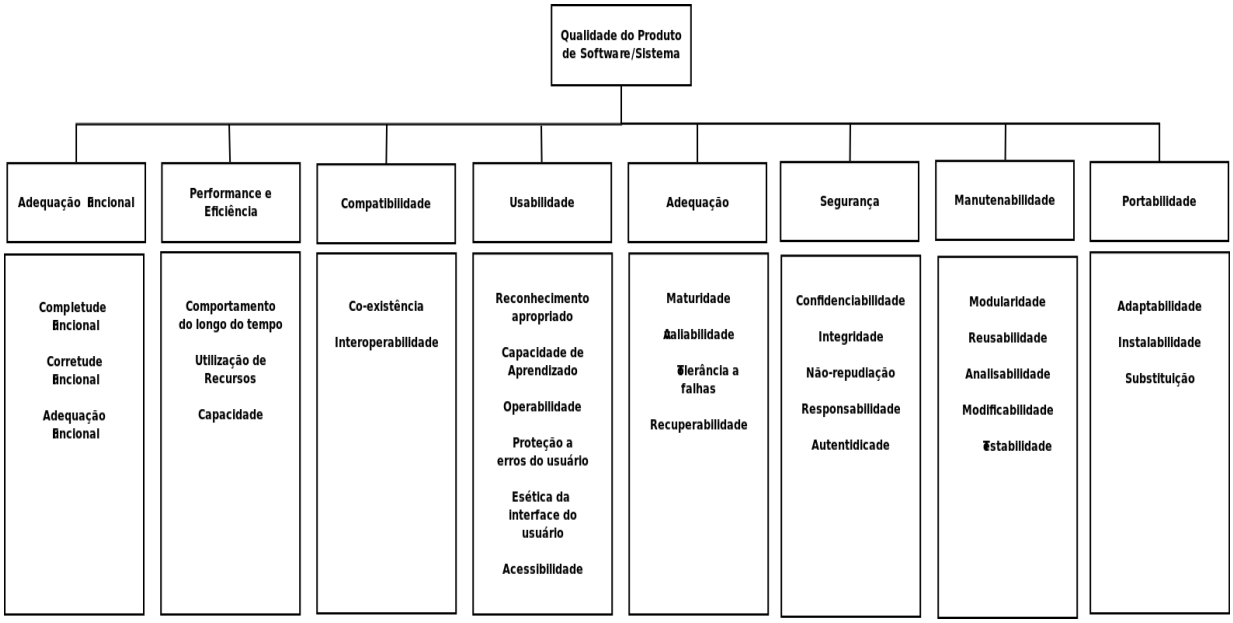



Figura 2 - Qualidade interna e externa. Fonte: ISO/IEC SQuaRE SERIES (2014).

A segunda parte do modelo especifica quatro características para a qualidade em uso (ISO/IEC, 2014) conforme apresentado na Figura 3:

- **Efetividade** - a capacidade do produto de software permitir que os usuários alcancem objetivos específicos com acuraria e completude em um contexto de uso específico;
- **Produtividade** - a capacidade do produto de software permitir que os usuários gastem quantidades específicas de recursos em relação à efetividade alcançada em um contexto de uso específico;
- **Segurança** - a capacidade do produto de software alcançar níveis aceitáveis de risco às pessoas, negócios, software, propriedades ou ambientes em um contexto de uso específico;
- **Satisfação** - a capacidade do produto de software satisfazer o usuário em um contexto de uso específico;

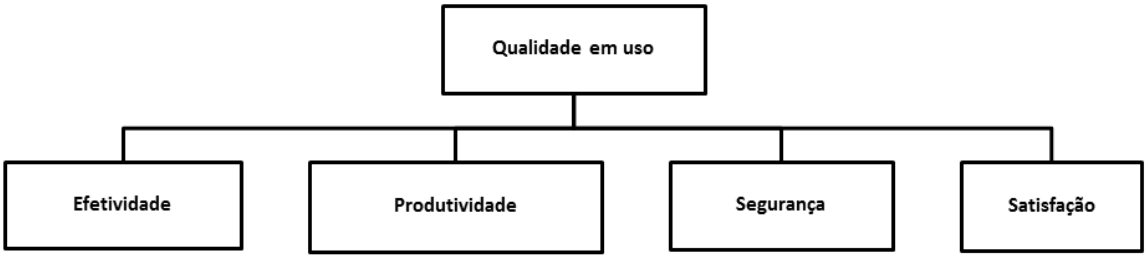



Figura 3 - Qualidade em uso. Fonte: ISO/IEC SQuaRE SERIES (2014).

As características de qualidade e as métricas associadas podem ser úteis não somente para a avaliação do produto de software, mas também para definir requisitos de qualidade, para avaliação da especificação de software, para definir particularidades e atributos do software, para avaliação antes da entrega e antes da aceitação do software desenvolvido e etc. A organização deve definir as características e sub-características de qualidade de acordo com as suas necessidades e identificar os diferentes níveis de características de qualidade necessárias para satisfazer o contexto de desenvolvimento de software (NAIK; TRIPATHY, 2011).

## 3.3. VERIFICAÇÃO DE SOFTWARE

Este trabalho tem como foco o processo de Verificação de Software. É durante o processo de verificação que é possível determinar se os produtos de software de uma atividade atendem completamente aos requisitos ou condições impostas a eles nas atividades anteriores (NBR/ISO, 2008). Ou seja, o processo verificação de software faz parte das atividades de monitoramento da execução do contrato de contratação, principalmente na avaliação da qualidade dos bens entregues e avaliação dos critérios de aceitação definidos em contrato de acordo com a Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b).

A norma internacional ISO/IEC 12207 (NBR/ISO, 2008) define verificação como sendo a confirmação, por exame e fornecimento de evidência objetiva, do atendimento dos requisitos especificados. Essa norma também apresenta um processo de verificação que pode ser usado como referência auxiliar na implementação deste processo e na interpretação de seus resultados esperados. É durante o processo de verificação que é possível confirmar se o produto de trabalho

obtido atende apropriadamente aos requisitos especificados (SOFTEX, 2013). Atividades de verificação devem ser executadas ao longo do desenvolvimento de um produto, começando geralmente com a verificação dos requisitos, seguindo com a verificação dos produtos intermediários (projeto (design), código, dentre outros) e concluindo com a verificação do produto final. Estas avaliações da qualidade, distribuídas ao longo de todo o ciclo de vida, têm como objetivos: (i) assegurar que os requisitos estabelecidos podem ser alcançados; (ii) identificar os requisitos que não podem ser alcançados; (iii) garantir que o software é desenvolvido de forma uniforme; (iv) identificar erros para tomar medidas corretivas o mais cedo possível; e (v) tornar o projeto mais gerenciável (PRESSMAN, 2011).

Apesar das recomendações da Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b) de que sejam estabelecidos critérios de aceitação que abrangem métricas, indicadores e valores mínimos, ou seja, critérios para a validação e verificação dos produtos obtidos com as contratações, a verificação da qualidade de produto de software nas contratações de TI da APF não é realizada adequadamente, como descrito na seção 1.2.

Segundo Sommerville ( 2007), existem duas abordagens complementares para a verificação e análise de software:

- **Inspeções de software ou revisões por pares** - analisam e verificam representações de sistemas como documentos, requisitos, diagramas de projeto e código-fonte do programa. Revisões de código, análises automatizadas e verificação formal são técnicas estáticas de análise, ou seja, não é necessário executar o software para que a análise seja realizada;
- **Testes de software** - envolvem a execução do software com dados de teste e análise das saídas e do comportamento operacional para verificar se o desempenho está de acordo com o que foi especificado. Os testes se classificam como uma técnica dinâmica de análise, ou seja, é necessário que o software seja executado para que a análise seja realizada.

Durante as inspeções de software o sistema é revisto para se encontrar erros, omissões e anomalias e geralmente as inspeções enfocam o código-fonte do

sistema. Durante as inspeção é usado o conhecimento do sistema, seu domínio de aplicação e a linguagem de programação ou modelo de projeto para se descobrir erros (SOMMERVILLE, 2007). Ainda segundo Sommerville (2007) existem três vantagens principais da inspeção sobre o teste: (i) não é necessário se preocupar com as interações entre erros, ou seja, um erro não irá mascarar a presença de outros erros; (ii) versões incompletas de um sistema podem ser inspecionadas sem custos adicionais; e (iii) as inspeções podem considerar também atributos de qualidade mais amplos como conformidade com padrões e facilidade de manutenção.

Em contratações de soluções de TI durante a fase Gerenciamento do Contrato, prevista pela Instrução Normativa MP/SLTI Nº04 (BRASIL, 2010b), deve-se receber provisoriamente o objeto resultante de cada ordem de serviço ou de fornecimento de bens a fim de se realizar avaliações de acordo com os critérios de aceitação especificados. As inspeções de software podem ser utilizadas como um dos instrumentos para a verificação da qualidade dos produtos resultantes dessas contratações.

## 3.4. ANÁLISE ESTÁTICA DE CÓDIGO-FONTE

Uma das formas de se inspecionar o código-fonte é pela análise estática de código-fonte. A análise estática consiste numa verificação automática do código-fonte, normalmente em tempo de compilação. Existem diversos analisadores estáticos que processam o código-fonte em cooperação com o compilador (YVES; WOUTER; FRANK, 2005). Existem também analisadores que requerem que o código-fonte seja compilado antes de o tratar (MCGRAW; CHESS; MIGUES, 2011). O objetivo da análise estática é encontrar vulnerabilidades nos sistemas analisados através de um conjunto de técnicas como o controle de fluxo e busca de padrões (SOTIROV, 2005).

Na seção seguinte são apresentadas uma coleção de ferramentas que apoiam a análise estática de código.

## 3.5. FERRAMENTAS PARA ANÁLISE ESTÁTICA DE CÓDIGO

Algumas das ferramentas mais conhecidas para análise estática de código são:

- **Checkstyle**: ferramenta de análise estática que verifica a utilização padrões de codificação pré-definidos[1];
- **Clang**: ferramenta de análise estática para as linguagens C, C++ e Objective-C[2;]
- **Coverity**: projeto que disponibiliza análises gratuitas para projetos de código aberto[3];
- **CppCheck**: ferramenta de análise estática de código para C e C++ que procura detectar bugs que os compiladores geralmente não encontram[4];
- **Findbugs**: ferramenta de análise estática de código para Java[5];
- **Flawfinder**: ferramenta de análise estática que detecta vulnerabilidades de diferentes tipos utilizando análise de padrões de problemas conhecidos[6];
- **HP Fortify**: ferramenta para análise estática de código que identifica e remove vulnerabilidades do código[7];
- **PMD**: ferramenta para análise estática de código que identifica erros comuns de codificação[8];
- **Red Lizard Goanna**: ferramenta de análise estática de código para C e C++[9].

[1] CheckStyle: www.checkstyle.sourceforge.net
[2] CLang: www. clang-analyzer.llvm.org
[3] Coverity: www.scan.coverity.com/about
[4] CppCheck: www.cppcheck.surceforge.net
[5] FindBugs: www.findbugs.sourceforge.net
[6] FlawFinder: www.dwheeler.com/flawfinder
[7] HPFortify: www8.hp.com/us/en/software-solutions/application-security/
[8] PMD: www.pmd.sourceforge.net
[9] Red Lizard Goanna: www.redlizards.com

### 3.5.1. SonarQube

As ferramentas de análise estática de código apresentadas geram diversos relatórios como resultado da análise. Porém, os relatórios gerados podem não ser os mais adequados para a visualização dos resultados, é necessário uma plataforma na qual seja possível a visualização de forma prática e inteligível. O SonarQube[10] provê os recursos necessários para a visualização adequada dos resultados.

O SonarQube é uma plataforma de qualidade de código gratuita e de código aberto que oferece uma visão da qualidade do código ao longo do projeto. O SonarQube é composto por três componentes (Fig. 4): (i) o banco de dados para armazenar as configurações da ferramenta e os resultados das análises; (ii) um web server no qual é possível visualizar o resultado das análises através de um browser e também para realizar as configurações necessárias; e (iii) um ou mais analisadores para realizar as análises dos projetos (CAMPBELL; PAPAPETROU, 2013). Dos analisadores citados na seção anterior, são suportados pelo SonarQube o Checkstyle, o FindBugs e o PMD.

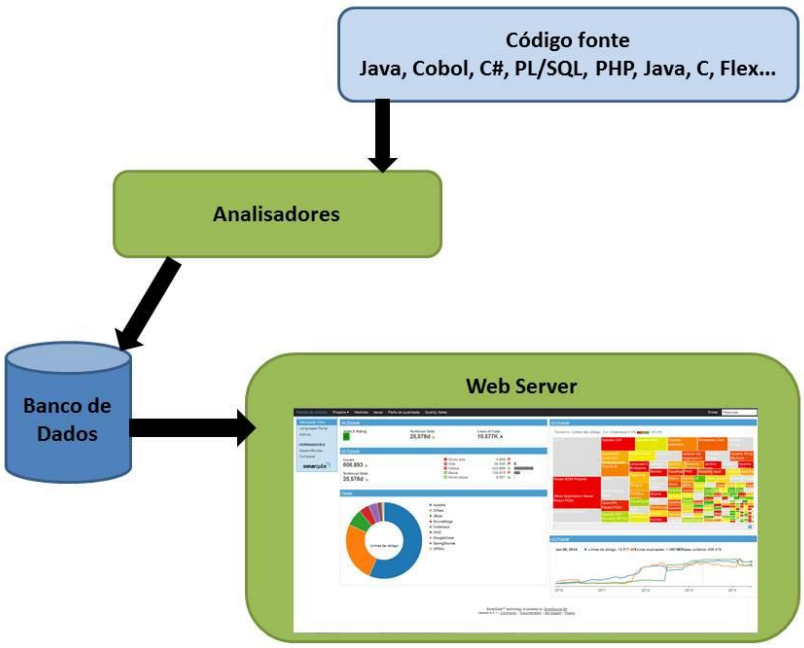



Figura 4 - Componentes do SonarQube (Fonte: Adaptado de SonarQube In Action (CAMPBELL; PAPAPETROU, 2013))

[10] SonarQube: http://www.sonarqube.org/

O SonarQube é baseado em um conceito chamado Sete Eixos da Qualidade (em inglês *Seven of Quality*) (CAMPBELL; PAPAPETROU, 2013). Os eixos da qualidade avaliados pelo SonarQube são:

- *Bugs* em potencial;
- Regras de codificação;
- Testes;
- Duplicações;
- Comentários;
- Arquitetura e design; e
- Complexidade.

As violações de regras reportadas pelo SonarQube são dividas em seis categorias (CAMPBELL; PAPAPETROU, 2013):

- **Bugs** - são erros lógicos que podem levar a *null pointer exceptions*, falhas ao fechar arquivos ou conexões com bancos de dados, mau comportamento em um ambiente *multithreaded*, métodos que checam igualdade, mas sempre retornam falso (ou verdadeiro), casts impossíveis etc;
- **Bugs em potencial** - são problemas condicionais que não irão ocorrer frequentemente. São potenciais *null pointer exceptions* que ocorrem em situações específicas, checagem de valores nulos que leva à desreferenciação, os itens que estão sendo checadas, operações matemáticas que usam precisões erradas etc;
- **Indicadores de possíveis erros de programação** - são comparações entre objetos utilizando-se == ou != ao invés do método .*equals(),* condicionais que que fazem atribuições ao invés de fazer comparações e sempre retornam verdadeiro, código que nunca é utilizado, blocos *catch* que ignoram exceções ao invés de usar logs etc;
- **Futuros erros de programação** - futuros erros de programação podem tornar o programa difícil de ser mantido. Podem ser convenções de nomeação, classes muito grandes que precisam ser refatoradas,

métodos que são muito longos e/ou muito complexos, condicionais com muitas cláusulas etc.;

- **Ineficiências** - podem impedir o programa de atingir a eficiência máxima e podem se tornar um problema quando o uso do banco de dados aumenta ou quando o uso da CPU é compartilhado. São usos ineficientes de *strings*, *imports* desnecessários, uso ineficiente de tipos especiais Java com o *BigDecimal* e código que nunca é usado;

Inconsistências de estilo: são regras referentes ao posicionamento de chaves, endentação etc.

Além disso, as violações de regras reportadas pelo SonarQube são divididas de acordo com a sua severidade em: muito altas, altas, médias, baixas e muito baixas (Fig. 5).

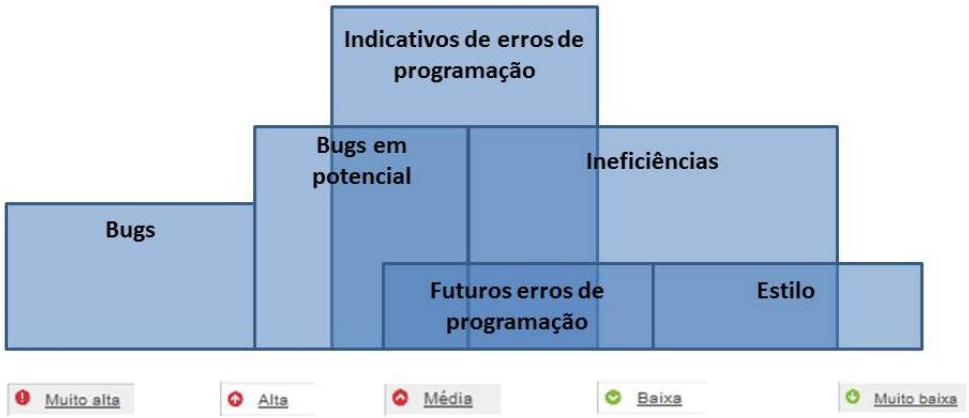



Figura 5 - Violações de regra classificadas por severidade (Fonte: Adaptado de SonarQube In Action (CAMPBELL; PAPAPETROU, 2013))

## 3.6. LIMITAÇÕES DAS FERRAMENTAS DE ANÁLISE ESTÁTICA

Ferramentas de análise estática frequentemente produzem falsos resultados que indicam uma possível vulnerabilidade quando na verdade não existem. São os chamados falsos positivos. Essas ferramentas podem também resultar em falsos negativos, resultados nos quais vulnerabilidades existentes não são reportadas (OWASP, 2014). Vários estudos vêm sendo realizados para melhorar a qualidade dos resultados das ferramentas de análise estática de código como o Static Analysis

Tool Exposition (SATE)[11] realizado pela equipe de pesquisadores do *National Institute of Standards and Technology* para explorar caraterísticas das ferramentas como a relevância, a corretude e a priorização das violações de regra apontadas.

Os resultados das análises podem ser verificados pelo SonarQube. O SonarQube disponibiliza a violação de regra e aponta a linha de código na qual ela foi identificada. As violações de regras podem ser analisadas e marcadas como “confirmadas” ou como “resolvidas” quando são falsos positivos.

## 3.7. CONSIDERAÇÕES FINAIS DO CAPÍTULO

A série ISO/IEC 25000 (ISO/IEC, 2014) é internacionalmente reconhecida e define uma linguagem comum relacionada à qualidade de produto de software. A identificação e definição das características de qualidade de software definidos pela série ISO/IEC 25000 (ISO/IEC, 2014) provê um modelo útil de referência e terminologias padronizadas, o que facilita a comunicação a respeito de qualidade de software.

Porém, a ISO/IEC 25000 (ISO/IEC, 2014) não é livre de falhas e vem sido criticada por ter sido definida de forma operacional e não prática (KANELLOPOULOS et al., 2010). A definição de métricas é baseada na avaliação posterior das características definidas e no geral as avaliações são baseadas no esforço e no tempo gastos nas atividades relacionadas ao produto de software. As medidas internas e externas não são baseadas diretamente na observação do produto de software, mas sim na observação de interações entre o produto e o seu ambiente: os seus mantenedores, os seus testadores, os seus administradores; ou na comparação do produto com suas especificações, o que pode ser incompleto, desatualizado ou incorreto (AL-KILIDAR; COX; KITCHENHAM, 2005).

As métricas propostas não são coletadas a partir código-fonte e da documentação, mas são medidas tendo como base a performance da atividade de manutenção realizada pela equipe técnica. Por exemplo, a medida “impacto de mudança” para medir a capacidade de mudança do sistema é computada a partir do número de modificações feitas e o número de problemas causados por essas modificações (HEITLAGER; KUIPERS; VISSER, 2007).

[11] SATE: http://samate.nist.gov/SATE5Workshop.html

Vários trabalhos vêm sendo desenvolvidos de modo a relacionar medidas de código-fonte às sub-características definidas pela ISO/IEC-9126 (ISO/IEC, 2003). Heitlager, Kuipers e Visser (2007) descrevem o mapeamento das características à nível de sistema para propriedades à nível de código-fonte, por exemplo, a característica capacidade de mudança de um sistema é ligada à propriedades como complexidade do código-fonte. Em seguida, para cada propriedade é mapeado uma ou mais medidas de código-fonte, como por exemplo complexidade que é medida em termos de complexidade ciclomática (HEITLAGER; KUIPERS; VISSER, 2007).

O trabalho de Antonellis et. al (2007) apresenta uma metodologia para avaliação do produto de software de acordo com a ISO/IEC-9126 (ISO/IEC, 2003), os autores combinam medições de dados extraídas de elementos de código-fonte e a expertise de avaliadores de sistemas. Os especialistas definem pesos para as métricas de código-fonte, relacionam tais métricas às sub-características da norma, em seguida atribuem pesos também às sub-caracterísitcas da ISO/IEC-9126 (ISO/IEC, 2003). A atribuição de pesos é feita utilizando o *Analytical Hierarchy Process* (AHP) e reflete a visão do especialista da importância de cada um dos elementos envolvidos. Em seguida são utilizadas técnicas de mineração de dados para agrupar elementos de código-fonte que possuem medidas semelhantes (ANTONELLIS et al., 2007).

O trabalho de Kanellopoulos et al. (2008) utiliza como base o mapeamento das características em nível de sistema para propriedades à nível de código-fonte definido por Heitlager, Kuipers e Visser (2007), atribui pesos, utilizando o AHP, a cada uma das métricas, sub-características e características, utiliza técnicas de mineração de dados para agrupar os elementos de código-fonte que possuem medidas semelhantes e disponibiliza os dados extraídos em forma de histogramas para a visualização dos diversos perfis de qualidade definidos.

As ferramentas de análise estática utilizadas pelo SonarQube apontam uma série de violações de regra que não estão associadas a um padrão ou norma específica. Ou seja, é possível obter indicadores sobre o número de violações de regras para cada projeto, sobre a complexidade ciclomática de determinado método ou sobre a má utilização de blocos *try-catch*. Porém, não é possível determinar qual o impacto dessas violações de regras nas dimensões de qualidade definidas pela ISO/IEC 25000 (ISO/IEC, 2014).

# 4. ESTUDO DE CASO

Este capítulo apresenta o estudo de caso realizado utilizando códigos-fonte de softwares disponibilizados por um órgão integrante da Administração Pública Federal também de *softwares* coletados no *Portal do Software Público Brasileiro*. Utilizando o SonarQube foram analisados vinte projetos *softwares* e as violações de regras indicadas foram comparadas para se obter uma amostra das regras mais violadas. O resultado da análise forma um conjunto de regras que devem ser analisadas em projetos resultantes de contratações de soluções de TI, pois representam, no contexto das contratações órgão da APF em questão, erros comuns à maioria dos projetos analisados.

## 4.1. CONFIGURAÇÃO DO SONARQUBE

Para o estudo de caso foram ativadas todas as regras disponíveis no SonarQube, além disso foram adicionados plug-ins de ferramentas de análise estática como o *plug-in* da ferramenta PMD, o plug-in da ferramenta *Checkstyle* e o *plug-in* para análise de código-fonte escrito em linguagem PHP.

A severidade associada às regras foi mantida e as análises foram realizadas com 545 regras ativas para as análises de projetos de Java nos quais não foi possível obter o *byte code* (Fig. 6), 839 regras ativas para projetos Java nos quais foi possível obter o *byte code* (Fig. 7) e 61 regras ativas para projetos PHP (Fig. 8).

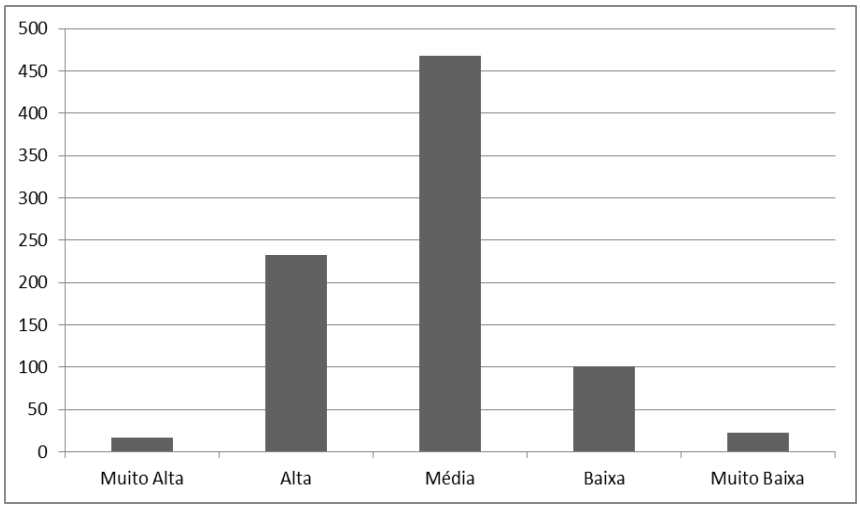



Figura 6 - Severidade das regras padrão do SonarQube para Java

Para projetos Java foram ativadas as regras do *FindBugs* e do *Squid* e também todas as regras dos plug-ins PMD e *Checkstyle*. Para o *FindBugs* é necessário que o *byte code* esteja disponível para que a análise seja realizada, então as regras do *FindBugs* foram ativadas apenas para os projetos SAF e SIF. Para projetos PHP foram ativadas todas as regras do plug-in PHP.

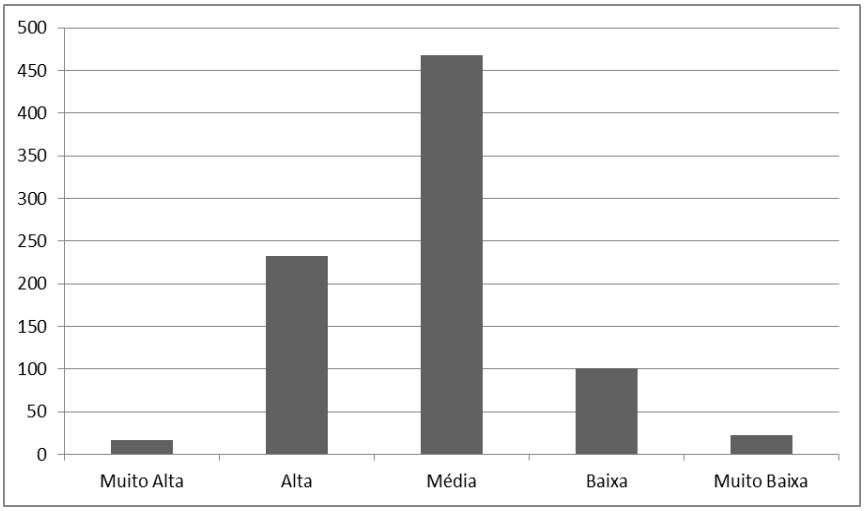



Figura 7 - Severidade das regras padrão do SonarQube para Java com *FindBugs*

Essas customizações das regras disponíveis compõe o Perfil de Qualidade no SonarQube. O Perfil de Qualidade é o “coração” do SonarQube, as avaliações são dependentes dessa seleção de regras e de como elas regras são classificadas (CAMPBELL; PAPAPETROU, 2013).

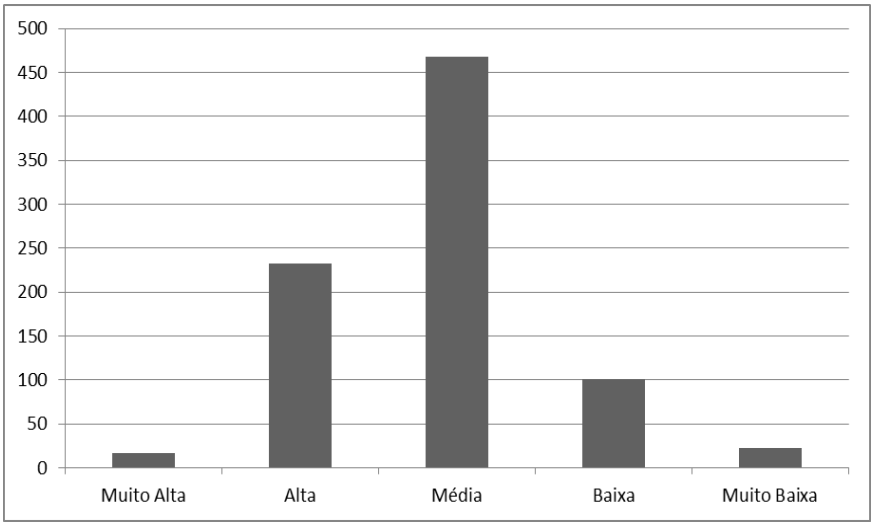



Figura 8 - Severidade das regras padrão do SonarQube para PHP

O Perfil de Qualidade define o tipo de linguagem a ser analisada, a severidade das violações de regra identificadas, padrões de expressões regulares que devem estar presentes ou não no código, os projetos aos quais se deve aplicar o Perfil de Qualidade e etiquetas que podem ser associadas a cada uma das regras.

## 4.2. O PORTAL DO SOFTWARE PÚBLICO BRASILEIRO

O Portal do Software Público Brasileiro[12] foi criado em abril de 2007 e tem como objetivo o compartilhamento de software e manutenção de comunidades para desenvolvimento e manutenção dos softwares disponibilizados. O portal é composto de *wikis*, sistema de controle de versões, bug tracking, mercado virtual para consultar prestadores de serviços, o 5CQualiBr que é um grupo que trabalha para evoluir a qualidade do software público brasileiro, entre outros. As soluções disponibilizadas são de livre uso e são utilizados principalmente por órgãos ligados ao governo.

### 4.2.1. Análise dos projetos do Portal do Software Público Brasileiro

Foram analisadas nove soluções disponibilizadas no Portal do Software Público, sendo que o critério utilizado para a escolha das soluções a serem analisadas foi a quantidade de participantes na comunidade de cada solução (Tab. 4), sendo selecionados aqueles com maior quantidade.

Tabela 4 - Soluções analisadas. Para o número de linhas de código não são considerados comentários ou espaços em branco.

| Solução analisada | Data de download do código | Linguagem | Linhas de código | Violações de regra identificadas | Severidade | |
|---|---|---|---|---|---|---|
| ASES | 05/05/2014 | Java | 39.187 | 167.968 | Muito alta | 4 |
| | | | | | Alta | 609 |
| | | | | | Média | 102.210 |
| | | | | | Baixa | 63,022 |
| | | | | | Muito baixa | 2,123 |
| Cocar | 05/05/2014 | PHP | 5.078 | 2.359 | Muito alta | 0 |
| | | | | | Alta | 0 |
| | | | | | Média | 2.163 |
| | | | | | Baixa | 128 |

[12] Portal do Software Público Brasileiro: http://www.softwarepublico.gov.br/

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | Muito baixa | 68 |
| Demoiselle Framework Core | 05/05/214 | Java | 10.866 | 87.520 | Muito alta | 8 |
| | | | | | Alta | 29 |
| | | | | | Média | 70.716 |
| | | | | | Baixa | 16.711 |
| | | | | | Muito baixa | 56 |
| I-Educar | 05/05/2014 | PHP | 156.317 | 67.273 | Muito alta | 0 |
| | | | | | Alta | 0 |
| | | | | | Média | 62.707 |
| | | | | | Baixa | 4.129 |
| | | | | | Muito baixa | 437 |
| MDarte | 05/05/2014 | Java | 11.714 | 18.395 | Muito alta | 55 |
| | | | | | Alta | 74 |
| | | | | | Média | 11.235 |
| | | | | | Baixa | 6.672 |
| | | | | | Muito baixa | 359 |
| Oasis | 05/05/2014 | PHP | 90.988 | 46.431 | Muito alta | 0 |
| | | | | | Alta | 1 |
| | | | | | Média | 38.834 |
| | | | | | Baixa | 7.447 |
| | | | | | Muito baixa | 149 |
| SGF | 05/05/2014 | Java | 15.231 | 45.690 | Muito alta | 0 |
| | | | | | Alta | 43 |
| | | | | | Média | 20.845 |
| | | | | | Baixa | 24.560 |
| | | | | | Muito baixa | 242 |
| Web Integrator | 05/05/2014 | Java | 41.483 | 140.519 | Muito alta | 37 |
| | | | | | Alta | 262 |
| | | | | | Média | 109.733 |
| | | | | | Baixa | 29.216 |
| | | | | | Muito baixa | 1.271 |

O SonarQube possibilita visualizar no painel referente a cada projeto as cinco violações de regra mais frequentes (Tab. 5). Essa visão proporciona uma amostra rápida dos problemas encontrados no projeto e pode ser usada como uma amostra da gravidade das violações de regra encontradas.

Tabela 5 - As cinco violações de regra mais frequentes de cada projeto

| **ASES** | |
|---|---|
| Código duplicado | 31.017 |
| Endentação | 30.249 |
| Código-fonte deve ser corretamente endentado | 10.457 |
| Largura da linha | 7.485 |
| Abre-chaves devem ser localizadas no início de uma linha | 6.975 |
| **Cocar** | |
| Linhas não devem terminar com espaço | 765 |
| Abre-chaves devem ser localizadas ao final da linha | 331 |
| If/else/for/whilo/do devem sempre usar chaves | 177 |
| Uma declaração por linha | 166 |
| Linhas não devem ser tão longas | 160 |
| **Demoiselle Framework Core** | |
| Código duplicado | 55.978 |
| Endentação | 7.283 |
| Código-fonte deve ser corretamente endentado | 2.968 |
| Largura da linha | 2.494 |
| Abre-chaves devem ser localizadas no início de uma linha | 2.072 |
| **I-Educar** | |
| Abre-chaves devem ser localizadas ao final da linha | 21.284 |
| Nomes de variáveis locais e parâmetros devem seguir a convenção de nomeação | 7.764 |
| Linhas não devem terminar com espaços | 6.375 |
| Linhas não devem ser tão longas | 6.249 |
| Nomes de campos devem estar de acordo com os padrões de nomeação | 5.812 |
| **MDarte** | |
| Código-fonte deve ser corretamente endentado | 3,022 |
| Abre-chaves devem ser localizadas ao final da linha | 1.945 |
| Abre-chaves | 1.816 |
| Lei de Demeter | 1.389 |
| Largura da linha | 1.294 |
| **Oasis** | |
| Linhas não devem ser tão longas | 10.565 |
| Linhas não devem terminar com espaços em branco | 10.461 |
| Abre-chaves devem ser localizadas no início de uma linha | 6.424 |
| Strings literais não devem ser duplicadas | 3.765 |
| Abre-chaves devem ser localizadas ao final da linha | 3.730 |

| SGF | |
|---|---|
| Endentação | 12.268 |
| Código-fonte deve ser corretamente endentado | 5.205 |
| Abre-chaves devem ser localizadas no início de uma linha | 3,058 |
| Comentário requerido | 2,640 |
| Lei de Demeter | 1,857 |
| **Web Integrator** | |
| Código duplicado | 48,932 |
| Código-fonte deve ser corretamente endentado | 16,332 |
| Abre-chaves devem ser localizadas no início de uma linha | 10,021 |
| Lei de Demeter | 6,236 |
| This é requerido | 5.982 |

## 4.3. O ÓRGÃO DA APF PARTICIPANTE DO ESTUDO DE CASO

A Universidade de Brasília tem um termo de cooperação técnica há cerca de um ano e um dos objetivos é cooperar com a melhoria do processo de contratação de software desse órgão. Essa parceria possibilitou que a estrutura interna do órgão em questão fosse conhecida e, em conjunto com a equipe da Divisão de Sistemas (DISIS), foi possível selecionar os projetos de software para a análise e estabelecer os critérios para a seleção das regras das ferramentas de análise estática.

### 4.3.1. Análise dos projetos do órgão da APF

Foram analisados onze projetos disponibilizados pelo órgão da APF, o critério utilizado para a escolha dos projetos foi a importância de cada um deles no contexto do órgão em questão (Tab. 6).

Tabela 6 - Soluções analisadas. Para o número de linhas de código não são considerados comentários ou espaços em branco

| Solução analisada | Data de análise do código | Linguagem | Linhas de código | Violações de regra identificadas | Severidade | |
|---|---|---|---|---|---|---|
| CPROD | 27/05/2014 | Java | 33.643 | 115.574 | Muito alta | 0 |
| | | | | | Alta | 250 |
| | | | | | Média | 59.420 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | Baixa | 55.463 |
| | | | | | Muito baixa | 441 |
| DSCOM | 27/05/2014 | PHP | 5.621 | 2.827 | Muito alta | 0 |
| | | | | | Alta | 0 |
| | | | | | Média | 2.556 |
| | | | | | Baixa | 209 |
| | | | | | Muito baixa | 62 |
| FAPTI | 27/05/214 | PHP | 53.944 | 36.265 | Muito alta | 0 |
| | | | | | Alta | 47 |
| | | | | | Média | 33,294 |
| | | | | | Baixa | 1,927 |
| | | | | | Muito baixa | 997 |
| REPNBL | 27/05/2014 | Java | 19.399 | 63.593 | Muito alta | 0 |
| | | | | | Alta | 33 |
| | | | | | Média | 30.983 |
| | | | | | Baixa | 32.186 |
| | | | | | Muito baixa | 391 |
| SAF | 27/05/2014 | Java | 6.724 | 22.856 | Muito alta | 0 |
| | | | | | Alta | 38 |
| | | | | | Média | 11,041 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | Baixa | 11,703 |
| | | | | | Muito baixa | 74 |
| SCOP | 27/05/2014 | Java | 7.940 | 25.150 | Muito alta | 0 |
| | | | | | Alta | 9 |
| | | | | | Média | 11.964 |
| | | | | | Baixa | 13.073 |
| | | | | | Muito baixa | 104 |
| SIF | 27/05/2014 | Java | 2.373 | 7.257 | Muito alta | 0 |
| | | | | | Alta | 0 |
| | | | | | Média | 3.694 |
| | | | | | Baixa | 3.545 |
| | | | | | Muito baixa | 18 |
| SISTP | 27/05/2014 | Java | 8.173 | 21.793 | Muito alta | 0 |
| | | | | | Alta | 20 |
| | | | | | Média | 9.395 |
| | | | | | Baixa | 12.302 |
| | | | | | Muito baixa | 76 |
| SOE | 27/05/2014 | Java | 2.388 | 6.409 | Muito alta | 0 |
| | | | | | Alta | 9 |
| | | | | | Média | 3,122 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | Baixa | 3,228 |
| | | | | | Muito baixa | 50 |
| SRAPD | 27/05/2014 | Java | 7,215 | 20,946 | Muito alta | 0 |
| | | | | | Alta | 3 |
| | | | | | Média | 9.839 |
| | | | | | Baixa | 10.965 |
| | | | | | Muito baixa | 139 |
| SRH | 27/05/2014 | Java | 27.368 | 78.813 | Muito alta | 0 |
| | | | | | Alta | 78 |
| | | | | | Média | 36,803 |
| | | | | | Baixa | 41.416 |
| | | | | | Muito baixa | 516 |

Na Tabela (7) são apresentadas as cinco violações de regra mais frequentes de cada projeto. Pode-se perceber que muitas das violações de regra coincidem com as violações de regra mais frequentes dos projetos do Portal do Software Público Brasileiro.

Tabela 7 - As cinco violações de regra mais frequentes de cada projeto

| **CPROD** | |
|---|---|
| Endentação | 29.845 |
| Código-fonte deve ser corretamente endentado | 12.691 |
| Abre-chaves devem ser localizadas no início de uma linha | 7.501 |
| Lei de Demeter | 7,073 |
| Comentário requerido | 6.642 |
| **DSCOM** | |
| Linhas não devem terminar com espaço | 853 |
| Abre-chaves devem ser localizadas no início de uma linha | 488 |
| If/else/for/whilo/do devem sempre usar chaves | 296 |
| Abre-chaves devem ser localizadas ao final de uma linha | 274 |
| Linhas não devem ser tão longas | 235 |
| **FAPTI** | |
| Linhas não devem terminar com espaço | 10.290 |
| Abre-chaves devem ser localizadas no início de uma linha | 7.904 |
| If/else/for/whilo/do devem sempre usar chaves | 3.074 |
| Instruções devem estar em linhas separadas | 2.425 |
| Abre-chaves devem ser localizadas ao final de uma linha | 1.865 |
| **REPNBL** | |
| Endentação | 16.127 |
| Código-fonte deve ser corretamente endentado | 7.000 |
| Largura da linha | 4.220 |
| Lei de Demeter | 4.192 |
| Abre-chaves devem ser localizadas no início de uma linha | 4.170 |
| **SAF** | |
| Endentação | 5.609 |
| Código-fonte deve ser corretamente endentado | 2.396 |
| Abre-chaves devem ser localizadas no início de uma linha | 1.372 |
| Comentário requerido | 1.314 |
| Largura da linha | 970 |
| **SCOP** | |
| Endentação | 7.021 |
| Código-fonte deve ser corretamente endentado | 3.316 |
| Abre-chaves devem ser localizadas no início de uma linha | 1.918 |
| Comentário requerido | 1.691 |
| Design para extensão | 1.042 |
| **SIF** | |
| Endentação | 2.034 |
| Código-fonte deve ser corretamente endentado | 850 |
| Abre-chaves devem ser localizadas no início de uma linha | 512 |
| Comentário requerido | 413 |
| Largura da linha | 364 |
| **SISTP** | |
| Endentação | 5.524 |
| Código-fonte deve ser corretamente endentado | 2.408 |
| Abre-chaves devem ser localizadas no início de uma linha | 1.486 |
| Comentário requerido | 1.394 |
| Espaços em branco em volta | 1.088 |
| **SOE** | |
| Endentação | 1.383 |
| Código-fonte deve ser corretamente endentado | 812 |
| Abre-chaves devem ser localizadas no início de uma linha | 475 |
| Comentário requerido | 429 |
| Javadoc método | 309 |
| **SRAPD** | |
| Endentação | 5.960 |

| | |
|---|---|
| Código-fonte deve ser corretamente endentado | 2.808 |
| Abre-chaves devem ser localizadas no início de uma linha | 1.599 |
| Comentário requerido | 1.406 |
| Design para extensão | 871 |
| **SRH** | |
| Endentação | 20.348 |
| Código-fonte deve ser corretamente endentado | 8.636 |
| Abre-chaves devem ser localizadas no início de uma linha | 4.994 |
| Comentário requerido | 4.851 |
| Javadoc método | 3.550 |

## 4.4. CONSIDERAÇÕES FINAIS DO CAPÍTULO

Observando-se o resultado da análise utilizando todas as regras disponibilizadas pelo SonarQube percebe-se que muitas regras são repetidas e até mesmo incoerentes entre si como as regras que falam sobre endentação do código e do posicionamento das chaves. A presença de regras repetidas faz com que o número total de violações de regra aumente e caso isso não seja corrigido pode causar redundâncias no processo de análise e correções das violações de regra.

Esse resultado era esperado dado que algumas ferramentas como o *Squid* e o PMD apresentam regras semelhantes. Para que esse efeito seja eliminado é necessário que as regras a serem analisadas sejam selecionadas e sua severidade seja atribuída de acordo com o contexto dos projetos que se deseja analisar. O próximo capítulo apresenta o processo de análise, seleção e classificação dessas regras que irão compor o Perfil de Qualidade para Órgãos da APF.

# 5. DEFINIÇÃO DO PERFIL DE QUALIDADE PARA AS CONTRATAÇÕES DE SOLUÇÕES DE TI

Para a definição do Perfil de Qualidade para Órgãos da APF, as violações de regras indicadas nas análises dos softwares do Portal do Software Público e do órgão participante do estudo de caso foram comparadas. Inicialmente foram selecionadas as violações de regras indicadas em oito ou mais projetos Java e em três ou mais projetos PHP. Em seguida a relevância de cada uma das regras foi analisada, as regras também foram classificadas de acordo com as categorias de violações descritas na Seção 3.5.1 e também foram atribuídas severidades de acordo com a descrição da Seção 3.5.1.

## 5.1. ANÁLISE DAS VIOLAÇÕES DE REGRA IDENTIFICADAS

Durante o estudo de caso foram identificadas 449 diferentes violações de regras nas análises de projetos Java e 61 diferentes violações de regras nas análises de projetos PHP. Essas violações de regra identificadas foram analisadas uma a uma em cinco etapas: 1- Identificação de regras repetidas; 2 - Identificação de violações de regra comuns; 3 - Seleção de regras adicionais; 4 - Identificação do impacto, definição da severidade e classificação; 5 - Mapeamento para as sub-características de qualidade definidas pela ISO/IEC 25000 (ISO/IEC, 2014). A seguir a descrição de cada uma das etapas:

- **Identificação das regras repetidas**: a análise iniciou-se com a eliminação das regras que, apesar de possuírem nomes diferentes, apontam a mesma violação de regra. As principais ocorrências de regras repetidas estão relacionadas às ferramentas PMD e *Squid*;
- **Identificação de violações de regra comuns**: após a eliminação das regras repetidas foram selecionadas as regras presentes em oito ou mais projetos Java e em três ou mais projetos PHP, ao final foram obtidas 130 regras para projetos Java, sem incluir as regras do *FindBugs* e 47 regras para projetos PHP;
- **Seleção de regras adicionais**: além das regras selecionadas a partir da análise dos projetos, também foram selecionadas, em conjunto com a equipe da DISIS do órgão participante do estudo de caso, outras regras relevantes que não foram identificadas na análise inicial. Entre

elas estão também as regras do *FindBugs*. Essas regras adicionais foram selecionadas levando-se em conta o contexto das contratações de soluções de TI do órgão participante do estudo de caso e estão relacionadas a más práticas de codificação, corretude, vulnerabilidades maliciosas de código, performance e segurança. Ao final foram obtidas 340 regras para projetos Java e 54 regras para projetos PHP;

- **Identificação do impacto, definição da severidade e classificação**: as regras obtidas foram detalhadamente analisadas para identificar o impacto causado pela violação de cada uma delas. Segundo as descrições de categorias e severidades da Seção 3.5.1 também foi determinada a categoria à qual cada regra pertence e também a severidade de cada uma delas;

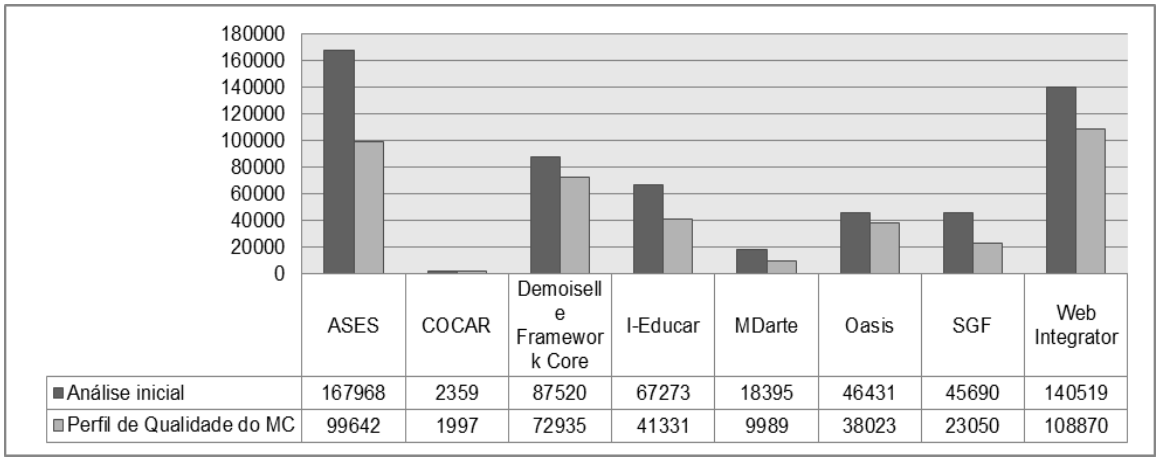
Mapeamento para as sub-características de qualidade: seguindo a abordagem proposta nos trabalhos de Heitlager et. Al (2007), Antonellis et. al (2007), Heitlager et. al (2007) e Kanellopoulos et al. (2008) as regras selecionadas foram mapeadas para as sub-características e características de qualidade definidas pela ISO/IEC 25000 (ISO/IEC, 2014). O mapeamento também foi feito em conjunto com a equipe da Divisão de Sistemas (DISIS) do órgão participante do estudo de caso e reflete o que, segundo o entendimento da equipe, afeta as dimensões de qualidade do produto definidas pela ISO/IEC 25000 (ISO/IEC, 2014). Utilizando o SonarQube foram adicionadas etiquetas a cada uma das regras referenciando as sub-características da ISO/IEC 25000 (ISO/IEC, 2014) que foram associadas à cada uma das regras.

Ao final da análise obteve-se o Perfil de Qualidade para Órgãos da APF que pode ser aplicado a projetos Java e a projetos PHP. A descrição completa das regras selecionadas encontra-se nos Anexo I e II. A maioria das regras está relacionada à característica Manutenabilidade da ISO/IEC 25000 (ISO/IEC, 2014), o que confirma a necessidade de verificação de código relacionada à Manutenabilidade dos softwares produtos das contratações de TI.

## 5.2. UTILIZAÇÃO DO PERFIL DE QUALIDADE DEFINIDO

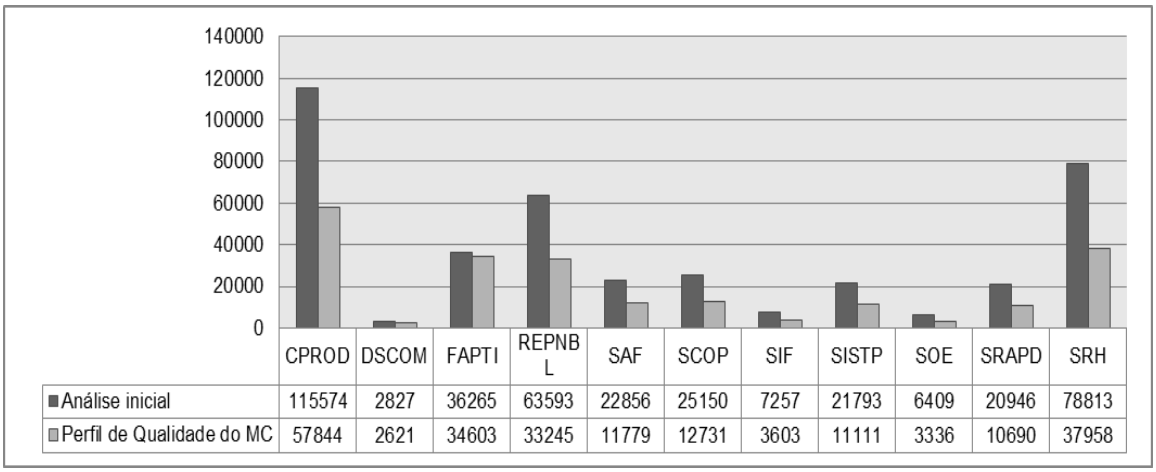
Após a definição do Perfil de Qualidade para Órgãos da APF os projetos do estudo de caso foram analisados novamente. Como já era esperado, observou-se a

diminuição do número de violações de regras identificadas em cada projeto devido à eliminação de regras repetidas ou que não se adequavam ao contexto dos projetos do órgão da APF participante do estudo de caso (Figs. 9 e 10).




| | ASES | COCAR | Demoiselle Framework Core | I-Educar | MDarte | Oasis | SGF | Web Integrator |
|---|---|---|---|---|---|---|---|---|
| Análise inicial | 167968 | 2359 | 87520 | 67273 | 18395 | 46431 | 45690 | 140519 |
| Perfil de Qualidade do MC | 99642 | 1997 | 72935 | 41331 | 9989 | 38023 | 23050 | 108870 |

Figura 9 - Violações de regra identificadas nos Projetos do Portal do Software Público Brasileiro




| | CPROD | DSCOM | FAPTI | REPNBL | SAF | SCOP | SIF | SISTP | SOE | SRAPD | SRH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Análise inicial | 115574 | 2827 | 36265 | 63593 | 22856 | 25150 | 7257 | 21793 | 6409 | 20946 | 78813 |
| Perfil de Qualidade do MC | 57844 | 2621 | 34603 | 33245 | 11779 | 12731 | 3603 | 11111 | 3336 | 10690 | 37958 |

Figura 10 - Violações de regra identificadas nos projetos do Órgão da APF participante do estudo de caso

Observando-se as cinco violações de regra mais frequentes pode-se verificar que, quando o Perfil de Qualidade para Órgãos da APF é utilizado, não são indicadas regras repetidas ou incoerentes e que a severidade das violações mais frequentes é classificada como “Muito Baixa”, “Baixa” ou “Média” (Tabs. 8 e 9).

Tabela 8 - As cinco violações de regras mais frequentes dos projetos do Portal do Software Público

| Violação de regra | Número de ocorrências | Severidade |
|---|---|---|
| **ASES** | | |
| Código duplicado | 31.017 | Média |
| Código-fonte deve ser corretamente endentado | 10.457 | Baixa |
| Abre-chaves devem ser localizadas no início de uma linha | 6.975 | Baixa |

| | | |
|---|---|---|
| This é requerido | 6.200 | Média |
| Lei de Demeter | 5.679 | Média |
| **Cocar** | | |
| Linhas não devem terminar com espaço | 765 | Baixa |
| If/else/for/whilo/do devem sempre usar chaves | 177 | Baixa |
| Uma declaração por linha | 166 | Baixa |
| Linhas de código não devem ser tão longas | 160 | Média |
| Nomes de variáveis locais e parâmetros devem seguir a convenção de nomeação | 143 | Baixa |
| **Demoiselle Framework** | | |
| Código duplicado | 55.978 | Média |
| Código-fonte deve ser corretamente endentado | 2.968 | Baixa |
| Abre-chaves devem ser localizadas no início de uma linha | 2.072 | Baixa |
| Linhas de código não devem ser tão longas | 1.916 | Baixa |
| Comentário requerido | 1.677 | Baixa |
| **I-Educar** | | |
| Nomes de variáveis locais e parâmetros devem seguir a convenção de nomeação | 7.764 | Muito baixa |
| Linhas não devem terminar com espaços em branco | 6.375 | Baixa |
| Linhas de código não devem ser tão longas | 6.249 | Média |
| Strings literais não devem ser duplicadas | 1.953 | Média |
| Nomes de funções devem seguir a convenção de nomeação | 1.522 | Baixa |
| **MDarte** | | |
| Código-fonte deve ser corretamente endentado | 3,022 | Baixa |
| Lei de Demeter | 1.391 | Média |
| Linhas de código não devem ser tão longas | 1.292 | Baixa |
| Tamanho do comentário | 572 | Baixa |
| Design para extensão | 427 | Baixa |
| **Oasis** | | |
| Linhas não devem ser tão longas | 10.565 | Média |
| Linhas não devem terminar com espaços em branco | 10.461 | Baixa |
| Strings literais não devem ser duplicadas | 3.765 | Média |
| Nomes de campos devem estar de acordo com os padrões de nomeação | 3.730 | Muito baixa |
| Copyright e licença devem estar definidos em todos os cabeçalhos de arquivos | 1.747 | Baixa |
| **SGF** | | |
| Código-fonte deve ser corretamente endentado | 5.205 | Baixa |
| Abre-chaves devem ser localizadas no início de uma linha | 3.058 | Baixa |
| Comentário requerido | 2.742 | Baixa |
| Lei de Demeter | 1.943 | Média |
| Design para extensão | 1.823 | Baixa |
| **Web Integrator** | | |
| Código duplicado | 48,932 | Média |
| Código-fonte deve ser corretamente endentado | 16,332 | Baixa |
| Abre-chaves devem ser localizadas no início de uma linha | 10,021 | Baixa |
| Lei de Demeter | 6,236 | Média |
| This é requerido | 5.982 | Média |

Tabela 9 - As cinco violações de regras mais frequentes dos projetos do Órgão da APF participante do estudo de caso

| **Violação de regra** | **Número de ocorrências** | **Severidade** |
|---|---|---|
| **CPROD** | | |
| Código-fonte deve ser corretamente endentado | 12.691 | Baixa |
| Abre chaves deve ser localizado no início da linha | 7.501 | Baixa |
| Lei de Demeter | 7.073 | Média |

| | | |
|---|---|---|
| Comentário requerido | 6.676 | Baixa |
| Design para extensão | 3.995 | Baixa |
| **DSCOM** | | |
| Linhas não devem terminar com espaço | 853 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 488 | Média |
| If/else/for/whilo/do devem sempre usar chaves | 296 | Baixa |
| Linhas não devem ser tão longas | 235 | Média |
| Nomes de funções devem estar de acordo com a convenção de codificação | 132 | Média |
| **FAPTI** | | |
| Linhas não devem terminar com espaço | 10.290 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 7.904 | Média |
| If/else/for/whilo/do devem sempre usar chaves | 3.074 | Baixa |
| Instruções devem estar em linhas separadas | 2.425 | Baixa |
| Nomes de funções devem estar de acordo com a convenção de codificação | 1.796 | Média |
| **REPNBL** | | |
| O código-fonte deve estar corretamente endentado | 7.000 | Baixa |
| Lei de Demeter | 7.764 | Média |
| Abre-chaves devem ser localizadas no início da linha | 4.170 | Baixa |
| Comentário requerido | 3.228 | Baixa |
| Linhas de código não devem ser tão longas | 2.429 | Baixa |
| **SAF** | | |
| O código-fonte deve estar corretamente endentado | 2.396 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 1.372 | Baixa |
| Comentário requerido | 935 | Baixa |
| Lei de Demeter | 935 | Média |
| This é requerido | 871 | Média |
| **SCOP** | | |
| O código-fonte deve estar corretamente endentado | 3.316 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 1.918 | Baixa |
| Comentário requerido | 1.709 | Baixa |
| Design para extensão | 1.042 | Baixa |
| This é requerido | 1.010 | Média |
| **SIF** | | |
| O código-fonte deve estar corretamente endentado | 850 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 512 | Baixa |
| Comentário requerido | 425 | Baixa |
| Lei de Demeter | 358 | Média |
| This é requerido | 299 | Média |
| **SISTP** | | |
| O código-fonte deve estar corretamente endentado | 2.408 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 1.486 | Baixa |
| Comentário requerido | 1,410 | Baixa |
| Espaços em branco em volta | 1.088 | Baixa |
| Design para extensão | 758 | Baixa |
| **SOE** | | |
| O código-fonte deve estar corretamente endentado | 812 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 475 | Baixa |
| Comentário requerido | 435 | Baixa |
| Design para extensão | 283 | Baixa |
| Tipos públicos, métodos e campos devem ser documentados com Javadoc | 208 | Baixa |
| **SRAPD** | | |
| O código-fonte deve estar corretamente endentado | 2.808 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 1.599 | Baixa |
| Comentário requerido | 1.435 | Baixa |
| Design para extensão | 871 | Baixa |

| | | |
|---|---|---|
| Lei de Demeter | 843 | Média |
| **SRH** | | |
| O código-fonte deve estar corretamente endentado | 8.636 | Baixa |
| Abre-chaves devem ser localizadas no início da linha | 4.994 | Baixa |
| Comentário requerido | 4.916 | Baixa |
| Design para extensão | 2.709 | Baixa |
| Lei de Demeter | 2.311 | Média |

### 5.1.1. Análise das violações de regra identificadas

A severidade das violações de regra do Perfil de Qualidade para Órgãos da APF foi definida de acordo com o contexto das contratações de soluções de TI do órgão da APF participante do estudo de caso. Pode-se perceber grandes variações relacionadas ao número de violações de regra identificadas e à severidade associada a cada uma delas quando as análises realizadas são comparadas.

A severidade "Muito Alta" foi atribuída para as violações de regra relacionadas à segurança e às vulnerabilidades maliciosas de código, essas violações de regra são identificadas pela ferramenta FindBugs. Utilizando todas as regras do SonarQube foram identificadas violações de regra classificadas como de severidade "Muito Alta" nos projetos do Portal do Software Público Brasileiro, mas quando a análise foi feita utilizando o Perfil de Qualidade para Órgãos da APF não foram identificadas violações de regra classificadas como de severidade "Muito Alta" (Fig. 11). Esse resultado está de acordo com o que era esperado, pois não foi possível obter o byte code dos projetos do Portal do software Público Brasileiro, o que impossibilitou a análise utilizando a ferramenta FindBugs.

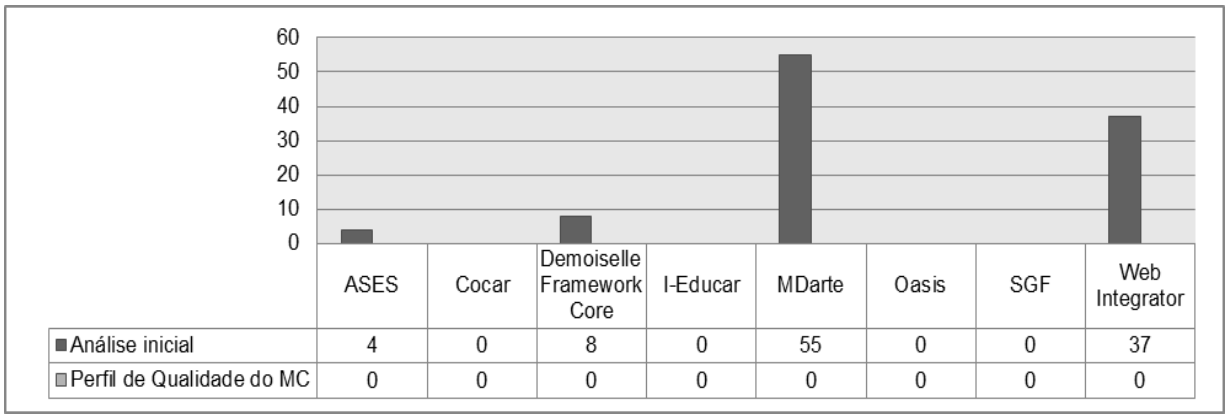
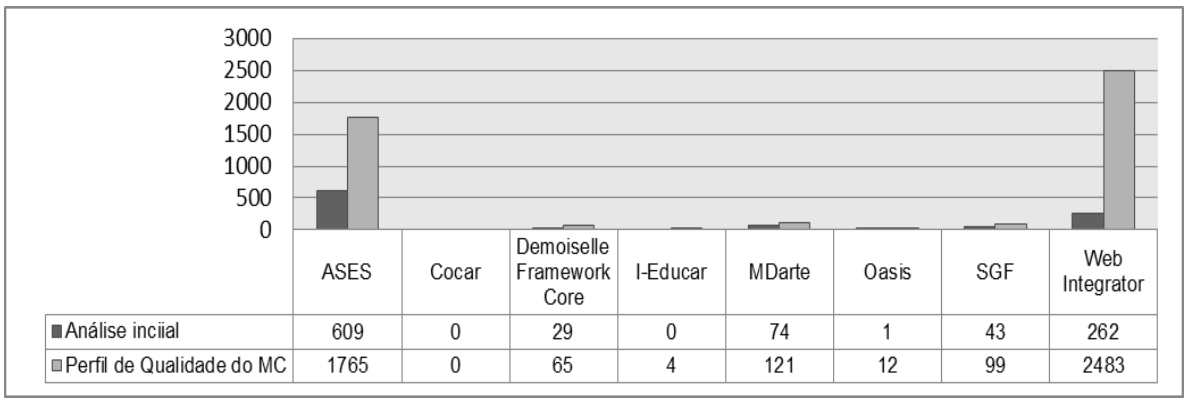



Figura 11 - Violações de regra de severidade "Muito Alta" identificadas nos projetos do Portal do Software Público Brasileiro

Apenas os projetos CPROD e SAF do Órgão da APF participante do estudo de caso foram analisados utilizando a ferramenta FindBugs porém não foram

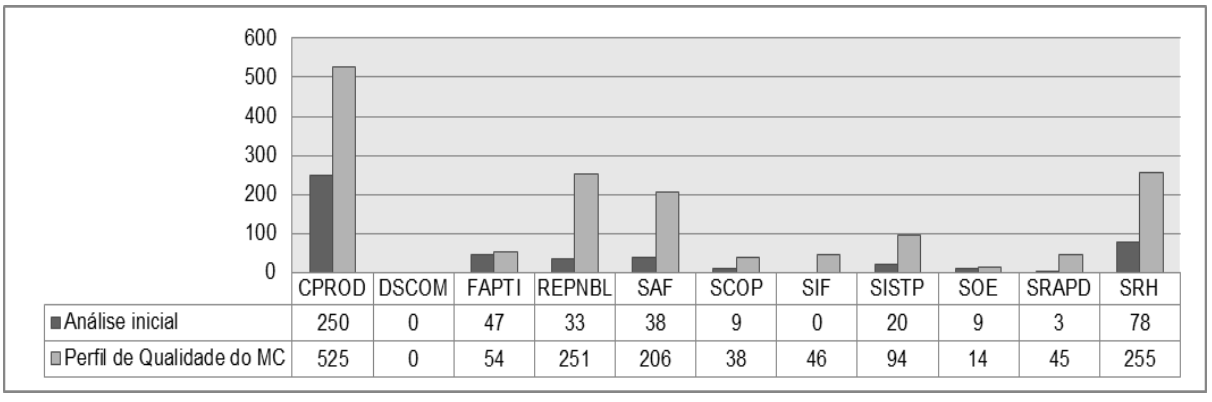
encontradas violações de regra com severidade “Muito Alta” na análise utilizando todas as regras do SonarQube ou na análise utilizando o Perfil de Qualidade para Órgãos da APF.

A severidade “Alta” foi atribuída para as violações de regra relacionadas à más práticas de programação, corretude, ineficiências, bugs, bugs em potencial e indicativo de erros de programação. Utilizando o Perfil de Qualidade para Órgãos da APF foi identificado um número maior de violações de regra classificadas como de severidade ”Alta” nos projetos do Portal do Software Público Brasileiro e também nos projetos Órgão da APF participante do estudo de caso (Figs. 12 e 13). Esse resultado é devido à atribuição de severidade “Alta” às violações de regra que antes eram classificadas apenas como de severidade “Média”.




| | ASES | Cocar | Demoiselle Framework Core | I-Educar | MDarte | Oasis | SGF | Web Integrator |
|---|---|---|---|---|---|---|---|---|
| Análise inciial | 609 | 0 | 29 | 0 | 74 | 1 | 43 | 262 |
| Perfil de Qualidade do MC | 1765 | 0 | 65 | 4 | 121 | 12 | 99 | 2483 |

Figura 12 - Violações de regra de severidade “Alta” identificadas nos projetos do Portal do Software Público Brasileiro

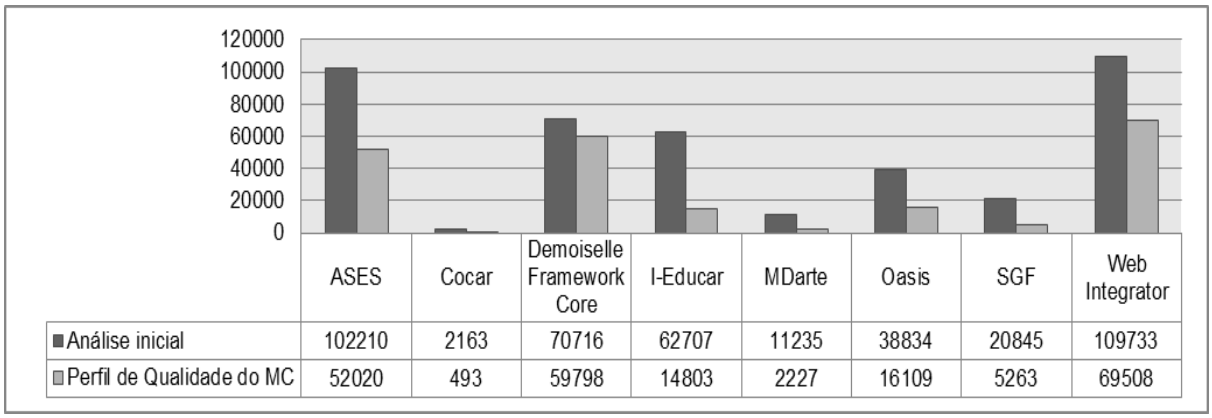



| | CPROD | DSCOM | FAPTI | REPNBL | SAF | SCOP | SIF | SISTP | SOE | SRAPD | SRH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Análise inicial | 250 | 0 | 47 | 33 | 38 | 9 | 0 | 20 | 9 | 3 | 78 |
| Perfil de Qualidade do MC | 525 | 0 | 54 | 251 | 206 | 38 | 46 | 94 | 14 | 45 | 255 |

Figura 13 - Violações de regra de severidade “Alta” identificadas nos projetos do Órgão da APF participante do estudo de caso

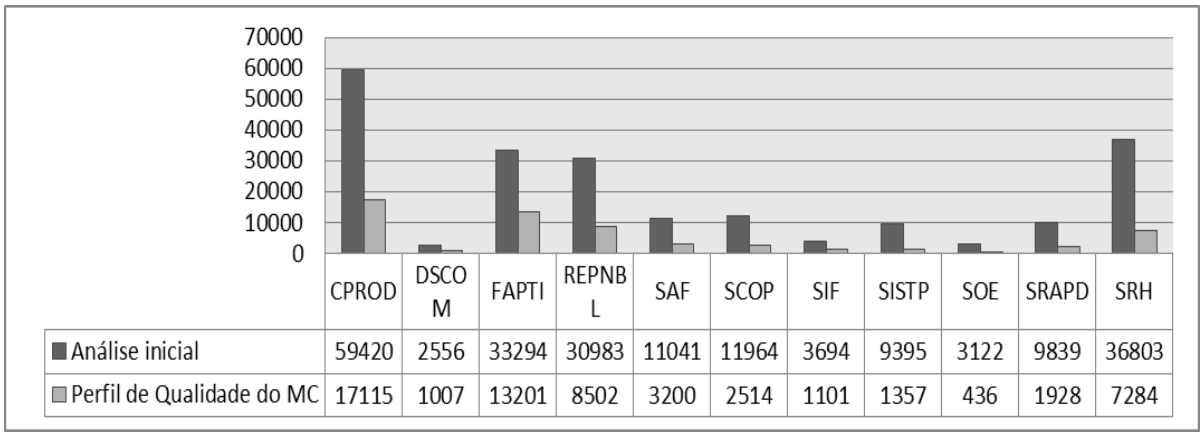
A severidade “Média” foi atribuída para as violações de regra relacionadas a más práticas de programação, corretude, ineficiências, futuros erros de programação, indicativos de erro de programação, inconsistências de estilo, bugs e

bugs em potencial. Utilizando o Perfil de Qualidade para Órgãos da APF foi identificado um número menor de violações de regra classificadas como de severidade ”Média” nos projetos do Portal do Software Público Brasileiro e também nos projetos do Órgão da APF participante do estudo de caso (Figs. 14 e 15). Esse resultado é devido à atribuição de severidade “Alta” à violações de regra que antes eram classificadas apenas como “Média”.




| | ASES | Cocar | Demoiselle Framework Core | I-Educar | MDarte | Oasis | SGF | Web Integrator |
|---|---|---|---|---|---|---|---|---|
| Análise inicial | 102210 | 2163 | 70716 | 62707 | 11235 | 38834 | 20845 | 109733 |
| Perfil de Qualidade do MC | 52020 | 493 | 59798 | 14803 | 2227 | 16109 | 5263 | 69508 |

Figura 14 - Violações de regra de severidade “Média” identificadas nos projetos do Portal do Software Público Brasileiro

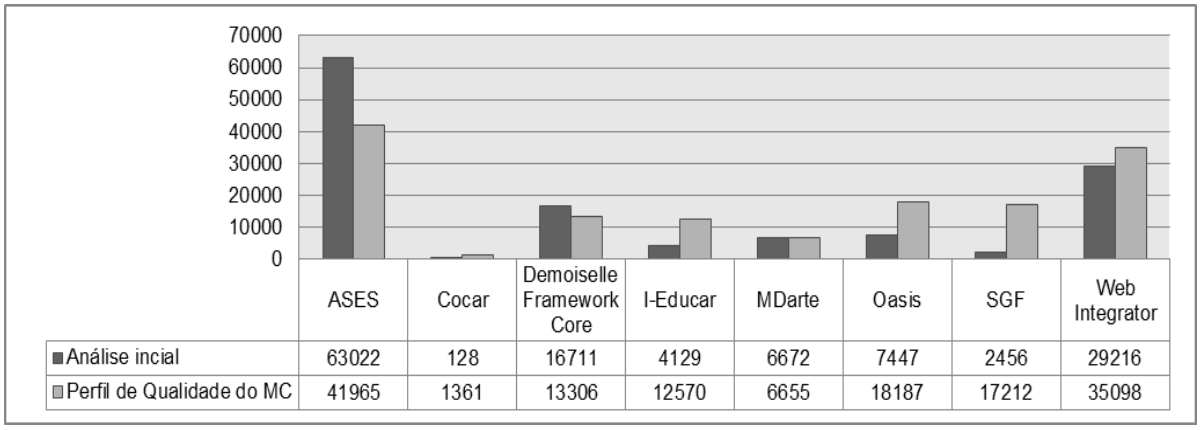



| | CPROD | DSCOM | FAPTI | REPNBL | SAF | SCOP | SIF | SISTP | SOE | SRAPD | SRH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Análise inicial | 59420 | 2556 | 33294 | 30983 | 11041 | 11964 | 3694 | 9395 | 3122 | 9839 | 36803 |
| Perfil de Qualidade do MC | 17115 | 1007 | 13201 | 8502 | 3200 | 2514 | 1101 | 1357 | 436 | 1928 | 7284 |

Figura 15 - Violações de regra de severidade “Média” identificadas nos projetos do Órgão da APF participante do estudo de caso

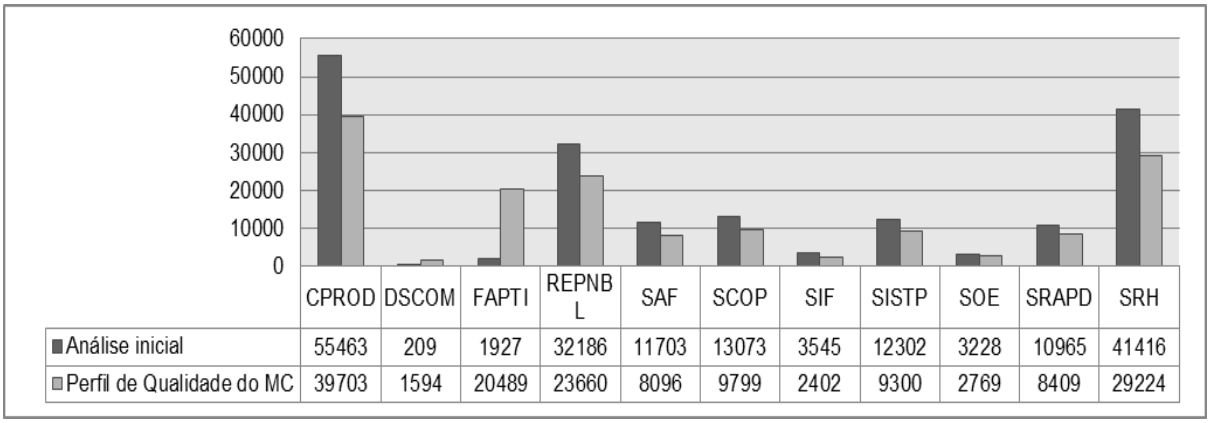
A severidade “Baixa” foi atribuída para as violações de regra relacionadas a inconsistências de estilo, corretude, ineficiências e indicativos de erro de programação. Utilizando o Perfil de Qualidade para Órgãos da APF foram identificadas variações entre as análises realizadas (Figs. 16 e 17). Para alguns projetos o Perfil de Qualidade para Órgãos da APF identificou mais violações de regra do que o que foi identificado na análise inicial e em outros casos o número violações de regra identificadas é menor. Esse resultado é devido à atribuição de

severidade “Baixa” principalmente às regras relacionadas às inconsistências de estilo como nomeação de variáveis e de métodos, essas regras antes eram classificadas como de severidade “Média”.




| | ASES | Cocar | Demoiselle Framework Core | I-Educar | MDarte | Oasis | SGF | Web Integrator |
|---|---|---|---|---|---|---|---|---|
| Análise incial | 63022 | 128 | 16711 | 4129 | 6672 | 7447 | 2456 | 29216 |
| Perfil de Qualidade do MC | 41965 | 1361 | 13306 | 12570 | 6655 | 18187 | 17212 | 35098 |

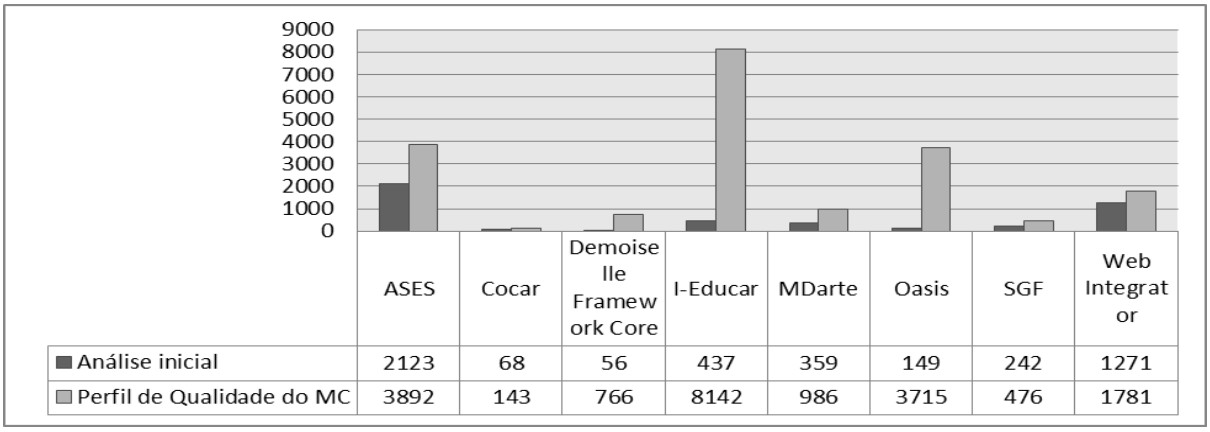
Figura 16 - Violações de regra de severidade “Baixa” identificadas nos projetos do Portal do Software Público Brasileiro




| | CPROD | DSCOM | FAPTI | REPNBL | SAF | SCOP | SIF | SISTP | SOE | SRAPD | SRH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Análise inicial | 55463 | 209 | 1927 | 32186 | 11703 | 13073 | 3545 | 12302 | 3228 | 10965 | 41416 |
| Perfil de Qualidade do MC | 39703 | 1594 | 20489 | 23660 | 8096 | 9799 | 2402 | 9300 | 2769 | 8409 | 29224 |

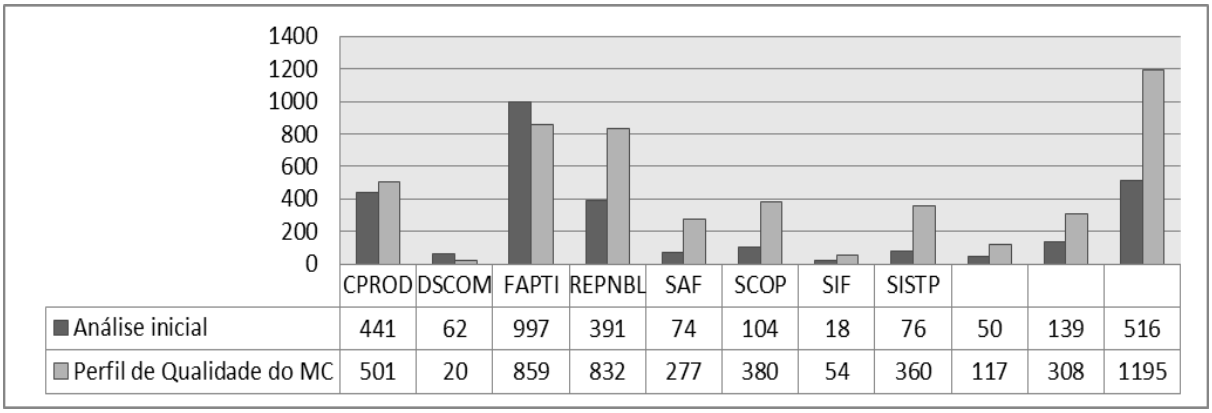
Figura 17 - Violações de regra de severidade “Baixa” identificadas nos projetos do Órgão da APF participante do estudo de caso

A severidade “Muito Baixa” foi atribuída para as violações de regra relacionadas a inconsistências de estilo, principalmente às regras que tratam de ordem de imports, modificadores e de espaços em branco. Utilizando o Perfil de Qualidade para Órgãos da APF foram identificadas variações entre as análises realizadas (Figs. 18 e 19). Esse resultado é devido à atribuição de severidade “Muito Baixa” principalmente às regras relacionadas às inconsistência de estilo que antes eram classificadas como “Baixa”.




| | ASES | Cocar | Demoiselle Framework Core | I-Educar | MDarte | Oasis | SGF | Web Integrator |
|---|---|---|---|---|---|---|---|---|
| Análise inicial | 2123 | 68 | 56 | 437 | 359 | 149 | 242 | 1271 |
| Perfil de Qualidade do MC | 3892 | 143 | 766 | 8142 | 986 | 3715 | 476 | 1781 |

Figura 18 - Violações de regra de severidade “Muito Baixa” identificadas nos projetos do Portal do Software Público




| | CPROD | DSCOM | FAPTI | REPNBL | SAF | SCOP | SIF | SISTP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Análise inicial | 441 | 62 | 997 | 391 | 74 | 104 | 18 | 76 | 50 | 139 | 516 |
| Perfil de Qualidade do MC | 501 | 20 | 859 | 832 | 277 | 380 | 54 | 360 | 117 | 308 | 1195 |

Figura 19 - Violações de regra de severidade “Muito Baixa” identificadas nos projetos do Órgão da APF participante do estudo de caso

O Perfil de Qualidade para Órgãos da APF evidencia um conjunto de violações de regras que impactam negativamente os produtos gerados pelas contratações de soluções de TI. O Perfil de Qualidade para Órgãos da APF pode ser utilizado como instrumento para verificação dos produtos de software resultantes dos contratos de desenvolvimento e manutenção de software, assim essas violações de regras podem ser evitadas e a aceitação do produto resultado da contratação pode ser feita de forma objetiva e controlada.

## 5.3. UTILIZAÇÃO EM TERMOS DE REFERÊNCIA PARA CONTRATAÇÕES DE SOLUÇÕES DE TI

Foram analisados alguns editais de licitação recentes que tratam da contratação de empresas para a execução de serviços de desenvolvimento,

manutenção e documentação de sistemas de informação para identificar a utilização de critérios de verificação de software para a aceitação dos produtos entregues. Os editais analisados foram o do Ministério do Planejamento, Orçamento e Gestão de 2012[13], o do Supremo Tribunal Federal de 2012[14] e do Tribunal de Contas da União de 2013[15]. Todos os editais de contratação de serviços de desenvolvimento, manutenção e documentação de sistemas analisados não definem métricas objetivas para analisar a qualidade do produto entregue. O edital do Ministério do Planejamento, Orçamento e Gestão de 2012 prevê a utilização de análise estática de código, mas não determina como a utilização da análise estática impacta na aceitação do produto.

O SonarQube apresenta o conceito de “Quality Gates” para a determinação dos níveis aceitáveis de violações de regras e de outras métricas analisadas pelo SonarQube (Fig. 20). Pode-se definir “Quality Gates” de acordo com os níveis aceitáveis para cada tipo de violação de regra. As discussões envolvendo os “Quality Gates” realizadas com a equipe do Órgão da APF participante do estudo de caso envolvem a quantidade de linhas e também o tamanho de pontos de função dos projetos. Por exemplo, pode-se definir que serão aceitas no máximo 60 violações de segurança de severidade “Alta” em projetos que possuam 10 mil linhas de código ou mais, porém para projetos com 5 mil linhas de código ou menos serão aceitas apenas 15 violações de segurança de severidade “Alta”.

O conceito de “Quality Gates” pode ser aplicado ao gerenciamento dos níveis de serviços que disponibiliza uma série de práticas que possibilitam não só a gestão da qualidade dos serviços realizados pelos fornecedores, mas também o cumprimento adequado do acordo do nível de serviço (CORBETT, 2010). O gerenciamento deve abranger: (i) o entendimento sobre os processos por meio da seleção de parâmetros e indicadores; (ii) a definição, em conjunto com os fornecedores, dos indicadores mais adequados para a gestão; (iii) a utilização das ferramentas para o monitoramento com objetivos de prover, divulgar e armazenar as informações e, finalmente; (iv) a definição das metas, parâmetros e periodicidade de monitoração dos indicadores (CRISTOFOLI, 2011).

---

13 http://www.planejamento.gov.br/secretarias/upload/Arquivos/licitacoes/pregao/2012/12_LICI_Pregao56_edital.pdf

14 http://www.stf.jus.br/portal/edital/identificarInteressado.asp?licitacao=20401&andamento=25101

15 http://portal2.tcu.gov.br/portal/page/portal/TCU/comunidades/licitacoes_contratos_tcu/licitacoes/concluidas/Edital_111.pdf

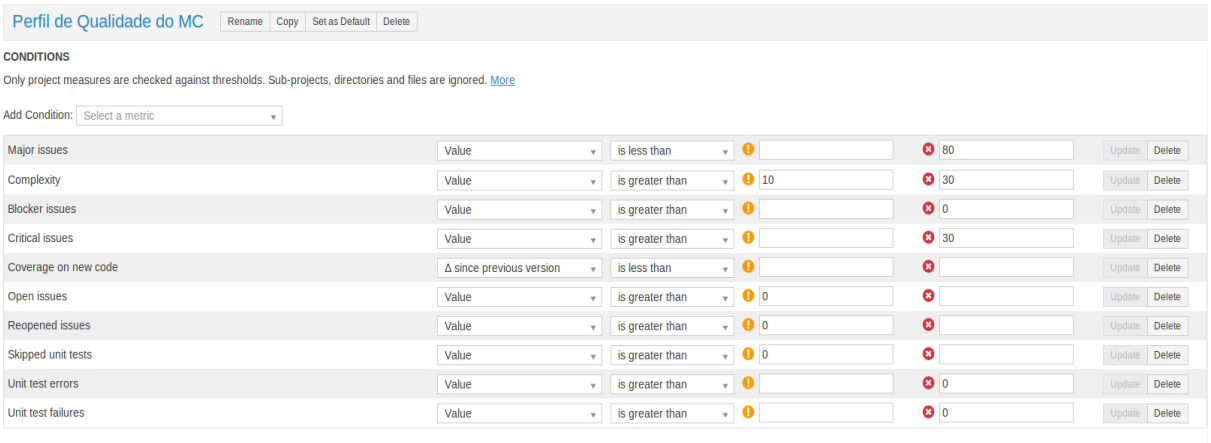



Figura 20 - Definição dos “Quality Gates” do SonarQube

A definição dos níveis de violações de regra aceitáveis só pode ser feito se baseado no histórico das análises realizadas ao longo do tempo. Deve-se levar em consideração as análises realizadas ao longo dos projetos para verificar se o perfil de qualidade utilizado é adequado e se as violações de regra indicadas são relevantes. A partir dessa análise pode-se utilizar os níveis de violações de regra como critérios objetivos a serem utilizados nos Termos de Referência das contratações de desenvolvimento e manutenção de software.

### 1.3.1. Proposta do uso da análise de código-fonte no processo de contratações de software do órgão da APF participante do estudo de caso

Como resultado da parceria da Universidade de Brasília com o órgão da APF participante do estudo de caso, foi desenvolvido o Processo de Gestão de Demandas de Desenvolvimento de Software Ágil. O processo foi desenvolvido com o objetivo de implementar práticas ágeis às contratações de soluções de TI do Órgão da APF participante do estudo de caso (Fig. 21).

=

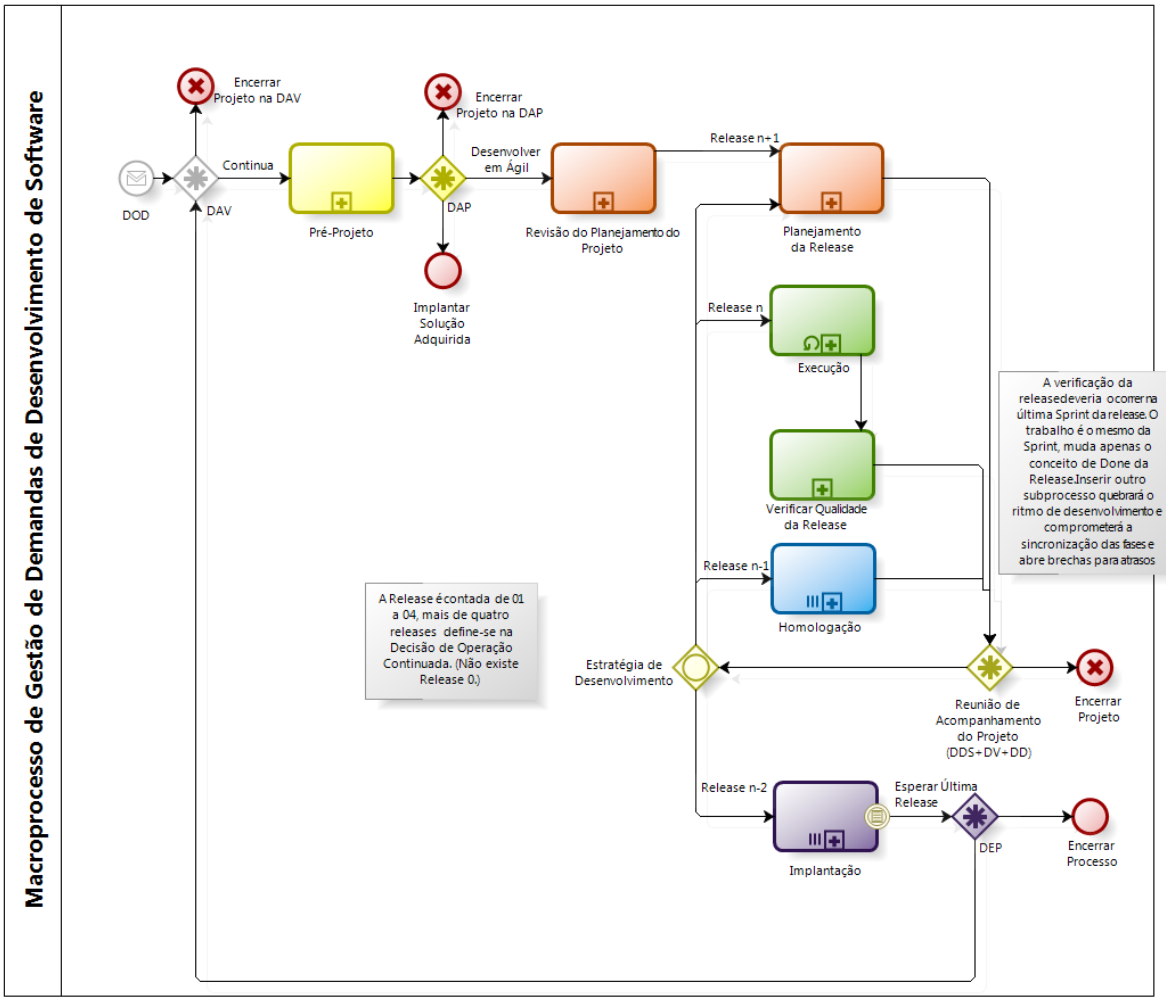



Figura 21 - Visão geral do Processo de Gestão de Demandas de Desenvolvimento de Software Ágil desenvolvido em parceria com a Universidade de Brasília

O Processo de Gestão de Demandas de Desenvolvimento de Software Ágil prevê atividades de verificação da qualidade do produto entregue. São elas “Verificar a Qualidade da Release”, que engloba testes mais amplos como os de integração, e “Verificar Qualidade do Incremento de Software”, que trata da verificação formal da qualidade do produto recebido provisoriamente, deve-se verificar se os critérios de qualidade definidos foram respeitados, relatar as não conformidades e o impacto nos acordos de níveis de serviço. (Fig. 22).

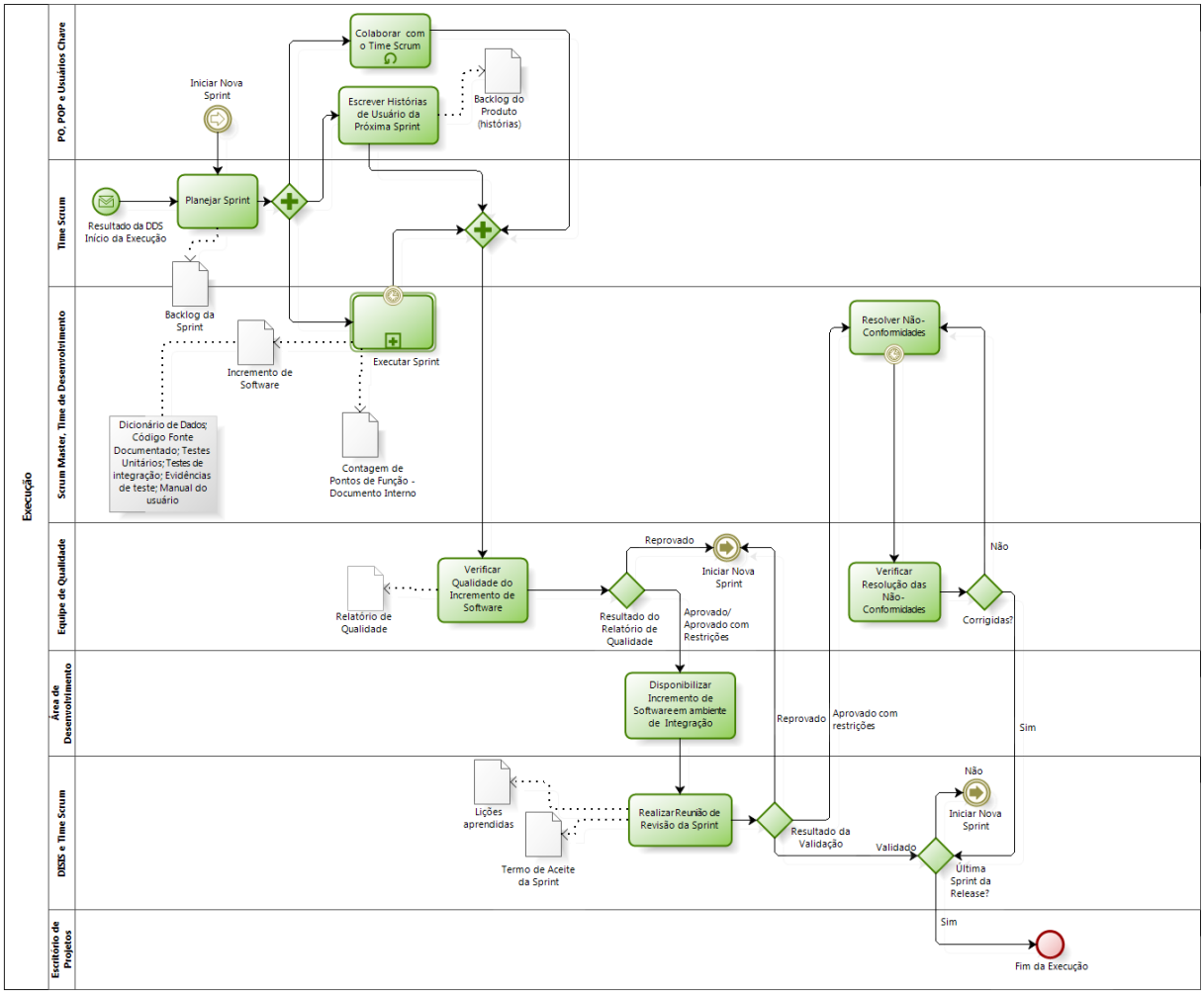



Figura 22 - Atividade “Execução” do Processo Ágil de Gestão de Demandas

A verificação do código-fonte utilizando o Perfil de Qualidade para Órgãos da APF pode ser realizada na atividade “Verificar a Qualidade do Incremento de Software”. A cada incremento entregue pode-se executar a análise do código-fonte e realizar o acompanhamento da evolução das violações de regra identificadas. Pode-se também estabelecer parâmetros para os incrementos de software como, por exemplo, o número de violações de regra consideradas de severidade “Muito Alta”, “Alta”, “Média”, “Baixa” e “Muito Baixa” que serão aceitas em cada incremento. Além disso, pode-se também estabelecer o número de correções mínimas necessárias para a aceitação dos próximos incrementos. Dessa forma, a verificação do produto de software entregue é feito de forma controlada e objetiva.

## 5.4. CONSIDERAÇÕES FINAIS DO CAPÍTULO

A definição de regras relacionadas ao código-fonte dos produtos resultantes das contratações permite que a avaliação de qualidade durante a fase de Gestão de Contratos seja feita baseando-se em critérios objetivos de qualidade. A análise e comparação das regras definidas pelo SonarQube para compor o Perfil de Qualidade para Órgãos da APF garante que a verificação do código-fonte considera os problemas comuns e relevantes aos softwares desenvolvidos no contexto do Órgão da APF participante do estudo de caso e a partir de terceirizações por meio de Fábricas de Software.

Porém, essa primeira versão do Perfil de Qualidade para Órgãos da APF é apenas um ponto de partida para a monitoração da qualidade dos produtos entregues pelas empresas contratadas. É necessário que as regras que compõem o Perfil de Qualidade para Órgãos da APF sejam constantemente revisadas, atualizadas e adequadas à realidade dos contratos de desenvolvimento e manutenção em execução. Além disso, outras ferramentas que apoiam o processo de verificação e validação dos produtos entregues devem ser adotadas para garantir a qualidade de software por meio de análise de parâmetros objetivos e controlados.

# 6.CONCLUSÃO

Neste trabalho foi apresentada uma proposta para a verificação da qualidade de código nas contratações de TI da Administração Pública Federal.

Para a definição da técnica de verificação, algumas atividades foram essenciais: a revisão da literatura; a análise de ferramentas para análise estática de código; a definição de um estudo de caso; e a análise dos dados do estudo de caso para definição dos critérios de qualidade.

A revisão da literatura possibilitou a contextualização a respeito das contratações de soluções de TI na APF e a respeito de conceitos relacionados à qualidade de software, especialmente às técnicas de verificação de software como a análise estática de código-fonte.

A análise de ferramentas possibilitou identificar algumas das ferramentas mais utilizadas para a análise estática de código-fonte e decidir qual a mais adequada para o contexto da APF.

Como estudo de caso foram selecionados *softwares* do Portal do Software Público Brasileiro e de um órgão da APF, foi realizada a análise estática do código-desses *softwares* utilizando a ferramenta SonarQube e o resultado da análise foi investigado para a identificação de violações de regras comuns entre esses *softwares*.

A análise dos dados do estudo de caso possibilitou identificar as violações de regras mais comuns em projetos no contexto da APF e, além disso, foram consideradas outras violações de regra relevantes que não foram indicadas na análise inicial. Ao final obteve-se um conjunto de regras baseadas na ISO/IEC 25000 que formam o Perfil de Qualidade para Órgãos da APF.

O Perfil de Qualidade para Órgãos da APF deve ser utilizado durante a fase de Gestão de Contratos para verificar a qualidade de código-fonte do produto de software recebido nas contratações de TI da APF. Pode-se também utilizar o conjunto de regras do Perfil de Qualidade para Órgãos da APF como parâmetros para os critérios de aceitação definidos nos Termos de Referências dos editais de contratações.

A definição do Perfil de Qualidade para Órgãos da APF é apenas um ponto de partida para a monitoração da qualidade dos produtos entregues pelas empresas contratadas. Esse conjunto de regras deve ser monitorado e atualizado de acordo com as necessidades das contratações de TI dos órgãos da APF.

Como trabalhos futuros, objetiva-se a validação e refinamento dos critérios de qualidade selecionados e também a criação de plug-ins para a ferramenta SonarQube para integrar a visualização da análise dos critérios de qualidade ao painel da ferramenta SonarQube.

## Bibliografia

AL-KILIDAR, H.; COX, K.; KITCHENHAM, B. The use and usefulness of the ISO/IEC 9126 quality standard. IEEE, 2005. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1541821>. Acesso em: 14 abr. 2014

BARBOSA, A. F. et al. Governança de TIC e Contratos no Setor Público. In: CONGRESSO ANUAL DE TECNOLOGIA DA INFORMAÇÃO – CATI. São Paulo: 2006

BEAL, Adriana. Segurança da Informação - Princípios e Melhores Práticas para a Proteção dos Ativos de Informação nas Organizações. 1ª Ed. São Paulo: Atlas, 2005. Boas práticas em segurança da informação. Tribunal de Contas da União do Brasil. 3ª ed. Brasília: TCU, Secretaria de Fiscalização de Tecnologia da Informação, 2008.

BEMER, R. W. Software systems customized by computer. In: IFIP. 1965

BRASIL. Decreto-Lei No. 200. Dispõe sobre a organização da

Administração Federal, estabelece diretrizes para a Reforma Administrativa e dá outras providências. Disponível em: <http://www.planalto.gov.br/ccivil_03/decreto-leti/Del0200.htm>. Acesso em: 20 jan. 2014.

BRASIL. Lei No. 8.443. Dispõe sobre a Lei Orgânica do Tribunal de Contas da União e dá outras providências. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/L8443.htm>. Acesso em: 27 mar. 2014.

BRASIL. Lei 8.666. Regulamenta o art. 37, inciso XXI, da Constituição Federal, institui normas para licitações e contratos da Administração Pública e dá outras providências. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/L8666cons.htm>. Acesso em: 15 fev. 2014.

BRASIL. Decreto n. 2.271. Dispõe sobre a contratação de serviços pela Administração Pública Federal direta, autárquica e fundacional e dá outras providências. Disponível em: <http://www.planalto.gov.br/ccivil_03/decreto/d2271.htm>. Acesso em: 3 set. 2014.

BRASIL. Ministério da Ciência e Tecnologia. Secretaria de Política de Informática. Pesquisa de Qualidade no Setor de Software Brasileiro. 2010

BRASIL. Instrução Normativa No. 04. Dispõe sobre o processo de contratação de Soluções de Tecnologia da Informação pelos órgãos integrantes do Sistema de Administração dos Recursos de Informação e Informática (SISP) do Poder Executivo Federal. Disponível em: <http://www.governoeletronico.gov.br/biblioteca/arquivos/instrucao-normativa-no-04-de-12-de-novembro-de-2010/download>. Acesso em: 27 jan. 2014b.

BRASIL. Tribunal de Contas da União. Guia de boas práticas em contratação de soluções de tecnologia da informção: riscos e controles para o planejamento da contratação. Tribunal de Contas da União, 2012a.

BRASIL. Ministério do Planejamento. Secretaria de Logística e tecnologia da informação (SLTI). Informações Gerenciais de Contratações Públicas de Bens e Serviços de Tecnologia da Informação. Disponível em: <http://www.comprasnet.gov.br/ajuda/04_-_Compras_de_TI.pdf>. Acesso em: 2 abr. 2014b.

BRASIL. Tribunal de Contas da União. Levantamento de Governança de TI 2012. Disponível em: <http://www.ifam.edu.br/portal/images/file/402.p>. Acesso em: 3 fev. 2014c.

CAMPBELL, G. A.; PAPAPETROU, P. P. SonarQube in Action. Manning, 2014 .

CORBETT, M. F. The outsourcing revolution: why it makes sense and how to do it right.. Chicago: Deaborn Trade Publishing, 2010.

CRISTOFOLI, F. Um estudo sobre a gestão da terceirização de serviçoos de tecnologia de informação baseados em modelos de governança. Tese (Doutorado em Administração). São Paulo: Universidade de São Paulo, 2011.

CRUZ, C. S. DA; ANDRADE, E. L. P. DE; FIGUEIREDO, R. M. DA C. Processo de Contratação de Serviços de Tecnologia da Informação para Organizações Públicas. Ministério da Ciência e Tecnologia. Secretaria de Política de Informática, 2011.

GIL, A. C. Como elaborar projetos de pesquisa. São Paulo: Atlas, 2008.

GLASER, B.; STRAUSS, A. L. The Discovery of Grounded Theory: Strategies for Qualitative Research. Chicago: Aldine Publishing Company, 1967.

HEFLEY, W. E.; LOESCHE, E. A. The eSourcing Capability Model for Client Organizations (eSCM-CL), Part 1: Model Overview. CMU-ITSQC-06-002. Pittsburgh, PA: IT Services Qualification Center. Carnegie Mellon University, 2006.

HEITLAGER, I.; KUIPERS, T.; VISSER, J. A Practical Model for Measuring Maintainability. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4335232>. Acesso em: 13 abr. 2014

IEEE. IEEE Standard for a Software Quality Metrics Methodology. IEEE Std 1061-1998, dez. 1998.

I NBR ISO/IEC 9126-1. ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. Tecnologia da informação – Qualidade de Produto de Software – Parte 1:Modelo de Qualidade. Rio de Janeiro, 2003.

ISO/IEC. Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE. [s.l: s.n.].

ITGI. INFORMATION TECHNOLOGY GOVERNANCE INSTITUTE. COBIT – Control Objectives for Information and related Technology. 5. ed. Rolling Meadows, 2012.

KANELLOPOULOS, Y. et al. Code Quality Evaluation Methodology Using The ISO/IEC 9126 Standard. International Journal of Software Engineering & Applications, v. 1, n. 3, p. 17–36, 26 jul. 2010.

KUIPERS, T.; VISSER, J.; VRIES, G. DE. Monitoring the Quality of Outsourced Software. In proceedings of the Workshop on Tools for Managing Globally Distributed Software Development, 2007.

MARTINS, S. P. A terceirização e o direito do trabalho. 2a Ed. Atlas, 2005.

MCGRAW, G.; CHESS, B.; MIGUES, S. Software [In]security: Third-Party Software and Security, 2011.

NAIK, S.; TRIPATHY, P. Software Testing and Quality Assurance: Theory and Practice. John Wiley & Sons, 2011.

NBR ISO 9000. ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS NBR ISO 9000 - Sistemas de gestão da qualidade - Fundamentos e vocabulário Rio de Janeiro: ABNT, dezembro 2005, 35 pp.

NBR ISO/IEC 12207. ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS NBR ISO/IEC 12207, Tecnologia de Informação – Processos de ciclo de vida de software; Rio de Janeiro ABNT out / 1998. 35 pp. OWASP. Static Code Analysis, 2014.

PRADO, E. P. V.; CRISTOFOLI, F. Resultados da Terceirização da tecnologia da informação em organizações brasileiras. Gestão & Regionalidade, v. Volume 28, n. 84, p. PP–77–88, 2012.

PRESSMAN, R. S. Engenharia de Software. McGraw Hill Brasil, 2011.

SOFTEX. Guia Geral do MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral, 2011. Disponível em: <http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2011.pdf>. Acesso em: 8 fev. 2014.

SOMMERVILLE, I. Engenharia de software. Addison Wesley Bra, 2007.

SOTIROV, A. I. Automatic vulnerability detection static source code analysis. A Master's degree Thesi, 2005

TURBAN, E.; MCLEAN, E.; WETHERBE, J. Tecnologia da informação para gestão. Porto Alegre: Bookman, 2004.

WRIGHT, C. Top three potential risks with outsourcing information systems. Information Systems Control Journal, v. 5, 2005.

YVES, Y.; WOUTER, J.; FRANK, P. A methodology for designing countermeasures against current and future code injection attacks. In: IWIA 2005: PROCEEDINGS OF THE THIRD IEEE INTERNATIONAL INFORMATION ASSURANCE WORKSHOP. Maryland, USA: 2005

**ANEXOS**

ANEXO I: Regras do Perfil de Qualidade Java para Órgãos da APF

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| "equals(Object obj)" should be overridden along with the "compareTo(T obj)" method | Podem ocorrer falhas quando versões diferentes da linguagem Java são utilizadas. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| "java.lang.Error" should not be extended | Erros relacionados à java.lang.Error e suas subclasses representam situações anormais que são encontradas apenas pela Java Virtual Machine (JVM). O uso de java.lang.Error e suas subclasses pode levar à confusão. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| "object == null" should be used instead of "object.equals(null)" | A variável presente no lado esquerdo da comparação pode ser nula e o uso de object.equals(null) não trata essa situação. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| "switch" statements should have at least 3 cases | Quando não existem três ou mais casos para a instrução switch deve-se substituir por instruções if. | Ineficiência | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| “public static" fields should always be constant | Variáveis que devem ser transformadas em constantes. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Abstract class names should comply with a naming convention | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Abstract Class Without Abstract Method | Pode indicar implementação incompleta. | Indicativo de erros de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Abstract classes without fields should be converted to interfaces | Convenção adotada a partir da versão 8 da linguagem Java recomenda que classes abstratas sem campos definidos deve ser convertida em uma interface. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Add Empty String | É uma maneira ineficiente de converter qualquer outro tipo ao tipo String. Pode também indicar implementação incorreta ou incompleta. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Anonymous inner classes containing only one method should become lambdas | A partir da versão 8 da linguagem Java recomenda--se a substituição de classes internas anônimas por lambdas. | Ineficiência | Baixa | Modificabilidade | Manutenabilidade |
| Append Character With Char | Deve-se evitar concatenar caracteres com os métodos que são definidos para serem usados com Strings. | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| Array designators "[]" should be on the type, not the variable | Indicadares de arrays devem ser localizados junto ao tipo para uma melhor legibilidade. | Inconsistência de estilo | Muito baixa | Analisabilidade | Manutenabilidade |
| Assignments should not be made from within sub-expressions | Atribuições em expressões são difíceis de identificar e afetam a legibilidade do código. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Avoid Branching Statement As Last In Loop | Pode indicar um erro de programação e afetam a legibilidade do código. | Indicativo de erros de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Avoid Catching Generic Exception | Captura exceções lançadas pela JVM e pode ignorar possíveis problemas. | Bug em potencial | Muito alta | Maturidade | Confiabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Avoid commented-out lines of code | Linhas de código comentada podem confundir e até mesmo se tornar um possível bug ao se remover o comentário por acidente. | Futuro erro de programação | Média | Analisabilidade | Manutenabilidade |
| Avoid cycle between java packages | Indica alto acoplamento entre pacotes o que afeta a reusabilidade e indica a necessidade de refatoração do código. | Ineficiência | Média | Modificabilidade, Analisabilidade, Testabilidade e Reusabilidade | Manutenabilidade |
| Avoid Final Local Variable | Nesses casos variáveis finais não são necessárias e devem ser transformadas em variáveis locais. | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| Avoid Inline Conditionals | Condicionais em uma mesma linha afetam a legibilidade do código. | Inconsistência de estilo | Muito baixa | Analisabilidade | Manutenabilidade |
| Avoid instantiating objects in loops | Objetos criados em loops podem ocupar espaço desnecessário na memória. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Avoid Protected Field In Final Class | Campos protegidos em classes finais não podem ser usados em subclasses. Deve-se considerar o uso do modificador privado ou alterar a acessibilidade do pacote. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Avoid Star Import | Deve-se evitar importar pacotes que não serão utilizados. | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| Avoid Throwing Null Pointer Exception | Exceções genéricas não dão um diagnóstico preciso do problema e podem ser confundidas com exceções lançadas pela JVM. | Futuro erro de programação | Média | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Avoid too complex class | Classe muito complexa que deve ser refatorada. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Avoid too deep inheritance tree | Árvores heranças muito grandes podem levar a código muito complexo. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Bad comparison of int value with long constant | Compara valores que têm limites diferentes, o que pode levar a erros. | Indicativo de erros de programação | Baixa | Corretude funcional | Adequação funcional |
| Bad practice - Abstract class defines covariant compareTo() method | Erro ao fazer a sobrecarga do método compareTo(). | Bug | Média | Corretude funcional | Adequação funcional |
| Bad practice - Certain swing methods needs to be invoked in Swing thread | Pode causar problemas na sincronização dos threads. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Check for sign of bitwise operation | Comparação do resultado de operações bitwise com o operador “maior que” pode levar a resultados inesperados | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Bad practice - Class defines clone() but doesn't implement Cloneable | Não é necessariamente um erro, mas pode indicar um possível erro de programação | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Bad practice - Class implements Cloneable but does not define or use clone method | Deve-se remover métodos que não são utilizados e reconsiderar a necessidade de implementar a classe Cloneable | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Bad practice - Class inherits equals() and uses Object.hashCode() | Se a classe herda equals() deve-se usar os métodos de hashcode providos por essa classe | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Class is Externalizable but doesn't define a void constructor | Se a classe não define um constructor *void* a serialização e a deserialização irão falar em tempo de execução | Bug | Média | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Bad practice - Class is not derived from an Exception, even though it is named as such | Pode causar confusão para os usuários da classe. | Inconsistência de estilo | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Class is Serializable but its superclass doesn't define a void constructor | Se a classe não define um constructor *void* a serialização e a deserialização irão falar em tempo de execução. | Bug | Média | Corretude funcional | Adequação funcional |
| Bad practice - Class is Serializable, but doesn't define serialVersionUID | Deve-se adicionar explicitamente o serialVersionUID para garantir a interoperabilidade entre versões do código. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Bad practice - Class names shouldn't shadow simple name of implemented interface | Afeta a legilibidade do código e pode causar confusão quando é necessário resolver as referencias das classes. | Inconsistência de estilo | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Bad practice - Class names shouldn't shadow simple name of superclass | Afeta a legilibidade do código e pode causar confusão quando é necessário resolver as referencias das classes. | Inconsistência de estilo | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Bad practice - Clone method may return null | Métodos Clone nunca devem retornar null, o que pode levar a um erro durante a verificação | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Comparator doesn't implement Serializable | Deve-se considerar a necessidade da implementação de Serializable, o que é uma prática de programação defensiva. | Inconsistência de estilo | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Comparison of String objects using == or != | Faz a comparação da referência ao invés do valor, o que pode levar a erros. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Comparison of String parameter using == or != | Faz a comparação da referência ao invés do valor, o que pode levar a erros. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Bad practice - Confusing method names | Nomes confusos que podem levar a erros de programação | Futuro erro de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Covariant compareTo() method defined | Erro ao fazer a sobrecarga do método compareTo(). | Bug | Média | Corretude funcional | Adequação funcional |
| Bad practice - Creates an empty jar file entry | Erro na chamada do método closeEntry() que leva a criação de arquivos jar vazios. | Bug | Muito alta | Corretude funcional | Adequação funcional |
| Bad practice - Creates an empty zip file entry | Erro na chamada do método closeEntry() que leva a criação de arquivos zip vazios. | Bug | Muito alta | Corretude funcional | Adequação funcional |
| Bad practice - Dubious catching of IllegalMonitorStateException | Possível erro de design. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Bad practice - Empty finalizer should be deleted | Método vazio que deve ser excluído. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Equals checks for noncompatible operand | Uso do método *equals* com tipos incompatíveis pode levar a comportamento inesperado. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Bad practice - equals method fails for subtypes | Método *equals* que apresentará erro ao ser usado por subclasses. | Bug | Muito alta | Corretude funcional | Adequação funcional |
| Bad practice - Equals method should not assume anything about the type of its argument | Apresenta erros caso o método *equals* não trate comparações com tipos incompatíveis. | Bug | Muito alta | Corretude funcional | Adequação funcional |
| Bad practice - equals() method does not check for null argument | Podem ocorrer erros durante a comparação se o método *equals* não checa valores nulos. | Bug em potencial | Muito alta | Corretude funcional | Adequação funcional |
| Bad practice - Fields of immutable classes should be final | Por convenção, os campos de classes imutáveis também devem ser imutáveis. | Inconsistência de estilo | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Finalizer does not call superclass finalizer | As ações de finalização definidas pela superclasse não serão executadas, o que pode causar comportamentos | Bug em potencial | Muito alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| | inesperados. | | | | |
| Bad practice - Finalizer does nothing but call superclass finalizer | Método redundante que deve ser removido. | Indicativo de erros de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Finalizer nullifies superclass finalizer | Deve-se verificar o uso do finalizador. | Indicativo de erros de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Bad practice - Finalizer nulls fields | Erro que anula campos do finalizador. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Bad practice - Iterator next() method can't throw NoSuchElementException | Implementação inadequada que não capacita o método *next()* a lançar exceções. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method doesn't override method in superclass due to wrong package for parameter | Erro que impede a sobrecarga de parâmetros. | Bug | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method ignores exceptional return value | Pode levar a comportamentos inesperados quando o retorno dos métodos não são checados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method ignores results of InputStream.read() | Pode levar a comportamentos inesperados pois ignora valores do *input strem.* | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method ignores results of InputStream.skip() | Pode levar a comportamentos inesperados pois ignora valores do *input strem.* | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method invoked that should be only be invoked inside a doPrivileged block | Quando o código é invocado por códigos que não fazem verificação de permissões deve-se fazer as operações dentro de um bloco doPrivileged para evitar falhas de segurança. | Bug em potencial | Média | Confidencialidade | Segurança |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Bad practice - Method invokes dangerous method runFinalizersOnExit | Chamada de método que pode levar a comportamento inesperado. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method may fail to close database resource | Resulta em ineficiência de performance e pode causar problemas de comunicação com o banco de dados | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Bad practice - Method may fail to close database resource on exception | Resulta em ineficiência de performance e pode causar problemas de comunicação com o banco de dados | Bug | Muito alta | Utilização de recursos | Eficiência de performance |
| Bad practice - Method may fail to close stream | Resulta em ineficiência de performance e pode causar a perda do manipulador de arquivos | Bug | Muito alta | Utilização de recursos | Eficiência de performance |
| Bad practice - Method may fail to close stream on exception | Resulta em ineficiência de performance e pode causar a perda do manipulador de arquivos | Bug | Muito alta | Utilização de recursos | Eficiência de performance |
| Bad practice - Method might drop exception | O método não trata a exceção e também impede que ela seja tratada, o que pode levar a comportamentos inesperados | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method might ignore exception | O método ignora a exceção, o que pode levar a comportamentos inesperados | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Method with Boolean return type returns explicit null | Implementação inadequada que pode levar à *NullPointerException* | Bug | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Needless instantiation of class that only supplies static methods | Instanciação desnecessária que deve ser evitada. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Bad practice - Non-serializable class has a serializable inner class | Pode levar a erros em tempo de execução, pois a classe tenta serializar a classe interna. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Bad practice - Non-serializable value stored into instance field of a serializable class | Pode levar a erros em tempo de execução, pois a classe tenta serializar os seus atributos | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Random object created and used only once | Deve-se guardar os valores aleatórios que foram gerados para evitar a repetição de valores | Ineficiência | Média | Corretude funcional | Adequação funcional |
| Bad practice - serialVersionUID isn't final | Se o UID é usado para serialização, então deve sempre ser do tipo *final.* | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Bad practice - serialVersionUID isn't long | Se o UID é usado para serialização, então deve sempre ser do tipo *long* | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - serialVersionUID isn't static | Se o UID é usado para serialização, então deve sempre ser do tipo *static* | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Static initializer creates instance before all static final fields assigned | Cria uma instância da classe antes de todos os campos serem atribuídos, o que pode levar a comportamentos inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Store of non serializable object into HttpSession | Pode levar a erros quando a sessão é migrada. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Superclass uses subclass during initialization | Possível uso de subclasse que ainda não foi inicializada. | Bug | Média | Corretude funcional | Adequação funcional |
| Bad practice - Suspicious reference comparison | Compara a referência em vez do valor. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - toString method may return null | Deve-se evitar retornar valores nulos, pois pode levar a comportamentos inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Bad practice - Transient field that isn't set by deserialization. | Se o campo não é marcado como *transient* seu valor nunca será atualizado, o que pode levar a comportamentos inesperados | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Bad practice - Unchecked type in generic call | Tipos que não são checados podem levar a comportamentos inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Bad practice - Usage of GetResource may be unsafe if class is extended | Pode levar a resultados inesperados se a classe for herdada em outro pacote. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Bean Members Should Serialize | Boa prática que determina que beans devem ser serializados | Inconsistência de estilo | Muito baixa | Modificabilidade | Manutenabilidade |
| Big Integer Instantiation | Não é necessário criar instâncias de valores já existentes. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Boolean Get Method Name | Convenção de nomeação que facilita o entendimento do código. | Inconsistência de estilo | Muito baixa | Analisabilidade | Manutenabilidade |
| Broken Null Check | Checagem que sempre irá lançar *NullPointerException*. | Bug | Alta | Corretude funcional | Adequação funcional |
| Call Super In Constructor | Boa prática que determina a chamada do método super() em construtores | Inconsistência de estilo | Muito baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Case insensitive string comparisons should be made without intermediate upper or lower casing | Requer a criação de objetos temporaries, o que não é eficiente. | Ineficiência | Média | Utilização de recursos | Eficiência de Performance |
| Class variable fields should not have public accessibility | Deve-se respeitar os princípios de encapsulamento. | Inconsistência de estilo | Baixa | Confidencialidade | Seguranca |
| Classes should not be coupled to too many other classes (Single Responsibility Principle) | Classes com acoplamento alto que devem ser refatoradas. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Collapsible "if" statements should be merged | Condicionais que devem ser mescladas para melhorar a legibilidade do código | Ineficiência | Média | Analisabilidade | Manutenabilidade |
| Collection.isEmpty() should be used to test for emptiness | Deve-se usar Collection.isEmpty() ao invés de Collection.size() para verificação de coleções. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Comment Size | Linhas de comentários muito longas que devem ser refatoradas. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Comments should not be located at the end of lines of code | Afeta a legibilidade do código. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Confusing Ternary | Boa prática que recomenda a não utilização de negações em condicionais que possuem *else.* | Futuro erro de programação | Média | Analisabilidade | Manutenabilidade |
| Consecutive Appends Should Reuse | Deve-se usar o objeto alvo para melhorar a performance da operação. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Consecutive Literal Appends | Concatenações repetidas que podem afetar a performance, deve-se refatorar. | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| Constant names should comply with a naming convention | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply | Afeta a legibilidade do código e indicam implementações do tipo "código espaguete" | Futuro erro de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Correctness - "." used for regular expression | Deve-se verificar possíveis erros no uso de expressões regulares. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - A collection is added to itself | Se o *hashcode* for computado irá resultar em *StackOverflowException* | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - A known null value is checked to see if it is an instance of a type | O teste irá sempre retornar falso, deve-se verificar se não é um erro de programação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - A parameter is dead upon entry to a method but overwritten | Atribuição de valores a parâmetros indica erros de programação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - An apparent infinite loop | Loop infinito que deve ser corrigido. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - An apparent infinite recursive loop | Loop infinito que deve ser corrigido. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Apparent method/constructor confusion | Nomes de métodos que podem indicar erros na definição de construtores. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Array formatted in useless way using format string | Deve-se usar Arrays.asList(...) antes da formatação como Strings. | Ineficiência | Média | Corretude funcional | Adequação funcional |
| Correctness - Bad attempt to compute absolute value of signed 32-bit hashcode | Essa implementação retorna o valor negativo. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Bad attempt to compute absolute value of signed 32-bit random integer | Essa implementação retorna o valor negativo. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Bad comparison of nonnegative value with negative constant | Compara valor negativo com outro valor que já se sabe ser negativo. | Ineficiência | Alta | Utilização de recursos | Eficiência de performance |
| Correctness - Bad comparison of signed byte | Comparação de valores com limites diferentes. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Bad constant value for month | Passagem de parâmetro incorreto. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Bitwise add of signed byte value | Comparação de valores com limites diferentes. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Bitwise OR of signed byte value | Indicativo de operações com valores incorretos. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Call to equals() comparing different interface types | Comparação de tipos diferentes. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Call to equals() comparing different types | Comparação de tipos diferentes. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Call to equals() comparing unrelated class and interface | Comparação de tipos diferentes. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Call to equals() with null argument | Comparação utilizando valores nulos. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Can't use reflection to check for presence of annotation without runtime retention | Boa prática no uso de reflexões. | Inconsistência de estilo | Média | Corretude funcional | Adequação funcional |
| Correctness - Check to see if ((...) & 0) == 0 | Expressão que sempre retorna verdadeiro e deve ser revista. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Class defines field that masks a superclass field | Nomeação confusa de campos afeta a legibilidade do código. | Indicativo de erros de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Class overrides a method implemented in super class Adapter wrongly | Sobrecarga incorreta de método que pode levar a comportamentos inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - close() invoked on a value that is always null | Instrução incorreta que lança *NullPointerException*. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Dead store of class literal | Código inutilizado que deve ser excluído. | Ineficiência | Alta | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Deadly embrace of non-static inner class and thread local | Implementação incorreta que pode levar a criação de objetos que não serão utilizados e também não serão elegíveis para o *garbage collector*. | Bug em potencial | Ineficiência | Corretude funcional | Adequação funcional |
| Correctness - Don't use removeAll to clear a collection | Boa prática de programação que recomenda o uso de clear para limpar coleções. | Ineficiência | Alta | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Doomed attempt to append to an object output stream | Possível erro na utilização de arquivos. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Doomed test for equality to NaN | Checagem incorreta de valores aos quais são atribuídos o valor especial NaN. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Double assignment of field | Atribui o mesmo valor duas vezes. Pode indicar um erro de programação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Double.longBitsToDouble invoked on an int | Possível utilização incorreta do método long.BitsToDouble. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals method always returns false | Método que sempre retorna falso indica um possível erro de programação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals method always returns true | Método que sempre retorna verdadeiro indica um possível erro de programação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals method compares class names rather than class objects | Comparação incorreta de objetos de classes. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals method overrides equals in superclass and may not be symmetric | Possível erro na utilização da sobrecarga de métodos *equals()* | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals() method defined that doesn't override Object.equals(Object) | Quando o método *equals*() é utilizado ele deve sobrecarregar o método *equals*(*Object*) | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals() used to compare array and nonarray | Comparação de objetos com tipos diferentes. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - equals(...) used to compare incompatible arrays | Comparação de *arrays* incompatíveis. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Field only ever set to null | Campo que sempre tem o valor nulo. Deve-se verificar erros de programação. | Indicativo de erros de programação | Baixa | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - File.separator used for regular expression | Indicativo de má utilização de expressões regulares | Indicativo de erros de programação | Baixa | Corretude funcional | Adequação funcional |
| Correctness - Format string placeholder incompatible with passed argument | Erro na formatação dos resultados para exibição. | Bug | Média | Corretude funcional | Adequação funcional |
| Correctness - Format string references missing argument | Erro na formatação dos resultados para exibição. | Bug | Média | Corretude funcional | Adequação funcional |
| Correctness - hasNext method invokes next | Possível erro na utilização do método hasNext. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Illegal format string | Formato ilegal de String que leva à erro em tempo de execução. | Bug | Média | Corretude funcional | Adequação funcional |
| Correctness - Impossible cast | Cast impossível que pode levar a comportamentos inesperados. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Impossible downcast | Cast que sempre vai lançar a exceção ClassCastException. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Impossible downcast of toArray() result | Cast que sempre vai lançar a exceção ClassCastException. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Incompatible bit masks (BIT_AND) | Máscara de bits incompatível que pode levar a comportamentos inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Incompatible bit masks (BIT_IOR) | Máscara de bits incompatível que pode levar a comportamentos inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - instanceof will always return false | Deve-se investigar verificações instanceof() que vão sempre retornar falso. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - int value cast to double and then passed to Math.ceil | Possível passagem de parâmetros incorreta. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - int value cast to float and then passed to Math.round | Possível passagem de parâmetros incorreta. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Integer multiply of result of integer remainder | Possível confusão na precedência de operadores. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Integer remainder modulo 1 | Possível confusão na utilização da operação módulo. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Integer shift by an amount not in the range 0..31 | Operação bitwise que pode causar resultados indesejáveis. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Invalid syntax for regular expression | Deve-se verificar possíveis erros no uso de expressões regulares. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Invocation of equals() on an array, which is equivalent to == | Compara a referência ao invés do valor. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Method assigns boolean literal in boolean expression | Indicativo de erro de programação que faz atribuição ao invés de comparação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Method attempts to access a prepared statement parameter with index 0 | Passagem de parâmetro incorreta que pode levar a resultados indesejáveis. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Method attempts to access a result set field with index 0 | Passagem de parâmetro incorreta que pode levar a resultados indesejáveis. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Method call passes null for nonnull parameter | Passagem de parâmetros nulos pode levar a resultados indesejáveis. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Method defines a variable that obscures a field | Variáveis com o mesmo nome de campos afeta a legibilidade do código. | Inconsistências de estilo | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Method does not check for null argument | Método não faz checagem de parâmetros nulos, o que pode levar a resultados indesejáveis. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Method doesn't override method in superclass due to wrong package for parameter | Possível erro na declaração dos tipos dos parâmetros | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Method ignores return value | Ignora o valor retornado pelo método, o que pode levar a resultados indesejáveis. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Method may return null, but is declared @NonNull | Métodos com tipos incompatíveis que podem levar a resultados indesejáveis. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Method must be private in order for serialization to work | Se o método não for privado ele será ignorado causando resultados indesejados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Method performs math using floating point precision | Deve-se considerar o uso de Double para não perder precisão. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - More arguments are passed that are actually used in the format string | Parâmetros desnecessários que devem ser excluídos. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - No previous argument for format string | Falta argumento para formatar a string, o que pode levar à *MissingFormatArgumentException* | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - No relationship between generic parameter and method argument | Parâmetros com tipos incompatíveis, o que pode levar a resultados indesejados. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Nonsensical self computation involving a field (e.g., x & x) | Operação desnecessária que pode indicar um erro de programação. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Non-virtual method call passes null for nonnull parameter | Passagem de parâmetro nulo para parâmetro anotado como não-nulo. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Null pointer dereference | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Null pointer dereference in method on exception path | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Null value is guaranteed to be dereferenced | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Nullcheck of value previously dereferenced | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Number of format-string arguments does not correspond to number of placeholders | Utilização incorreta de valores em Strings, pode levar a resultados inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Overwritten increment | Sobrescrita de valores que já foram atribuídos, pode indicar um possível erro de programação e levar a resultados inesperados. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Possible null pointer dereference | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Possible null pointer dereference in method on exception path | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Primitive array passed to function expecting a variable number of object arguments | Parâmetros com tipos incompatíveis, o que pode levar a resultados indesejados. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Primitive value is unboxed and coerced for ternary operator | Conversão de tipos incompatível que pode causar resultados indesejados | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Correctness - Random value from 0 to 1 is coerced to the integer 0 | Geração incorreta de numerous aleatórios | Bug | Alta | Corretude funcional | Adequação funcional |
| Correctness - Read of unwritten field | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Repeated conditional tests | Testes condicionais repetidos que devem ser refatorados | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Self assignment of field | Atribuição de valores de um campo para ele mesmo, o que indica erros de programação. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Self comparison of field with itself | Comparação de um campo com ele mesmo, o que indica erros de programação. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Self comparison of value with itself | Comparação de um valor com ele mesmo, o que indica erros de programação. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Signature declares use of unhashable class in hashed construct | Assinatura de classe inconsistente, o que indica erro de programação e pode levar a comportamento inesperado. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - The readResolve method must not be declared as a static method. | Para que o método seja reconhecido, ele deve ser declarado como estático. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Uncallable method defined in anonymous class | Métodos que não são utilizados e devem ser excluídos | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Uninitialized read of field in constructor | Leitura de um campo que ainda não foi inicializado, o que pode levar a comportamento inesperado. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Uninitialized read of field method called from constructor of superclass | Leitura de um campo que ainda não foi inicializado, o que pode levar a comportamento inesperado. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Unnecessary type check done using instanceof operator | Checagem de tipo desnecessária que deve ser refatorada | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Unneeded use of currentThread() call, to call interrupted() | É recomendado utilizaro o método Thread.interrupted(). | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Correctness - Unwritten field | Sempre retornará o valor padrão, o que indica erro de programação e pode levar a comportamento indesejado. | Indicativo de erros de programação | Alta | Corretude funcional | Adequação funcional |
| Correctness - Useless assignment in return statement | Valor que não é utilizado e deve ser removido. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Useless control flow to next line | Valor que não é utilizado e deve ser removido. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Correctness - Using pointer equality to compare different types | Comparação de tipos incompatíveis, o que pode levar a resultados indesejáveis. | Bug em potencial | Alta | Corretude funcional | Adequação funcional |
| Correctness - Value annotated as carrying a type qualifier used where a value that must not carry that qualifier is required | Utilização incorreta de modificadores. | Indicativo de erros de programação | Média | Corretude funcional | Adequação funcional |
| Correctness - Value is null and guaranteed to be dereferenced on exception path | Irá causa NullPointerException quando o código for executado. | Bug | Alta | Corretude funcional | Adequação funcional |
| Dataflow Anomaly Analysis | Verifica o uso anormal de variáveis que pode levar a erros | Indicativo de erros de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList" | Boa prática de programação que recomenda o uso de interfaces para declaração de coleções | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Design For Extension | Classes devem ser projetadas para a herança. | Inconsistência de estilo | Baixa | Modularidade | Manutenabilidade |
| Dont Use Float Type For Loop Indice | Float pode não ter a precisão desejada. | Futuro erro de programação | Média | Corretude funcional | Adequação funcional |
| Empty arrays and collections should be returned instead of null | Boa prática que força a checagem do retorno. | Inconsistência de estilo | Baixa | Utilização de recursos | Eficiência de performance |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Empty statements should be removed | Trechos vazios geralmente são introduzidos por engano e podem causar efeitos indesejáveis. | Indicativo de erros de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Exception handlers should provide some context and preserve the original exception | Deve-se preservar o contexto e a exceção para evitar possíveis confusões durante o *debug*. | Futuro erro de programação | Média | Analisabilidade e Testabilidade | Manutenabilidade |
| Exception types should not be tested using "instanceof" in catch blocks | Boa prática que recomenda o uso de vários blocos *catch* ao invés de capturar diversas exceções. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Exceptions should not be thrown in finally blocks | Pode mascara exceções anteriores e perder o contexto no qual as exceções foram lançadas. | Futuro erro de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Excessive Class Length | Classe muito grande que deve ser refatorada | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Excessive Public Count | Classe muito grande que deve ser refatorada | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Expressions should not be too complex | | Ineficiência | Média | Analisabilidade | Manutenabilidade |
| Field names should comply with a naming convention | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |
| Files should contain an empty new line at the end | Evita conflitos na utilização de *softwares* de controle de versão. | Inconsistência de estilo | Baixa | Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Files should not have too many lines | Arquivos com muitas linhas indicam classes ou métodos muito longas e com muitas responsabilidades que devem ser refatoradas. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| God Class | Classe muito grande e com muitas responsabilidades que deve ser refatorada. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| if/else/for/while/do statements should always use curly braces | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Immutable Field | Verifica a necessidade de se usar o modificador final em determinados campos | Inconsistência de estilo | Baixa | Modificabilidade | Manutenabilidade |
| Import Order | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Increment (++) and decrement (--) operators should not be mixed with other operators in an expression | Afeta a legibilidade e pode levar a comportamentos inesperados. | Futuro erro de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Inefficient String Buffering | Boa prática que recomenda que não se deve concatenar não-literais em StringBuffers | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Insufficient comment density | Deve-se determinar a quantidade aceitável de comentários que colaboram com o entendimento do código. | Inconsistência de estilo | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Insufficient String Buffer Declaration | Pode causar a mudança do tamanho da String várias vezes durante o tempo de execução, o que pode afetar a performance do programa. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Integer Instantiation | Instanciação desnecessária | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Interface names should comply with a naming convention | Convenções de nomeação facilitam a legibilidade do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| IP addresses should not be hardcoded | O IP deve ser capturado dinamicamente para evitar a necessidade de recompilação e a necessidade de inserir o IP em ambiente de desenvolvimento. | Ineficiência | Média | Utilização de recursos | Eficiência de performance |
| Lambdas and anonymous classes should not have too many lines | Deve-se evitar classes anônimas e lambdas com muitas linhas de código, pois elas são usadas geralmente para inserir comportamento sem a necessidade de se criar uma classe específica. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Law Of Demeter | Boa prática de programação que evita o acoplamento entre classes ou objetos. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Left curly braces should be located at the beginning of lines of code | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Lines of code should not be too long | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Literal boolean values should not be used in condition expressions | Afeta a legibilidade do código. | Inconsistência de estilo | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Local variable and method parameter names should comply with a naming convention | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Local variables should not shadow class fields | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Loggers should be "private static final" and should share a naming convention | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Long Variable | Nome muito longo que deve ser refatorado. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Loop counters should not be assigned to from within the loop body | Pode causar comportamento inesperado ao se modificar o contador dentro do loop. | Bug em potencial | Média | Corretude funcional | Adequação funcional |
| Loops should not contain more than a single "break" or "continue" statement | Boa prática de programação que evita *loops* muito complexos. | Ineficiência | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Magic Number | Convenção de codificação que afeta a legibilidade do código. | Inconsistência de estilo | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Malicious code vulnerability - Field is a mutable array | *Array* pode ter seu valor alterado por código malicioso. | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Field is a mutable Hashtable | *Hastable* pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Field isn't final and can't be protected from malicious code | Campo pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Field isn't final but should be | Campo pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Field should be both final and package protected | Campo pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Field should be moved out of an interface and made package protected | Campo pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Field should be package protected | Campo pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Malicious code vulnerability - Finalizer should be protected, not public | Finalizador pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Malicious code vulnerability - Public static method may expose internal representation by returning array | *Array* retornado pode ter seu valor alterado por código malicioso | Bug | Muito Alta | Segurança | Integridade |
| Method names should comply with a naming convention | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Method parameters, caught exceptions and foreach variables should not be reassigned | Não se deve associar valor à parâmetros, pois esses valores serão perdidos, indica um erro de programação. | Indicativo de erros de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Methods should not be empty | Método vazio que deve ser removido. | Indicativo de erros de programação | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Methods should not be too complex | Métodos muito complexos devem ser refatorados. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Methods should not contain too many return statements | Métodos muito complexos devem ser refatorados. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Methods should not have too many lines | Métodos muito complexos e com muitas responsabilidades devem ser refatorados. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Modifiers should be declared in the correct order | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Naming - Avoid field name matching method name | Nome de variáveis iguais ao nome dos métodos devem ser removidos | Inconsistência de estilo | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Nested blocks of code should not be left empty | Blocos vazios que devem ser removidos. | Indicativo de erros de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| NPath complexity | Métodos muito complexos que afetam a manutenabilidade | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Null Assignment | Boa prática que recomenda que não se deve atribuir o valor null à variáveis | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Overriding methods should do more than simply call the same method in the super class | Consome recursos com uma operação ineficiente que deve ser evitada. | Ineficiência | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Package names should comply with a naming convention | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Performance - Could be refactored into a named static inner class | Definição desnecessária de classe interna que provoca a utilização desnecessária de recursos. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Could be refactored into a static inner class | Boa prática que recomenda a criação de classe interna com os modificadores *static ineer* | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Huge string constants is duplicated across multiple class files | Valores duplicados que provocam a utilização desnecessária de recursos. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Inefficient use of keySet iterator instead of entrySet iterator | Boa prática que recomenda o uso de entrySet para evitara utilização desnecessária de recursos. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Maps and sets of URLs can be performance hogs | Deve-se considerar o uso de *java.net.URI* para evitar problemas de perfomance | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Performance - Method allocates a boxed primitive just to call toString | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method allocates an object, only to get the class object | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method calls static Math class method on a constant value | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method concatenates strings using + in a loop | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method invokes inefficient floating-point Number constructor; use static valueOf instead | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method invokes inefficient new String() constructor | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method invokes inefficient new String(String) constructor | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method invokes inefficient Number constructor; use static valueOf instead | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method invokes toString() method on a String | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Method uses toArray() with zero-length array argument | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Performance - Primitive value is boxed and then imMédiately unboxed | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Primitive value is boxed then unboxed to perform primitive coercion | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - The equals and hashCode methods of URL are blocking | Deve-se considerar o uso de *java.net.URI* para evitar problemas de perfomance | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Unread field | Campo que nunca é utilizado deve ser excluído. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Unread field: should this field be static? | Campo que nunca é utilizado deve ser excluído. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Unused field | Campo que nunca é utilizado deve ser excluído. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Performance - Use the nextInt method of Random rather than nextDouble to generate a random integer | Operação desnecessária que utiliza recursos que poderiam ser preservados. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| Public methods should throw at most one checked exception | Boa prática que recomenda que métodos públicos devem lançar exceções que foram checadas anteriormente. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Public types, methods and fields (API) should be documented with Javadoc | Boa prática que recomenda o uso do padrão Javadoc para documentação. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Redundant Field Initializer | Inicializador redundante que deve ser refatorado. | Ineficiência | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Redundant Modifier | Modificador redundante que deve ser refatorado. | Ineficiência | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Right curly brace and next "else", "catch" and "finally" keywords should be located on two different lines | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Security - A prepared statement is generated from a nonconstant String | Código SQL que se não for checado pode causar danos à aplicação. | Bug | Muito Alta | Integridade | Segurança |
| Security - Array is stored directly | Boa prática que previne a modificação indesejada de valores de *arrays*. | Bug | Muito Alta | Integridade | Segurança |
| Security - Empty database password | Indica que o banco de dados não está protegido. | Bug | Muito Alta | Integridade | Segurança |
| Security - Hardcoded constant database password | Senha do banco de dados não deve estar disponível no código fonte. | Bug | Muito Alta | Integridade | Segurança |
| Security - HTTP cookie formed from untrusted input | Formação de *cookie* que se não for checada pode causar danos à aplicação. | Bug | Muito Alta | Integridade | Segurança |
| Security - HTTP Response splitting vulnerability | Formação de resposta HTTP que se for exposta pode levar à exposição indesejada de informações. | Bug | Muito Alta | Integridade | Segurança |
| Security - JSP reflected cross site scripting vulnerability | Escreve parâmetros HTTP diretamente em uma saída JSP, o que pode levar à exposição indesejada de informações. | Bug | Muito Alta | Integridade | Segurança |
| Security - Method returns internal array | Permite que usuários modifiquem diretamente código que pode ser crítico | Bug | Alta | Integridade | Segurança |
| Security - Nonconstant string passed to execute method on an SQL statement | Pode executar código SQL malicioso. | Bug | Alta | Integridade | Segurança |
| Security - Servlet reflected cross site scripting vulnerability | Escreve parâmetros HTTP diretamente em uma página de erros, o que pode levar à | Bug | Alta | Integridade | Segurança |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| | exposição indesejada de informações. | | | | |
| Short Class Name | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Short Variable | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Singular Field | Campo usado por um único método pode ser substituído por uma variável local | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| Source code should be correctly indented | Convenções de codificação que facilitam o entendimento do código. | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Static Variable Name | Campo usado por um único método pode ser substituído por uma variável local | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| Strict Duplicate Code | Afeta a manutenção do código e indica alto acoplamento. | Indicativo de erros de programação | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| String literals should not be duplicated | DIficulta a refatoração, strings duplicadas podem ser substituídas por uma constante. | Ineficiência | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| String To String | Conversão desnecessária | Ineficiência | Baixa | Utilização de recursos | Eficiência de performance |
| String.valueOf() should not be appended to a String | Afeta a legibilidade do código. | Inconsistência de estilo | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Strings literals should be placed on the left side when checking for equality | Previne que a exceção *NullPointerException* seja lançada. | Bug em potencial | Alta | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Switch cases should not have too many lines | Blocos switch muito complexos que devem ser refatorados. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Switch statements should end with a default case | Boa prática de programação defensive. | Inconsistência de estilo | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used | Pode impactar negativamente na performance. | Ineficiência | Muito Alta | Utilização de recursos | Eficiência de performance |
| System.out and System.err should not be used as loggers | Deve-se usar *loggers* específicos. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| The default unnamed package should not be used | Convenções de nomeação facilitam o entendimento do código. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| The members of an interface declaration or class should appear in a pre-defined order | Convenções de codificação que facilitam o entendimento do código | Inconsistência de estilo | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Throwable.printStackTrace(...) should never be called | Deve-se usar o *logger* adequado. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Throws declarations should not be redundant | Declaração redundate que deve ser removida. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Too Many Fields | Clases com muitos campos que deve ser refatorada | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Too many methods | Classe muito grande que deve ser refatorada | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Try-catch blocks should not be nested | Afeta a legibilidade do código. | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unnecessary constructor | Construtor desnecessário que deve ser removido. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| Unnecessary Local Before Return | Return desnecessário que deve ser removido. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused formal parameter | Parâmetro desnecessário que deve ser removido. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused local variables should be removed | Variável desnecessária que deve ser removida. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused Modifier | Modificador desnecessário que deve ser removido | Ineficiência | Muito baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused Null Check In Equals | Checagem desnecessária que deve ser removida | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused private fields should be removed | Variável desnecessária que deve ser removida. | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused private method | Método desnecessário que deve ser removido | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused protected method | Método desnecessário que deve ser removido | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Use ConcurrentHashMap | Boa prática que recomenda o uso de uma interface específica para Map quando se está desenvolvendo aplicações concorrentes | Inconsistência de estilo | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Use Locale With Case Conversions | Boa prática que recomenda o uso de Locale para conversões de caixa alta e caixa baixa | Bug em potencial | Alta | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Use String Buffer For String Appends | Verifica o uso de += para fazer a concatenação de strings | Ineficiência | Média | Modificabilidade e Analisabilidade | Manutenabilidade |
| Useless imports should be removed | *Imports* desnecessários que devem ser removidos | Ineficiência | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Useless parentheses around expressions should be removed to | Parenteses desnecessários que devem ser removidos | Ineficiência | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |

| Regras | Impacto | Categoria | Severidade | Sub-característica associada | Característica associada |
|---|---|---|---|---|---|
| prevent any misunderstanding | | | | | |
| Utility classes should not have a public constructor | *Utility* classes não devem ser instanciadas, por isso não podem ter construtores públicos. | Indicativo de erros de programação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Variables should not be declared and then immediately returned or thrown | Variável desnecessária que deve ser removida. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |

ANEXO II: Perfil de Qualidade PHP para Órgãos da APF

| Regras | Impacto | Categoria | Severidade | Sub-caracteristica associada | Caracteristica |
|---|---|---|---|---|---|
| Class names should comply with a naming convention | Convenções de codificação que facilitam o entendimento do código | Inconsistência de estilo | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Comments should not be located at the end of lines of code | Pode afetar a legibilidade do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Function names should comply with a naming convention | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Functions should not have too many lines | Uma função que é muito longa tende a agregar muitas responsabilidades. Tal função é inevitavelmente mais difícil de entender e de manter. | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| "eval" function should not be used | A função *eval* avalia o parâmetro passado como código PHP. Porém, esse parâmetro pode ser malicioso ou não validado. | Bug em potencial | Alta | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| "switch/case" clauses should not have too many lines | Afeta a legibilidade do fluxo de alternativas do switch | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| Classes should not be too complex | Na maioria das vezes uma classe muito complexa quebra o princípio da responsabilidade única. | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| Empty statements should be removed | Trechos vazios geralmente são introduzidos por engano e podem causar efeitos indesejáveis. | Indicativo de erros de programação | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Local variables should not have the same name as class fields | Afeta a legibilidade do código. | Convenção de codificação | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |

| | | | | | |
|---|---|---|---|---|---|
| Copyright and license headers should be defined in all source files | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Switch cases should end with an unconditional break statement | Executa os dois casos seguidos em um switch, o que pode levar a comportamentos inesperados. | Bug em potencial | Alta | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Sections of code should not be commented out | Afeta a legibilidade do código e o comentário ao ser removido pode levar a comportamentos inesperados. | Futuro erro de programação | Media | Analisabilidade | Manutenabilidade |
| A loop's counter should not be assigned within the loop body | Pode-se obter comportamento inesperado quando o contador é modificado no no corpo do método | Ineficiência | Media | Utilizacao de recursos | Performance Efficiency |
| if ... else if" constructs shall be terminated with an "else" clause | Programação defensiva que adiciona uma instrução else ao final para tomar a ação apropriada caso as instruções anteriores falhem. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Increment (++) and decrement (--) operators should not be mixed with other operators in an expression | Afeta a legibilidade e pode levar a comportamentos inesperados. | Futuro erro de programação | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| A "while" loop should be used instead of a "for" loop | Deve-se substituir instruções for por while quando existe apenas uma condição para o fluxo de controle. | Ineficiência | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Variable variables should not be used | Afeta a manutenção do código pois a variável com o modificador variável tem o seu valor modificado em tempo de execução. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| "switch" statements should end with a "case default" clause | Programação defensiva que adiciona uma instrução case deafult ao final para tomar a ação apropriada caso as instruções anteriores falhem. | Bug em potencial | Alta | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |

| | | | | | |
|---|---|---|---|---|---|
| Literal boolean values should not be used in condition expressions | Afeta a legibilidade do código. | Convenção de codificação | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply | Afeta a legibilidade do código e indicam implementações do tipo "código espaguete" | Futuro erro de programação | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| String literals should not be duplicated | DIficulta a refatoração, strings duplicadas podem ser substituídas por uma constante. | Ineficiência | Média | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Functions should not have too many parameters | Indica que uma nova estrutura deve ser criada para conter todos os parâmetros ou que a função tem muitas responsabilidades. | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| Source files should not have any duplicated blocks | Afeta a manutenção do código e indica alto acoplamento. | Indicativo de erros de programação | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Source files should have a sufficient density of comment lines | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Lines should not end with trailing whitespaces | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| "if/else/for/while/do" statements should always use curly braces | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| An open curly brace should be located at the end of a line | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Lines should not be too long | Convenções de codificação que facilitam o entendimento do código | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Field names should comply with a naming convention | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Functions should not be too complex | Funções muito complexas que devem ser refatoradas. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |

| | | | | | |
|---|---|---|---|---|---|
| Unused local variables should be removed | Variável desnecessária que deve ser removida. | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| Local variable and function parameter names should comply with a naming convention | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Unused function parameters should be removed | Parâmetros desnecessários que deve ser removidos. | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| Collapsible "if" statements should be merged | Convenções de codificação que facilitam o entendimento do código | Ineficiência | Media | Analisabilidade | Manutenabilidade |
| PHP 4 constructor declarations should not be used | Deve-se usar as convenções definidas pela nova versão da linguagem PHP. | Ineficiência | Media | Analisabilidade | Manutenabilidade |
| Classes should not have too many methods | Classes muito complexas que devem ser refatoradas. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Nested blocks of code should not be left empty | Indicativo de erros de programação | Indicativo de erros de programação | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Files should not have too many lines | Arquivos muito complexas que devem ser refatoradas. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Expressions should not be too complex | Expressões muito complexas que devem ser refatoradas. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Tabulation characters should not be used | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Files should contain an empty new line at the end | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Analisabilidade | Manutenabilidade |
| Local variables should not be declared and then immediately returned or thrown | Variável desnecessária que deve ser removida. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Statements should be on separate lines | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Analisabilidade | Manutenabilidade |

| | | | | | |
|---|---|---|---|---|---|
| Sections of code should not be "commented out" | Afeta a legibilidade do código e o comentário ao ser removido pode levar a comportamentos inesperados. | Futuro erro de programação | Media | Analisabilidade | Manutenabilidade |
| Functions should not contain too many return statements | Funções muito complexas que devem ser refatoradas. | Ineficiência | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| Deprecated predefined variables should not be used | Variáveis predefinidas que não são mantidas e que não devem ser utilizadas. | Convenção de codificação | Media | Analisabilidade | Manutenabilidade |
| Constant names should comply with a naming convention | Convenções de codificação que facilitam o entendimento do código | Convenção de codificação | Baixa | Analisabilidade | Manutenabilidade |
| Unused private fields should be removed | Campos que não são utilizados e devem ser removidos. | Ineficiência | Media | Modificabilidade e Analisabilidade | Manutenabilidade |
| Classes should not be coupled to too many other classes (Single Responsibility Principle) | Classes muito acopladas que devem ser refatoradas. | Futuro erro de programação | Media | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| "if" statement conditions should not always evaluate to "true" or to "false" | Pode indicar erro de programação e faz com que o bloco de código não seja totalmente utilizado. | Ineficiência | Alta | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |
| "switch" statements should have at least 3 "case" clauses | Quando não existem três ou mais casos para a instrução switch deve-se substituir por instruções if. | Ineficiência | Baixa | Modificabilidade e Analisabilidade | Manutenabilidade |
| Return of boolean expressions should not be wrapped into an "if-then-else" statement | Expressões muito complexas que devem ser refatoradas. | Futuro erro de programação | Baixa | Modificabilidade, Analisabilidade e Testabilidade | Manutenabilidade |