Universidade Federal de Pernambuco

Graduação em Ciência da Computação
Centro de Informática
2014.1

QA Soft: Uma Ferramenta de Análise de Software Baseada em Indicadores de Qualidade

Trabalho de Graduação

Aluno - Leonardo da Silva Leandro (lsl2@cin.ufpe.br)
Orientador - Carlos André Guimarães Ferraz (cagf@cin.ufpe.br)

Recife, Agosto de 2014.

Agradecimentos

Em primeiro lugar, gostaria de agradecer a Deus por me dar a grande oportunidade de viver e aprender tantas coisas que me tornaram um ser humano melhor e mais capaz.

Gostaria de agradecer aos meus familiares, meus pais, Niedja e Ivonilso, pela educação que me deram, sempre me incentivando nos estudos e me apoiando nas horas difíceis. Em especial, gostaria de agradecer a minha mãe, por sempre ter sido o meu grande exemplo de pessoa e de profissional. Mãe, a senhora não precisava falar nada, bastava apenas ficar lhe observando viver, e eu já tinha um curso completo de como me comportar no meu dia-a-dia.

Agradeço a Nayalla, minha esposa, pelo companheirismo e pela paciência com meus horários tão inconstantes, bem como com os momentos de desespero durante os vários projetos e as viradas de noite fora de casa. Graças a esse grande apoio e graças à sua presença em minha vida, tive mais um motivo pra vencer essa etapa.

Agradeço aos meus orientadores, Carlos Ferraz e Thiago Prota, por aceitarem guiar-me nesse trabalho e pela paciência comigo durante as etapas preparatórias. Vocês sempre foram solícitos e atenciosos.

Agradeço ao Centro de Informática pela infraestrutura oferecida desde o começo do curso, me proporcionando condições de aprender e exercitar as técnicas aprendidas através de recursos de qualidade.

Agradeço aos meus amigos, principalmente aos guerreiros que comigo conviveram durante todos esses anos de curso, vivenciando momentos de alegria e de desespero, virando noites para terminar projetos, ou mesmo para estudar para provas e sempre me ajudando quando preciso. Vocês têm grande participação nessa conquista.

Obrigado!

Resumo

As disciplinas de engenharia alinham atividades de projeto e construção com atividades que verificam produtos intermediários e finais de forma que os defeitos possam ser identificados e removidos. A Engenharia de Software não é exceção: a construção de software de alta qualidade requer a combinação de atividades de projeto e verificação ao longo do desenvolvimento.

Contudo, existem detalhes importantes que devem compor um software de alta qualidade e que muitas vezes são esquecidas, ou mesmo mal avaliadas pelos desenvolvedores no decorrer da implementação. O desempenho é uma delas. Muitas vezes a preocupação com as entregas contínuas, ou com o prazo estabelecido com o cliente, toma conta do processo de desenvolvimento ao ponto de se deixar de lado a avaliação do desempenho do software.

Qualidade é um ramo de estudo que se subdivide em diversos grupos. Durante a concepção de uma aplicação, métricas como robustez, eficácia e operabilidade parecem ter um peso maior. Existem diversas ferramentas CASE que avaliam um software, por meio de testes, ou por meio de análise de resultados, levando em consideração diferentes tipos de critérios de qualidade: cobertura, robustez, recuperação, eficiência.

A proposta deste trabalho foi apresentar um protótipo de uma ferramenta de análise de software, capaz de avaliar um determinado software usado como alvo, tendo como foco da análise (para esta primeira versão) o desempenho.

Ao final, a ferramenta foi capaz de fornecer visões para diferentes stakeholders (engenheiros de softwares, CMs, clientes finais) da ferramenta analisada com um resumo da avaliação feita, e um quadro da evolução ou da regressão do desempenho do software.

Palavras-chave: qualidade de software, análise de resultados, testes, desempenho, eficiência.

Abstract

Engineering disciplines align project and construction activities with actions that verify intermediate and final products. By doing this, any defect can be identified and removed. Software Engineering is not an exception: high quality software construction requires a combination of project activities and verification throughout the development.

However, there are important details that should compose high-quality software that are often forgotten or are not well evaluated by developers during the implementation. Performance is one of these important parts. In many cases, continuous concern with project delivery or due date agreed with the client overcomes the software development and neglects its performance evaluation.

Quality is an area of study which is subdivided in many groups. During the beginning of an application, metrics such as robustness, efficiency and operability appear to be more influential. There are diverse CASE tools that evaluate a software, either by testing or analysing of results, considering different types of quality criteria: coverage, robustness, recovery efficiency.

The aim of this project was to present a prototype of a software analysis tool able to assess any specific software focusing on the performance analysis (in this first version).

In the end, this tool could provide insights for different stakeholders (software engineers, CMs, final clients) of the analysed tool, with a summary of the assessment, and a picture of the evolution or regression of the software performance.

Key-words: Software quality, analysis of results, tests, performance, efficiency

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

Este capítulo apresenta os aspectos introdutórios deste trabalho e está organizado em três seções. Na primeira, apresenta-se a motivação deste trabalho, definindo o problema atacado. Na segunda seção, são descritos os objetivos. Em seguida, na terceira e última seção, menciona-se a estrutura do documento.

1.1 Definição do Problema

Desempenho é um ponto crucial na qualidade de um software. Sistemas complexos podem requisitar grande quantidade de memória e processamento, dependendo de como foram concebidos e do tipo de trabalho realizado.

Quando se fabrica software para um cliente, são definidos muitos requisitos durante as reuniões de alinhamento e de planejamento. Requisitos não-funcionais, como desempenho, normalmente são explicitamente especificados somente quando são uma característica esperada. Contudo, mesmo quando o desempenho não é especificado, é esperado que o software pudesse fornecer resultados em um tempo razoável [9].

Essa é uma questão bastante complicada, principalmente quando o requisito de desempenho não é explicitamente mencionado. Em sua maioria, os sistemas que são feitos sem foco em *performance* acabam apresentando defasagem nos testes de estresse e nos testes de uso de recursos. Aparentemente não existe um cuidado com o tipo de código produzido, assim como com os algoritmos e as estruturas de dados usados.

A problemática relacionada com qualidade de software também pode ser descrita da seguinte forma: "*Qualidade, de maneira simplista, significa que um produto deve atender às suas especificações... Isso é problemático para os sistemas de software... Existe uma tensão entre os requisitos de qualidade do cliente (eficiência, confiabilidade, etc.) e requisitos de qualidade do desenvolvedor (facilidade de manutenção, reusabilidade, etc.)... As especificações de software são, geralmente, incompletas e frequentemente inconsistentes*" [12].

1.2 Motivação

Ainda que o desenvolvimento de software deva ser alinhado com as questões de qualidade, percebe-se que muitas vezes se faz necessário o uso de ferramentas CASE para auxiliar no desenvolvimento, avaliando certos pontos, como cobertura de requisitos, robustez e também desempenho.

Existem diversas ferramentas no mercado, ou mesmo *open source* que auxiliam na identificação de pontos de melhoria de qualidade em softwares. Contudo, a maioria delas é de análise estática de código, o que de fato é muito importante, mas que se percebe não ser tão usual no cotidiano dos engenheiros. Um exemplo de ferramenta de análise estática de código é FxCop, que analisa um assembly baseado nas regras definidas pelas diretrizes de design para o desenvolvimento de bibliotecas de classes [3].

Existem muitos outros exemplos, uns voltados para a automatização de testes, outros com a mesma visão do FxCop.

Sendo assim, percebe-se a importância da criação de uma ferramenta de análise de software, através da qual a avaliação seja feita de acordo com o produto gerado pelo software analisado, e não o código propriamente dito. Isso pode ser visto como um último recurso do desenvolvedor, antes de bater o martelo e considerar o software pronto para entrega.

1.3 Objetivos

Este trabalho tem como objetivo principal a implementação de uma ferramenta de análise de qualidade de software, através da qual seja possível avaliar um software continuamente, visando diferentes pontos de qualidade.

Para esta primeira versão pretende-se codificar um protótipo que permita analisar um item de qualidade, a saber, o desempenho. Contudo a ideia é que outros pontos de análise de qualidade de software possam ser abordados no futuro, tornando a ferramenta uma boa aliada aos desenvolvedores de software.

1.4 Estrutura

Esta monografia está estruturada em mais quatro capítulos, além deste introdutório. O capítulo 2 apresenta a fundamentação teórica que embasa a realização deste trabalho e que tornará melhor o entendimento do mesmo. Sendo assim, uma visão geral sobre qualidade de software será feita, mais especificamente em performance e indicadores de desempenho.

O capítulo 3 é reservado para a descrição da ferramenta proposta. Para tanto, inicialmente é feita uma breve apresentação do software usado como "alvo" da ferramenta, para que se tenha uma ideia superficial do tipo de informação trabalhada pelo software e do que necessariamente será avaliado pelo analisador desenvolvido. Nesta seção também serão descritos os requisitos elicitados e será mostrado qual o indicador de qualidade foi selecionado para o protótipo proposto e como o indicador foi adquirido, a partir das informações coletadas do software analisado.

O capítulo 4 apresenta as etapas de desenvolvimento da ferramenta. Primeiramente, fala-se da estrutura de dados que compõe o *core* do analisador e como os dados são distribuídos nessa estrutura. Após isso, começa-se tratando da modelagem UML dos objetos, de forma a deixar claro como foi pensada a arquitetura da ferramenta, para esta primeira versão. Em seguida, será descrito o Template do arquivo de entrada da análise. Então terá inicio a apresentação das etapas de codificação. Finalizando este capítulo, serão apresentados os resultados da análise.

No capítulo 5, serão apresentadas as conclusões do trabalho. Será descrito as vantagens do uso da ferramenta, as limitações da mesma, bem como as dificuldades encontradas no processo de concepção do software e as perspectivas futuras.

2. Qualidade de Software

Antes de apresentar os detalhes da ferramenta, que é o produto deste trabalho, é importante reservar um espaço para escrever sobre a fundamentação teórica, conceituando qualidade de software e falando sobre desempenho.

Uma definição interessante sobre qualidade de produtos de software é dada como segue: "*Pode-se definir qualidade de produto de software como a conformidade a requisitos funcionais e de desempenho declarados explicitamente, padrões de desempenho claramente documentados e as características implícitas que são esperadas de todo software desenvolvido profissionalmente. Por necessidades explicitas, podem-se entender requisitos do usuário; necessidades implícitas relacionam-se, por exemplo, com a performance de execução do sistema...*". [6]

2.1 ABNT e ISO

Fundada em 1940, a Associação Brasileira de Normas e Técnicas é o órgão responsável pela normalização técnica no país, fornecendo a base necessária ao desenvolvimento tecnológico brasileiro [1].

A ABNT é a representante oficial no Brasil da ISO (International Organization for Standardization), uma entidade não governamental com o objetivo de promover o desenvolvimento da normalização e de atividades que facilitem a troca de bens e de serviços no mundo [1].

No que diz respeito a software, a ISO/IEC 9126 é responsável pelas definições de qualidade. A norma define seis características essenciais para a qualidade de produto: funcionalidade, usabilidade, confiabilidade, eficiência, manutenibilidade e portabilidade, porém não define métodos para medição dessas características [14]. Na Figura 1 é possível verificar que essas seis características se dividem em sub-características.

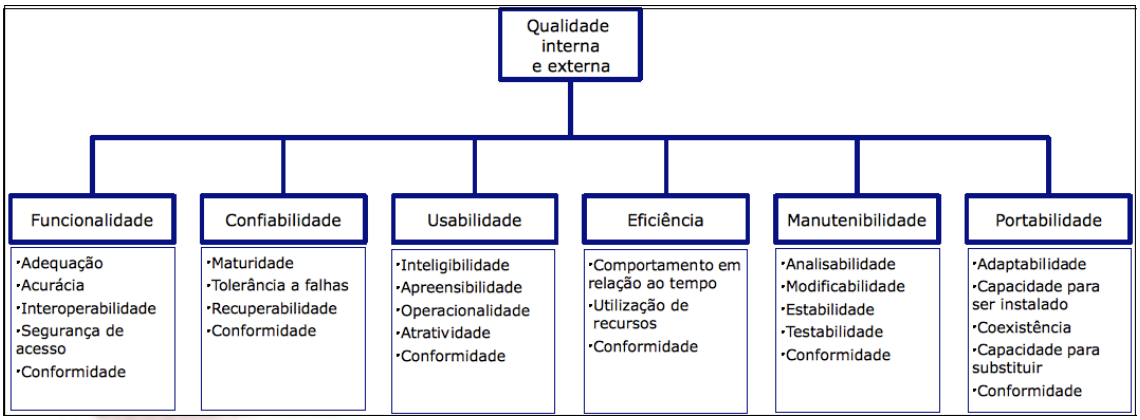

Figura 1 - ISSO/IEC 9126 – Qualidade de Software

2.2 Desempenho e Eficiência de Software

São informações correlacionadas. A eficiência relaciona o nível de desempenho do software e a quantidade de recursos utilizados sob condições pré-estabelecidas [2].

Sub-características:

- Comportamento em relação ao tempo: capacidade de prover informações em período de tempo adequado;
- Comportamento em relação aos recursos: capacidade de usar os recursos adequadamente;
- Conformidade: Capacidade do produto de software de estar adequado a padrões, normas ou convenções relativos à eficiência.

2.3 Indicadores e Testes de Desempenho

Com relação aos testes de performance, são testes realizados para se verificar o tempo de resposta de uma aplicação, determinando assim sua escalabilidade e confiança, de acordo com uma carga [9].

Outra definição para os testes de performance, a qual se adequa muito melhor a este trabalho, é que são testes usados também para se identificar os famosos gargalos de um sistema, determinar *compliance* com os requisitos não-funcionais de performance e coletar outras informações como hardware necessário para a operação da aplicação [13].

Estes testes são vistos de formas diferentes dependendo dos objetivos estabelecidos para os indicadores de performance.

Os indicadores de performance representam as características do sistema que estão diretamente ligadas ao desempenho do software. Como exemplo, pode-se citar o *throughput*, que representa a quantidade de atividades realizadas, vinculadas a uma quantidade de tempo. Quando referido às consultas em banco de dados, quanto maior o *throughput*, mais rápido e mais requisições podem ser realizadas em um intervalo de tempo reduzido.

Para os fins desse trabalho, a performance será avaliada pelo atributo *elapse time*, o qual representa o tempo total que o software em análise gasta para avaliar uma entrada, levando em consideração as questões de hardware da máquina (memória, processador, etc).

3. A Ferramenta

Neste capítulo detalha-se todo o processo de desenvolvimento do QA Soft. Primeiramente, será apresentada uma breve descrição do software a ser avaliado, mostrando algumas características e falando sobre os gargalos do mesmo, os quais deram origem às entradas do analisador e motivaram a criação dos indicadores de desempenho.

Em seguida, dar-se início ao detalhamento da ferramenta de análise, falando sobre os requisitos funcionais e não funcionais elicitados e finalizando com a escolha do indicador de qualidade e do limiar de aceitabilidade.

3.1 Software Alvo

O software analisado pela ferramenta QA Soft proposta nesse trabalho, consiste em uma aplicação complexa, construída na linguagem Java. Recebe como entrada aplicações para TV Digital, implementadas usando as linguagens NCL e LUA, e gera uma saída contendo informações sobre essas aplicações, como, por exemplo, o tempo em que o software demorou em avaliar cada uma dessas aplicações, a quantidade de linhas de código de cada aplicação, entre outras coisas.

Dentre as informações contidas nos arquivos de saída do software alvo (por questões de políticas de não divulgação de informações, ele será referenciado neste trabalho como software ou ferramenta **X**) o mais importante para a análise é o tempo de avaliação de cada aplicação NCL/LUA, denominado no arquivo de saída de *elapse time*.

3.2 Metodologia

A implementação da ferramenta QA Soft foi realizada seguindo as boas práticas de programação e de engenharia de software. Até a sua conclusão, o software passou pelas diversas etapas do ciclo de vida de um software, como mostrado na Figura 2.

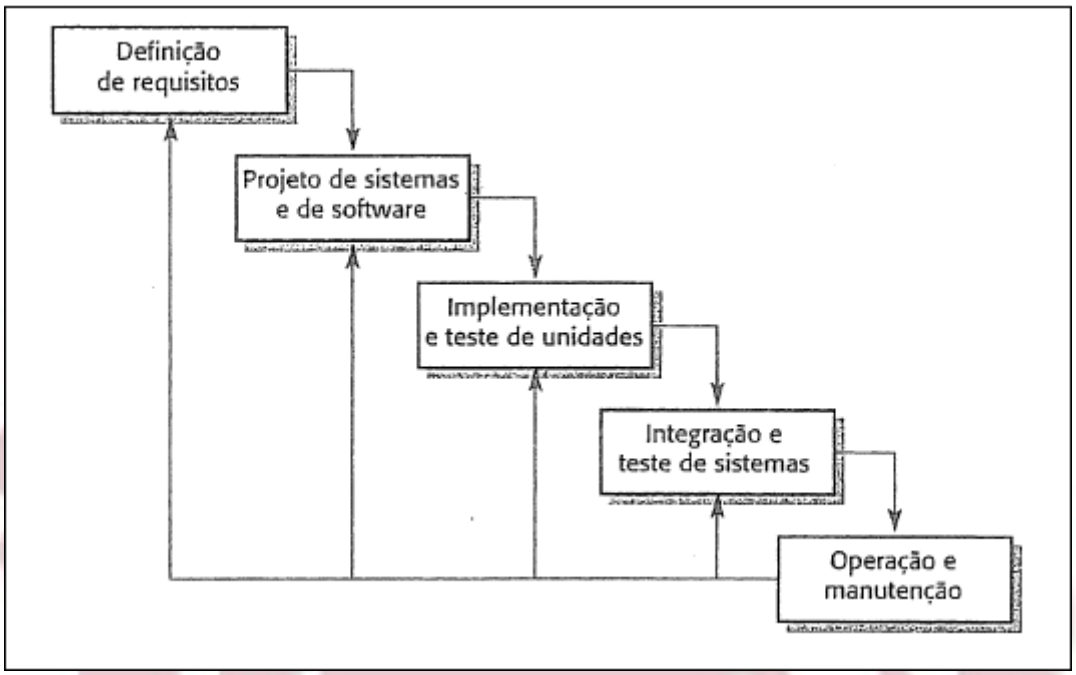

Figura 2 - Ciclo de vida de um software no modelo Cascata [12]

O desenvolvimento seguiu o modelo Cascata, contudo de forma customizada, visto que na prática são muito difíceis os estágios do ciclo não se sobreporem. Pelo fato de o software não necessitar de uma validação continua com cliente e também de não acontecerem mudanças contínuas nos requisitos, foi possível usar este modelo sem maiores atrasos e retrabalhos. Os processos de desenvolvimento serão detalhados nos capítulos subsequentes.

3.3 Requisitos de Software

Requisito é uma capacidade que o software que o usuário precisa a fim de resolver um problema e atingir seu objetivo. Uma capacidade de software que deve ser atendida ou possuída por um sistema ou componentes do sistema para satisfazer um contrato, padrão, especificação, ou outros documentos formalmente impostos [7].

Os requisitos podem ser funcionais e não funcionais.

3.3.1 Requisitos Funcionais

Os requisitos *funcionais* expressam o comportamento do software. Normalmente são orientados a ação (“Quando o usuário faz x, o sistema fará y”). Quando se está definindo requisitos funcionais, deve-se procurar encontrar um equilíbrio entre o quão específico e o quão genérico ou ambíguo deseja-se que o requisito seja [7].

Durante o processo de Definição de Requisitos da ferramenta QA Soft, foi construído um diagrama de casos de uso, de forma a poder se ter uma visão das funcionalidades oferecidas pelo sistema em alto nível, bem como quais os atores que interagem com o mesmo. O modelo apresentado na Figura 3 mostra o diagrama de casos de uso com os únicos dois atores participantes: o usuário e o próprio sistema.

O sistema é bem prático e objetivo, oferecendo apenas dois serviços ao usuário. Um deles, o principal, é o de *Realizar Análise*, o qual o sistema recebe a entrada escolhida pelo usuário (gerada pelo software X) e realiza toda a rotina de análise dos dados e geração de um report para visualização. O outro serviço oferecido para o usuário é o de *Atualizar Limiar*, que diz respeito ao limiar de aceitação relacionado ao desempenho da ferramenta X. O limiar traz informações a respeito dos indicadores de desempenho que devem ser analisados. Ele será mais bem detalhado nos capítulos seguintes, quando forem apresentadas todas as etapas do desenvolvimento.

Com relação ao ator *Sistema*, existe uma gama maior de casos de uso. O sistema pode realizar atividades relacionadas com banco de dados, evidenciadas nos casos de uso *Atualizar Limiar, Salvar Limiar, Atualizar Report* e *Salvar Report.*

Além dos serviços relacionados com banco de dados, o sistema também possui casos de uso relacionados com cada etapa do processo. São esses os casos de uso *Parser XML, Análise de Reports* e *Exportar Resultados.*

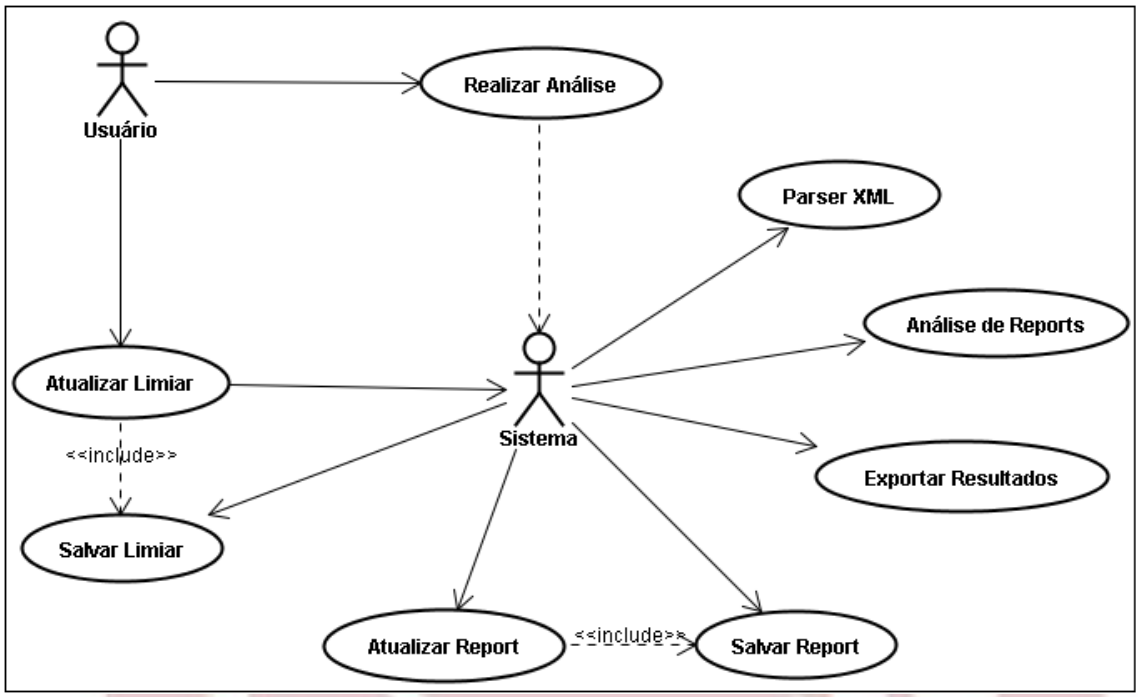

Figura 3. Diagrama de Casos de Uso – QA Soft

3.3.2 Requisitos Não Funcionais

Os requisitos *não funcionais* são usados com maior frequência para expressar alguns "atributos do sistema" ou "atributos do ambiente do sistema" (usabilidade, confiabilidade, desempenho, etc.) [7].

Outra definição para requisitos não funcionais pode ser: "*Os requisitos não funcionais, como o nome sugere, são aqueles que não dizem respeito diretamente às funções específicas fornecidas pelo sistema. Eles podem estar relacionados a propriedades de sistema emergentes, como confiabilidade, tempo de resposta e espaço em disco*" [12].

Para a concepção da ferramenta QA Soft, alguns requisitos não funcionais foram definidos, tais como:

- *Confiabilidade:* O sistema estará sempre disponível.
- *Robustez:* O sistema responde bem a falhas, realizando os tratamentos necessários e mostrando-se disponível novamente.
- *Desempenho:* O sistema realiza suas atividades em retorna os resultados ao usuário em tempo hábil e usando o mínimo de recursos de hardware.

- *Portabilidade:* o sistema é capaz de executar em diferentes plataformas.

3.4 Escolha do Indicador de Qualidade e cálculo do Limiar de aceitabilidade

Visto que a ferramenta foi concebida inicialmente com a ideia de analisar a performance de outros softwares, a escolha e criação do indicador de qualidade necessitou de um estudo sobre a ferramenta X a ser analisada e também sobre as linguagens NCL e LUA.

Durante o estudo, identificou-se que a ferramenta X realiza uma série de processamentos sobre suas entradas, os quais são divididos em etapas. A avaliação do NCL e do LUA é feita por componentes separados, sendo a parte do LUA muito mais onerosa do que a do NCL.

Uma análise sobre as duas linguagens mostra que a linguagem NCL é uma linguagem declarativa, composta de tags, semelhantes ao xml. É uma linguagem simples, com uma árvore sintática reduzida, quando comparada a linguagem LUA, que é uma linguagem funcional, com um maior nível de detalhes e estruturas.

Sendo assim, o consumo de processamento e de memória são maiores em aplicações que possuem LUA. Quanto maior o número de arquivos LUA na aplicação, maior o tempo de avaliação.

Outro agravante no tempo de avaliação é o número de linhas de código. Este não é tão relevante quanto o anterior, mas foi percebido que aplicações com menos linhas de código são avaliadas em menor tempo quando comparadas a aplicações com um total de linhas de código muito alto.

Sendo assim, foram identificados os pontos focais, os quais a ferramenta QA Soft precisa concentrar sua atenção: a influência do número de arquivos LUA e do número total de linhas de código no tempo total de avaliação (*elapse time*) do software X. Em ordem de peso:

1. Número de arquivos LUA;
2. Número de linhas de código.

4. Etapas do Desenvolvimento

Nesta seção serão descritas as etapas de modelagem e codificação da ferramenta QA Soft. Contudo, antes de começar a descrever as etapas, é importante apresentar a estrutura de dados que compõe o núcleo da análise, a partir dos dados apresentados no final do capítulo anterior.

4.1 Árvore Binária de Decisão

Uma árvore de decisão consiste numa hierarquia de nós internos e externos que são conectados por ramos. O nó interno, também conhecido como nó decisório, ou nó intermediário, é a unidade de tomada de decisão que avalia, através de teste lógico, qual será o próximo nó descendente ou filho. Em contraste, um nó externo (não tem nó descendente), também conhecido como folha ou nó terminal, está associado a um rótulo ou a um valor [11].

Para definir os limiares de aceitabilidade, foi usada uma suíte de testes de conformidade. Esta suíte é usada pelos desenvolvedores da ferramenta X, como forma de garantir confiabilidade, cobertura e agora, a performance do sistema. No total são 19 testes, os quais são bastante diversificados com relação à quantidade de linhas de código e quantidade de arquivos LUA. Ao final da rodada de testes, a ferramenta X gera saídas separadas para cada aplicação avaliada, com o detalhamento da análise.

Em seguida realizaram-se alguns cálculos estatísticos, como a média [8] de linhas de código e de arquivos LUA nas 19 aplicações de teste da suíte. O resultado é apresentado na Tabela - 1.

Indicador	Média
Linhas de código	2820
Arquivos LUA	12

Tabela 1 - Média de linhas de código e de arquivos LUA totais nas aplicações de teste da suite

Com as médias calculadas, foi possível construir a arvore de decisão. A média de linhas de código e a média de aplicações lua constituem diferentes camadas de nós internos da árvore. As 19 aplicações foram divididas de acordo com a quantidade de linhas de código e de arquivos LUA de cada uma. Foram criados seis subgrupos:

- Aplicações com mais de 12 arquivos LUA e mais de 2820 linhas de código;
- Aplicações com mais de 12 arquivos LUA e menos de 2820 linhas de código;
- Aplicações com menos de 12 arquivos LUA e mais de 2820 linhas de código;
- Aplicações com menos de 12 arquivos LUA e menos de 2820 linhas de código;
- Aplicações sem arquivos LUA e com mais de 2820 linhas de código;
- Aplicações sem arquivos LUA e com menos de 2820 linhas de código.

Para cada subgrupo foi realizado o cálculo do limiar de aceitabilidade, através de uma média ponderada [8]. Ao final, cada folha tem o seu peso e a árvore ficou como mostra a Figura 4.

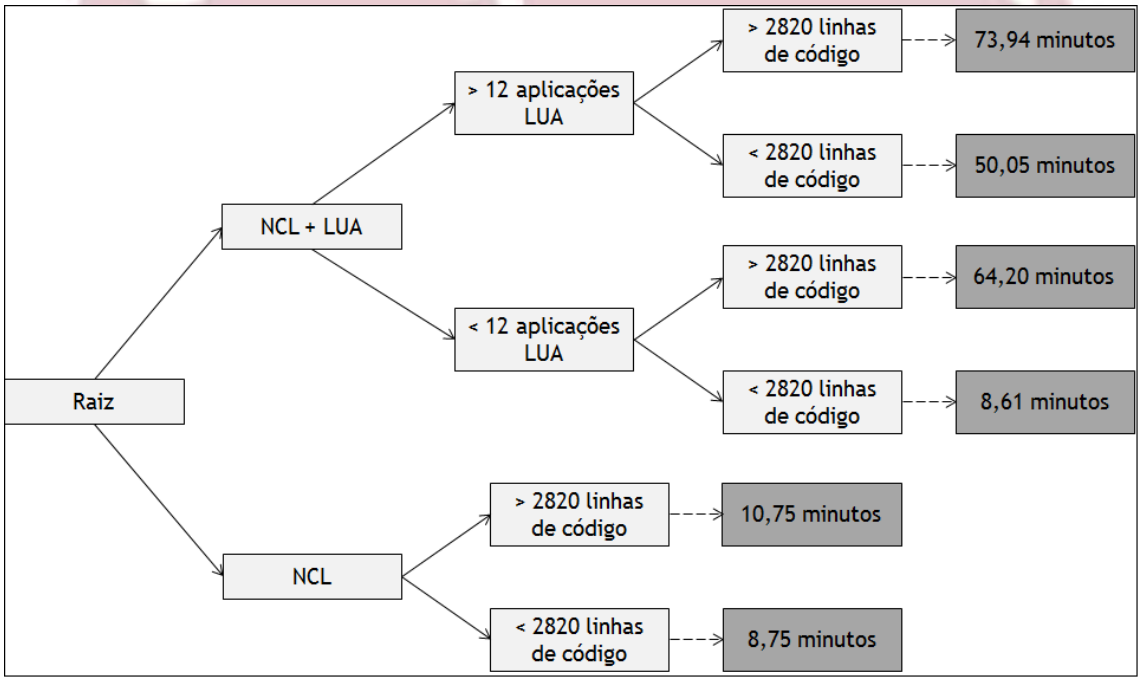

Figura 4 - Árvore de Decisão

4.2 Criação do Template XML de entrada da ferramenta

Com a árvore definida, falta representar as informações interpretadas pelo analisador de uma forma estruturada, compreensível pela ferramenta. Optou-se por estruturar a saída do software X em um arquivo XML [10] que sirva de entrada para o QA Soft. Para que a análise seja realizada, o XML precisa conter os dados do report do avaliador X, são eles:

- Elapse Time: representa o tempo total de avaliação da aplicação de teste pelo software X;
- Lua Files: quantidade de arquivos LUA usados pela aplicação de teste;
- Code lines: quantidade de linhas de código totais da aplicação de testes (soma de todas as linhas de código de todos os NCLs e LUAs que compõem a aplicação de teste).

Esses três dados, são de crucial importância para o QA Soft. Aliado a isso, é preciso que o XML contenha também o nome da aplicação, para que seja identificada no banco de dados quando forem realizadas consultas. O formato do arquivo XML é apresentado na Figura 5.

```xml
<?xml version="1.0" encoding="utf-8" ?>

<applicationData>
    <applicationName>ApplicationTest</applicationName>
    <elapseTime>30 min</elapseTime>
    <applicationSize>1.88 Mbytes</applicationSize>
    <codeLines>3000</codeLines>
    <totalLuaApps>35</totalLuaApps>

    <applicationFilesData>
        <applicationFileData>
            <applicationFileName>Application_001</applicationFileName>
            <applicationType>ncl/lua</applicationType>
            <codeLines>500</codeLines>
            <usedMedias>
                <quantity>34</quantity>
                <luaFiles>34</luaFiles>
            </usedMedias>
        </applicationFileData>

        <applicationFileData>
            <applicationFileName>Application_002</applicationFileName>
            <applicationType>ncl/lua</applicationType>
            <codeLines>2500</codeLines>
            <usedMedias>
                <quantity>14</quantity>
                <luaFiles>1</luaFiles>
            </usedMedias>
        </applicationFileData>
    </applicationFilesData>
</applicationData>
```

Figura 5 - Template XML do arquivo de entrada da ferramenta QA Soft

A tag **<aplicationData></aplicationData>** é a tag principal do XML. Dentro dela, o dados se dividem da seguinte forma:

- *applicationName*: referente ao nome da aplicação;
- *elapseTime*: tag que contém o tempo total de avaliação da aplicação;
- *applicationSize:* tag referente ao tamanho total da aplicação em Mega Bytes;
- *codeLines*: contém o número total de linhas da aplicação;
- *totalLuaApps*: total de arquivos LUA contidos na aplicação;
- *applicationFilesData*: representa a lista de NCLs que compõe a aplicação.
    - *applicationFileData*: engloba os detalhes de um determinado NCL. Nesta tag, é possível saber a quantidade de arquivos LUA

por aplicação NCL, bem como a quantidade de linhas de código e a quantidade de mídias totais por aplicação NCL;

- *applicationFileName*: nome do NCL;
- *applicationType*: nesta tag é explicitado se se trata de um NCL/LUA, ou apenas de um NCL puro;
- *codeLines*: número de linhas do NCL em questão;
- *usedMedias*: detalhamento sobre a quantidade de mídias usadas pelo NCL*;*
  - *quantity*: representa a quantidade total de mídias usadas pelo NCL em questão, incluindo os arquivos LUA;
  - *luaFiles*: quantidade de arquivos LUA totais usados pelo NCL em questão.

4.3 Modelagem UML dos Objetos

Nesta seção será descrito como foi realizada a modelagem dos objetos que mais tarde deram origem as classes do projeto do analisador.

A modelagem foi realizada seguindo os fundamentos básicos do UML (Unified Modeling Language) [4]. O projeto foi divido em seis diferentes camadas, representadas por *pacotes*. O motivo da separação foi puramente organizacional, visando modularidade e reuso de software [12]. A Figura 6 apresenta a modelagem UML do projeto.

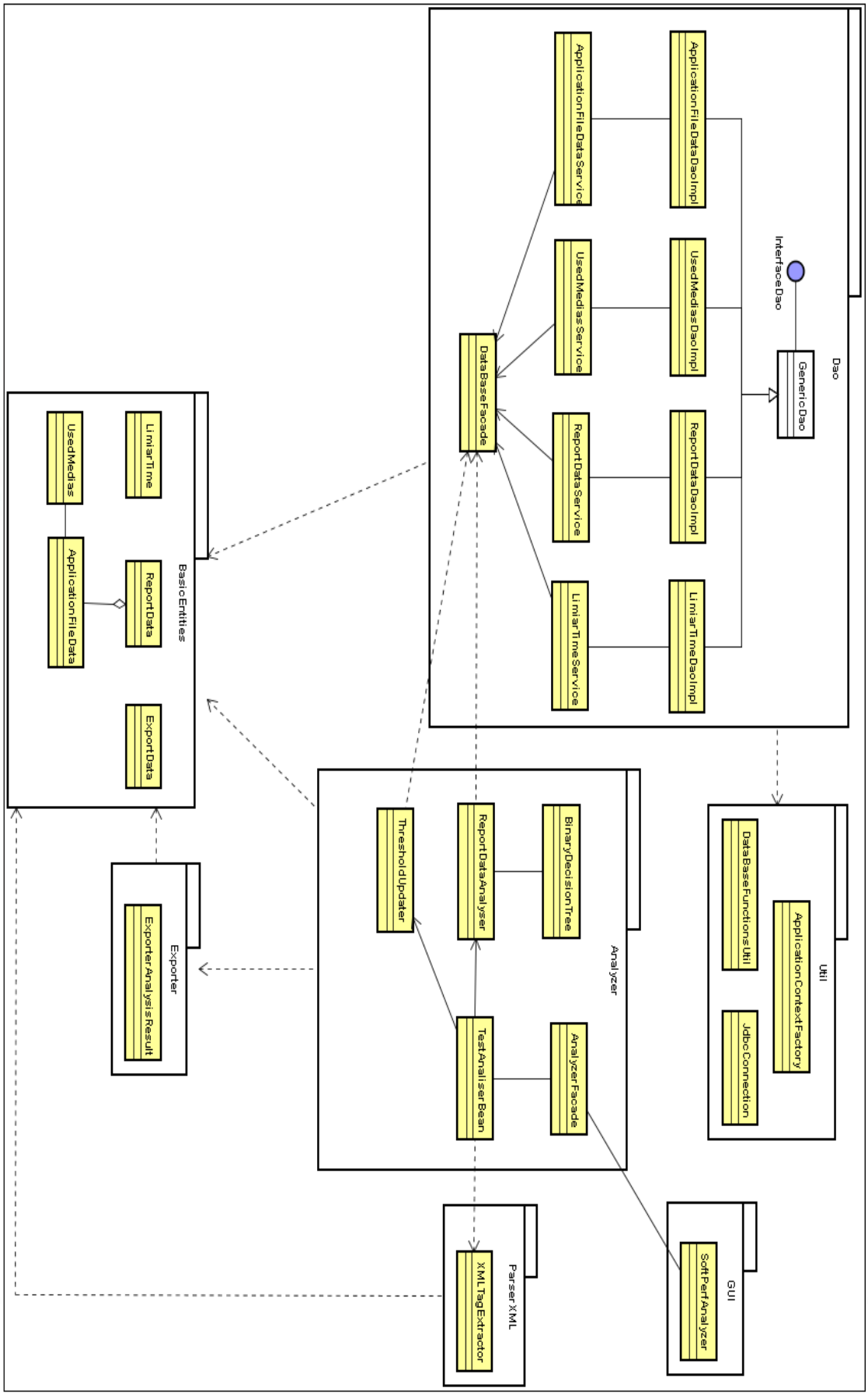

Figura 6 - Modelagem UML dos objetos do projeto QA Soft

Abaixo segue uma breve descrição de cada camada (os detalhes sobre as funcionalidades das camadas principais do projeto serão melhores descritas na seção seguinte):

- *Dao*: É a camada mais baixa do projeto, referente ao banco de dados. Esta camada guarda todos os serviços relacionados à persistência e consulta de objetos. De forma a centralizar as chamadas aos serviços do banco de dados foi usado o padrão de projeto *Facade* [5], evidenciado na classe *DataBaseFacade.* Outro padrão de projeto usado também nesta classe foi o *Singleton,* que garante uma única instancia para a fachada do banco, fornecendo um ponto global de acesso [5].
- *Util*: Camada de serviços relacionados à conexão do *Dao* com o banco de dados PostGresSQL. O Dao foi implementado usando Hibernate, logo, algumas classes de configuração são necessárias para estabelecer a conexão com o banco de dados.
- *BasicEntities*: Camada responsável por guardar todas as entidades persistentes do projeto. Este pacote é usado por praticamente todas as outras camadas. Os objetos que encapsulam o report XML e a saída da análise estão neste pacote.
- *ParserXML*: Camada responsável pela primeira etapa da ferramenta de análise: O Parser do arquivo XML. Nesta camada se encontra a classe *XMLTagExtractor,* responsável por "transformar" o XML em um Objeto Java.
- *Analyzer*: É o *core* do projeto. Esta camada realiza todas as chamadas e direcionamentos para os demais serviços contidos nas outras camadas. A classe *AnalyzerFacade* também faz uso dos padrões de projeto *Facade* e *Singleton* e é responsável pela comunicação entre a interface gráfica, contida na camada *GUI,* com serviços da ferramenta.
- *GUI*: Camada de comunicação com o usuário. Esta fornece a interface gráfica pela qual o usuário solicita os serviços do QA Soft. Ela se comunica diretamente com a classe *AnalyzerFacade*. As figuras 7, 8 e 9 apresentam, respectivamente, a tela inicial, a tela de atualização do

limiar de aceitabilidade (*threshold*) e a tela de escolha do report para análise.

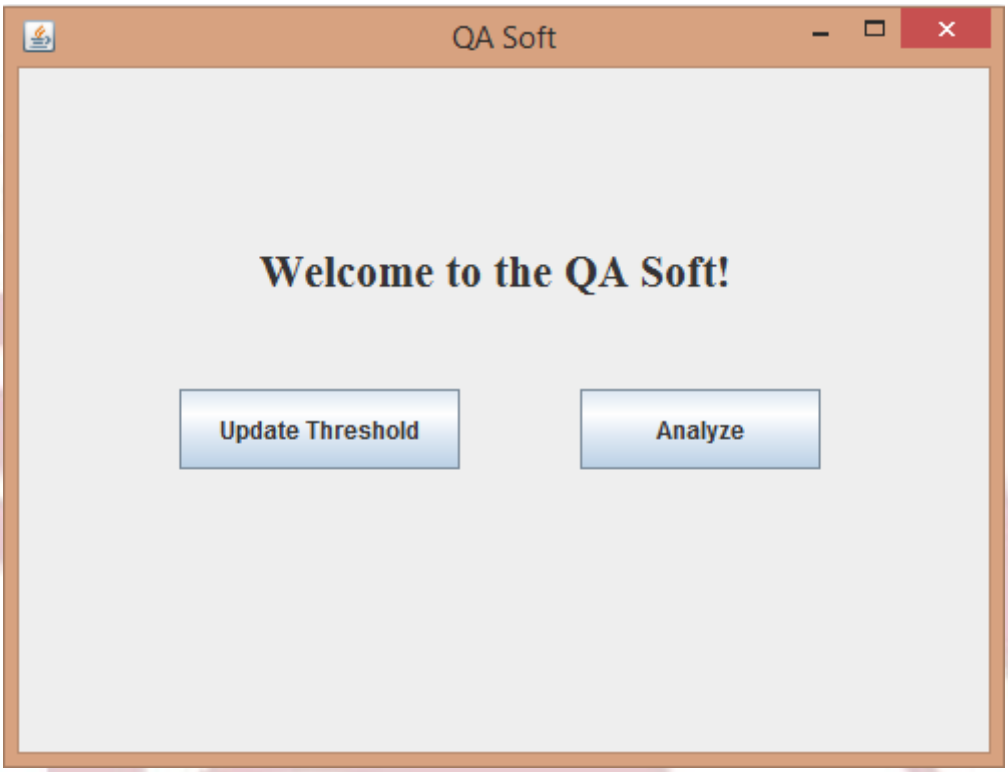

Figura 7 - Tela Inicial da ferramenta QA Soft

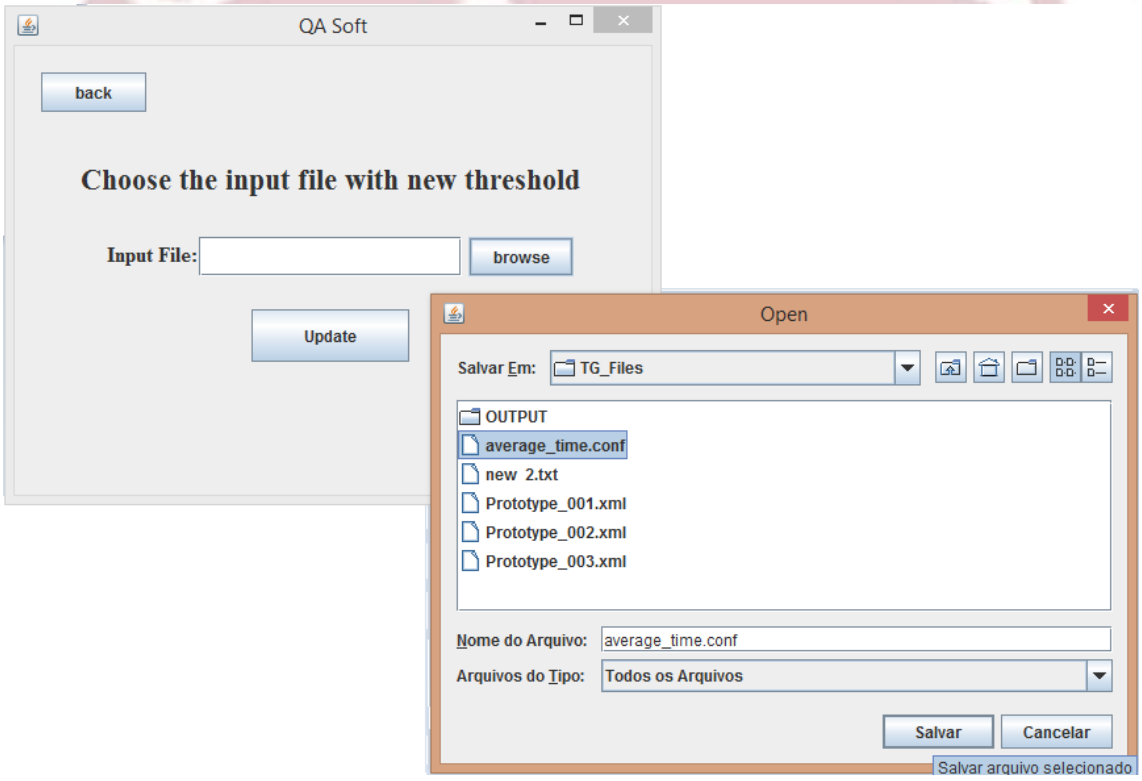

Figura 8 - Tela de atualização do limiar de aceitação

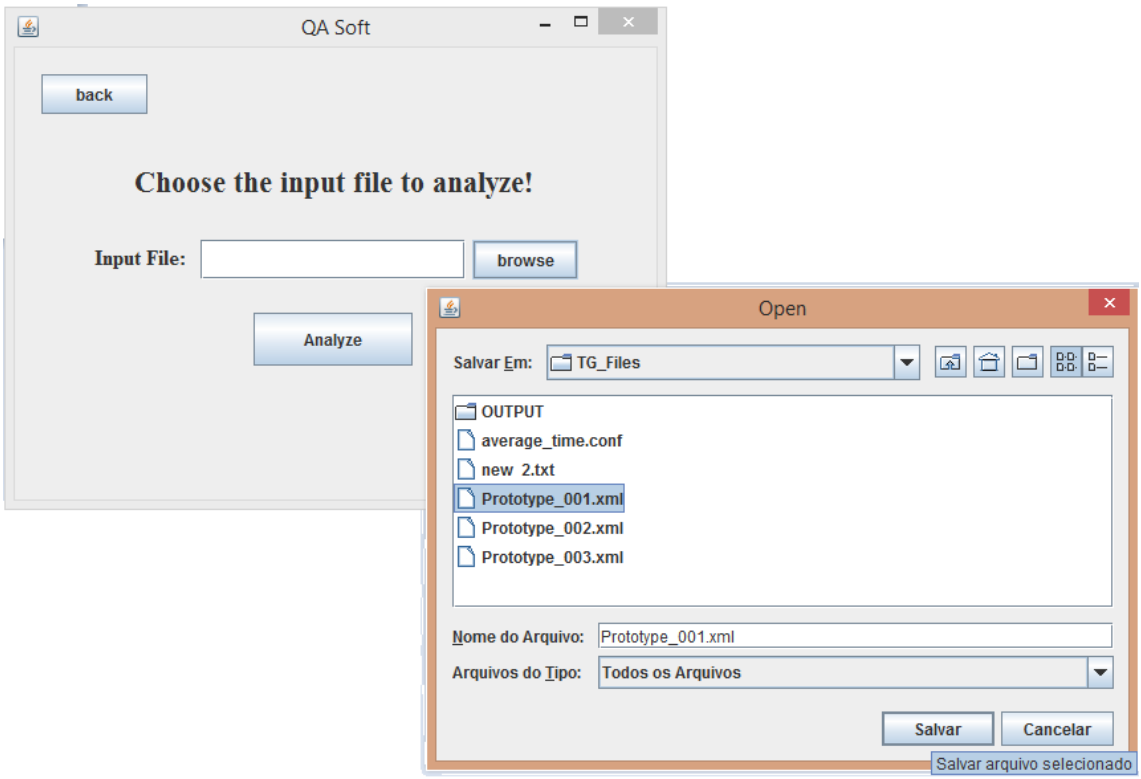

Figura 9 - Tela de escolha do report a ser analisado

4.4 Análise

Nesta seção será detalhado o processo de análise realizado pela ferramenta. Basicamente este processo é dividido em três etapas: *Extração dos dados XML*, *Análise comparativa* e *Organização dos dados e formatação de saída*.

4.4.1 Parser XML

O pacote *parserXML* é responsável por guardar os serviços de extração de dados do arquivo de entrada e encapsula-los em objetos Java. Para este procedimento, foi usada a API Java *XStream*, responsável por serializar objetos Java para XML e também realizar o processo contrário [15].

A classe *XMLTagExtractor* realiza o parser XML usando o *XStream*. Primeiramente, ela realiza a leitura do arquivo XML e converte-o para uma String, através do método *readXMLFile*.

Em seguida, o XStream realiza o parser, através do método *aliasType* [15]. Este método associa a tag XML, especificada como parâmetro, com a classe Java

também passada como parâmetro. Para tanto, a tag precisa ter exatamente as mesmas informações que a classe, ou seja, se a classe Java possui três atributos, então a tag precisa conter três tags filhas com os mesmos nomes dos atributos da classe Java.

Após o uso do *aliasType*, todas as tags XML já estão guardadas no XStream, faltando apenas criar o objeto que receberá as informações contidas no XStream. O objeto responsável por encapsular as informações da entrada é o *ReportData*. Logo, como uso de mais uma funcionalidade do XStream, o método *fromXML* converte todos os dados serializados no respectivo Objeto Java . A Figura 10 apresenta o código referente à construção do objeto *ReportData* a partir do XML de entrada.

```
/**
 * This method calls the readXMLFile method for get the XML string code and then use the XStream API to build the ReportData object
 * @param xmlFile
 * @return reportData
 */
public static ReportData buildReportDataByXMLFile(File xmlFile) {
    ReportData reportData = new ReportData();

    String xmlCode = readXMLFile(xmlFile);
    XStream xStream = new XStream();

    xStream.aliasType("applicationData", ReportData.class);
    xStream.aliasType("applicationFilesData", ArrayList.class);
    xStream.aliasType("applicationFileData", ApplicationFileData.class);
    xStream.aliasType("usedMedias", UsedMedias.class);

    reportData = (ReportData)xStream.fromXML(xmlCode);

    return reportData;
}
```

Figura 10. Trecho de código referente ao parser XML

4.4.2 Analyzer

No pacote *Analyzer*, a classe *TestAnalyzerBean* chama o parser XML e recebe o *ReportData* como resultado. A partir de então, ela instancia a classe *ReportDataAnalyzer*, responsável por realizar a análise. O método *analyze*, recebe como parâmetro o *ReportData* gerado pelo parser XML, e se divide em dois procedimentos:

- Análise baseada na árvore de decisão: Neste passo, o analisador chama a árvore de decisão, definida anteriormente, e verifica se a ferramenta X avaliou a aplicação contida no *ReportData* em tempo menor, igual ou maior do que o limiar aceitável. Para tanto, a análise faz uso dos

atributos referentes à quantidade de linhas de código, quantidade de arquivos LUA e tempo total de avaliação (elapseTime).

- Análise comparativa com ReportData do banco de dados: Se por acaso esta mesma aplicação já tive sido analisada pelo QA Soft, ela terá guardada no banco de dados a última versão de seu *ReportData* e dessa forma, pode-se fazer um comparativo, a fim de se saber o que mudou da última avaliação até o momento atual, e que possivelmente possa ter causado aumento, redução, ou mesmo nenhuma alteração no tempo de avaliação da ferramenta X. Se não existir nenhum histórico no banco de dados, então o ReportData atual é salvo e o resultado da análise será apenas o resultado da árvore de decisão.

Durante o processo de análise realizado pela árvore de decisão, a aplicação representada pelo *ReportData* "caminha" sobre as possibilidades da árvore, até se encaixar em uma folha. Nesse ponto, comparasse o tempo contido na aplicação, com o tempo do limiar de aceitabilidade.

Além de gerar o resultado dessa comparação, a análise também gera uma possível causa de um aumento ou redução do tempo comparado ao limiar. Com o uso dos indicadores de desempenho, que são, em ordem de importância, a quantidade de arquivos LUA e a quantidade de linhas de código, geram-se percentuais referentes às médias encontradas. Por exemplo, se a quantidade de arquivos LUA da aplicação é de 40, então esta aplicação possui 33% arquivos LUA a mais do que a media calculada (a saber, 12). Se o tempo de avaliação desta aplicação for menor do que o limiar de aceitabilidade, então houve uma melhoria de desempenho da ferramenta X em sua avaliação.

No tocante à comparação com uma avaliação anterior, armazenada no banco de dados, a comparação é feita para cada atributo. A primeira comparação é com o tempo de avaliação de cada um (atributo *elapseTime*). A partir de então, compara-se o número de aplicações LUA e o número de linhas de código.

Assim como no caso da árvore de decisão, na comparação entre versões do *ReportData* também é possível gerar uma recomendação a respeito das causas da melhora ou piora no tempo de avaliação do software X. Neste caso, existe uma granularidade maior do resultado, pois além de falar em quanto tempo a mais ou a menos a avaliação foi feita, é possível gerar o percentual de aumento ou redução de cada indicador. Por exemplo, se na versão anterior do *ReportData* a quantidade de

arquivos LUA era menor e o *ReportData* atual apresenta um tempo de avaliação reduzido, então significa que houve uma melhoria na ferramenta X, pois ela está conseguindo avaliar a mesma aplicação em menos tempo, ainda que desta vez ela esteja com mais arquivos LUA presente.

Ao final da análise, resultados são armazenados no objeto *ExportData*, o qual é usado como entrada da camada *Exporter* para geração da saída formatada.

4.4.3 Exporter

O Exporter é responsável por formatar os dados gerar o arquivo de saída da análise. O arquivo final é um arquivo de texto comum, contendo as informações coletadas.

A classe *ExporterAnalysisResult* é responsável por serializar os dados do *ExportData* em um *stream* de saída. A classe *ExportData* possui um *Override* do método *toString()*, de forma a personalizar a saída dos dados.

De forma simplória o método *export* da classe *ExporterAnalysisResult* cria os objetos de escrita em arquivo e cria o arquivo de saída, formatado com o nome da aplicação seguido da data e hora atual da geração.

A Figura 11 mostra o arquivo de saída formatado.

```
#########################################################################################################
################################   QA SOFT - EXPORT DATA   ##############################################
#########################################################################################################

#APPLICATION NAME: ApplicationTest
#ANALYSIS DETAILS:
    -> COMPARATION WITH THRESHOLD:
        - The application "ApplicationTest" evaluation time took less than the threshold
        - Number of LUA files: 35 (25% above the average)
        - Total number of code lines: 3000 (6.38% above the average)
        - elapse time: 45 min
        - Maximum acceptable time: 71.4 min (36% reduction in the elapse time)

    -> COMPARATION WITH LAST EVALUATION:
        - The total size of the application not changed
        - The total evaluation time of the application not changed
        - The number of LUA files not changed
        - The number of code lines of the application not changed
        - The elapse time of the new evaluation was less than the last evaluation
            - New evaluation elapse time:  45.0 min
            - Last evaluation elapse time: 78.5 min
        - Compared with the last evaluation, the elapse time decreased by 33.5 min

# ANALYSIS RESULT:
    - Evaluation showed improvement of 36% in the elapse time with respect to the acceptable
      time (threshold)
    - Compared with the last evaliation, the elapse time decreased by 42.5%
```

Figura 11 - Arquivo final resultante da análise

4.5 Resultados

Nesta seção, serão apresentados os resultados obtidos a partir dos testes realizados no QA Soft. Para tanto, inicia-se falando sobre a metodologia de testes usada neste trabalho, que garantiu a corretude dos dados obtidos nos resultados. Após isso, serão descritas as três visões de resultado deste protótipo (a quem se destina e porque foi feita essa separação).

Primeiramente, com relação à metodologia de testes que gerou os resultados apresentados a seguir, foram feitos testes contínuos da mesma aplicação de entrada sobre a ferramenta X em dois diferentes momentos: Em um primeiro snapshot, que representa o momento em que não foram identificados possíveis pontos de melhoria de desempenho; e em um segundo momento (segundo snapshot) no qual foram atribuídas melhorias na ferramenta X analisada pelo QA Soft. Sendo assim, as duas formas de se obter os resultados da análise (através da árvore de decisão e através da comparação com a avaliação no primeiro snapshot) usam os resultados gerados pela ferramenta analisada X para saber se de fato houve um ganho de desempenho após as melhorias implementadas.

A partir da saída produzia pelo QA Soft, podem-se gerar visões para diferentes *stakeholders*: O desenvolvedor da ferramenta X, o CM da ferramenta X (responsável pela consolidação e geração do release) e o Cliente da ferramenta X.

Para o desenvolvedor do software X, o arquivo resultante apresentado anteriormente pela Figura 11 já mostra as informações relevantes para que ele possa voltar para a ferramenta e procurar realizar as melhorias (para o caso em que a análise apresentar uma piora no tempo de avaliação) ou de fato considerar as alterações realizadas como melhorias (para o caso em que a análise apresentar redução no tempo de avaliação) e a partir de então realizar os *commits* com os resultados adquiridos.

Contudo, um gráfico pode ser gerado para fins de visualização. A Figura 12 apresenta um gráfico com os resultados de duas avaliações feitas pelo software X sobre a mesma aplicação de entrada (em dois momentos diferentes). Essas informações estão contidas, conforme apresentado anteriormente, nos XMLs que servem como entrada do QA Soft. Nesse caso, toda avaliação feita pela ferramenta X que é colocada como entrada para o QA Soft será armazenada no banco de dados como registro a ser comparado numa próxima vez. Então, sempre que uma aplicação é colocada para análise no QA Soft, ela é comparada com a última avaliação realizada, de forma a identificar melhorias na ferramenta X nesse intervalo entre avaliações.

Para um CM da ferramenta X, é importante saber em quanto tempo a ferramenta X realiza a avaliação da suíte padrão de testes, para saber quando tempo ele precisa para gerar a release. Visto que um release necessita de testes de integração, é preciso ter a noção de quanto tempo cada tipo de teste, de cada frente do projeto, necessita antes de se estimar o tempo de geração de um release. Dessa forma, o tempo de release pode ser ré estimado, para mais ou para menos, de acordo com a melhoria ou não apresentada nos resultados do QA Soft.

Para um processo de release que levava um tempo de 200 minutos, sendo que 70 minutos eram apenas execução da suíte de testes pela ferramenta X, pode-se plotar um gráfico com o tempo total do release antes das melhorias, o tempo total do release depois das melhorias e os tempos de execução da suíte de testes na ferramenta X, antes e depois das melhorias. A Figura 13 apresenta o gráfico de desempenho do release na visão do CM, usando os resultados obtidos pela ferramenta QA Soft.

Para um cliente da ferramenta X, o que importa é que a ferramenta esteja avaliando as aplicações em tempo hábil, definido por ele. Por exemplo, se o cliente definiu que o tempo aceitável de avaliação para a mais complexa das aplicações deva

ser de 15 minutos fixos, então é possível gerar um gráfico com os resultados adquiridos pelo QA Soft e plotar um gráfico apresentando as melhorias adquiridas nas últimas avaliações. A Figura 14 mostra apresenta um gráfico resultante da análise feita pela ferramenta proposta em um report gerado pela ferramenta X após analisar uma aplicação de teste. O gráfico é apresentado na visão do cliente da ferramenta, tendo em vista que o limiar aceito pelo mesmo é de 15 minutos.

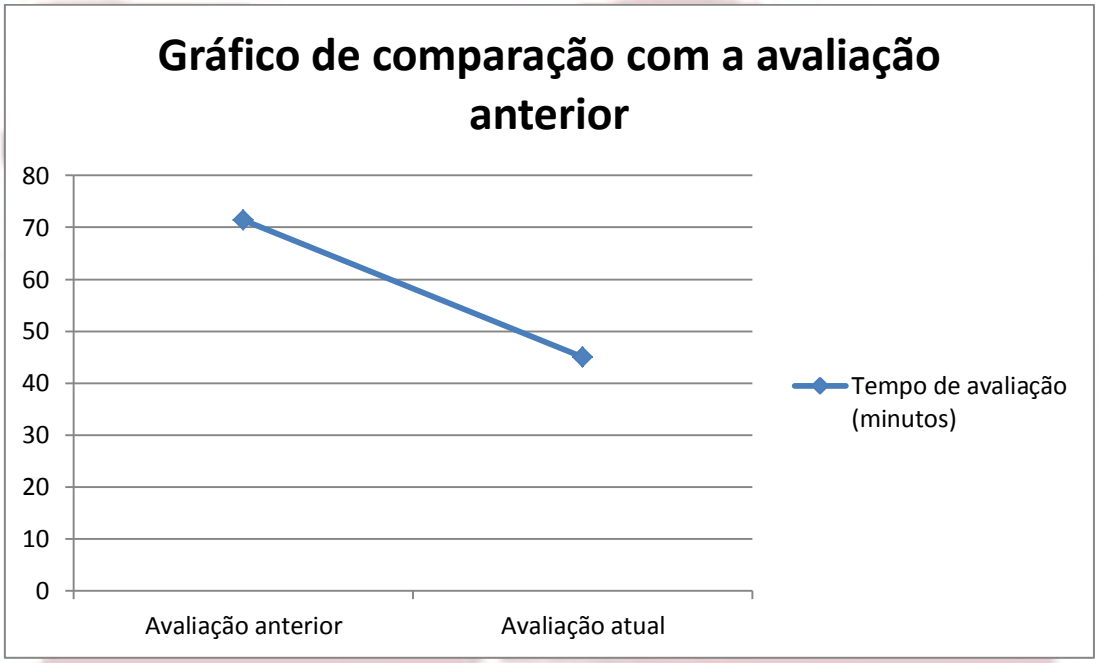

Figura 12 - Gráfico de resultado na visão do desenvolvedor da ferramenta X

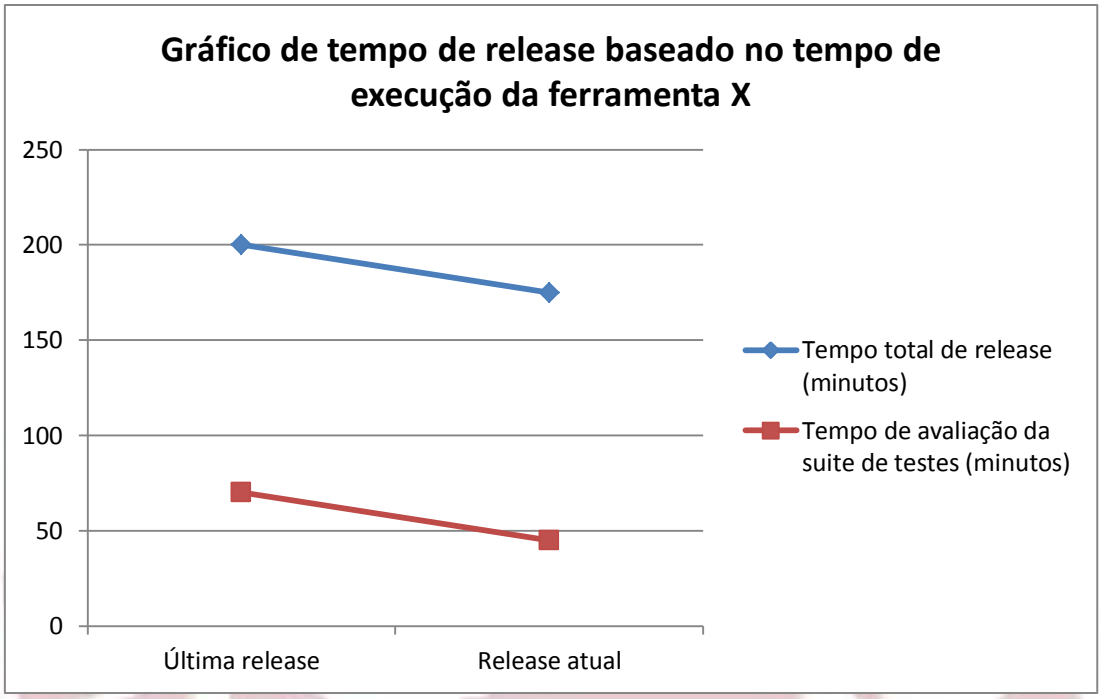

Figura 13. Gráfico de resultado na visão do CM da ferramenta X

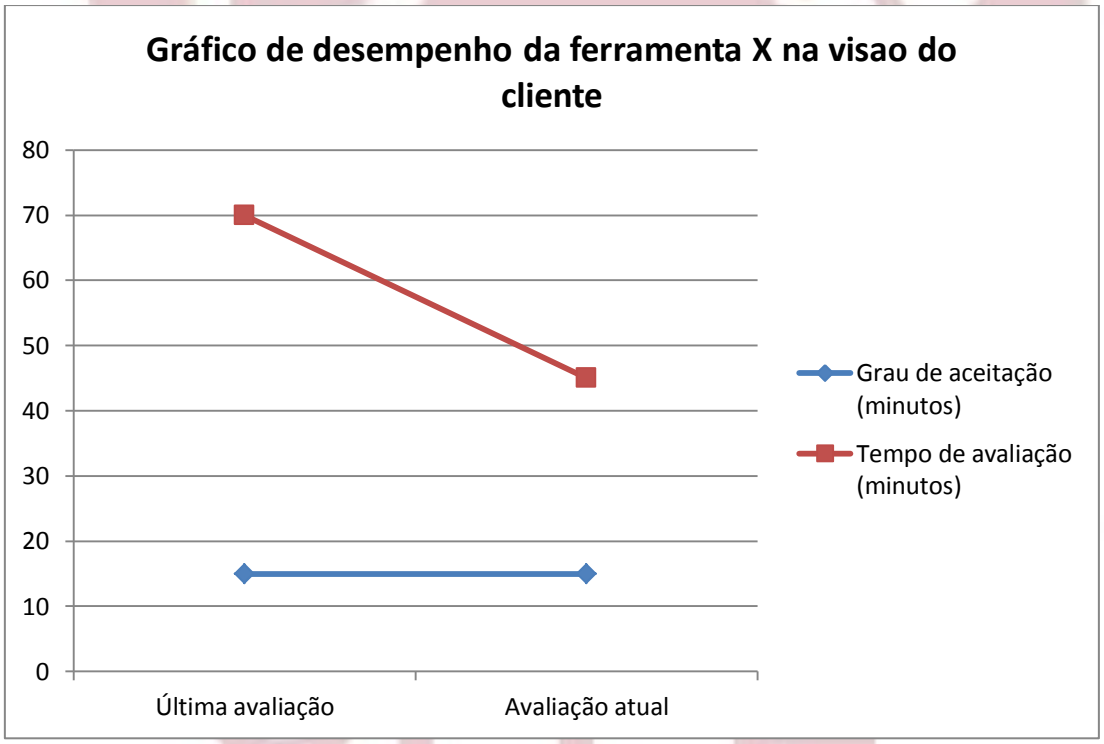

Figura 14. Gráfico de resultado na visão do cliente da ferramenta X

5. Conclusão

A ferramenta QA Soft foi concebida, em sua versão protótipo, com a intenção de analisar o desempenho de um software, a partir de indicadores de performance. A análise foi realizada com sucesso, a partir de informações colhidas da ferramenta analisada, que serviram de entrada para o QA Soft.

Os resultados obtidos foram proveitosos para todos os stakeholders: desenvolvedores do software analisado, CM responsável pelo release e o cliente da ferramenta analisada. Para cada um foi possível gerar uma saída diferente contendo informações que eram relevantes para as suas necessidades.

A ferramenta ainda apresenta algumas limitações, referentes aos gráficos de resultado, esses só podem ser obtidos manualmente, utilizando-se das saídas formatadas da ferramenta.

Com relação às dificuldades enfrentadas durante as etapas deste trabalho, pode-se citar a definição do indicador de qualidade de software, a elaboração da árvore de decisão, a qual necessitou, além da revisão bibliográfica, de muita atenção aos cálculos e de muitos testes para ter certeza da eficiência e da precisão dos dados.

Ainda é possível evoluir a ferramenta para poder analisar mais detalhadamente as entradas. Para uma próxima versão do QA Soft, além de avaliar o desempenho de um software qualquer, a ferramenta poderia também analisar a confiabilidade e a cobertura com relação aos requisitos listados pelo cliente. Para tanto, é preciso colher mais informações da ferramenta analisada, que influenciem nesses indicadores de qualidade, bem como codificar novas estruturas de análise de dados.

Referências Bibliográficas

[1] ABNT: Associação Brasileira de Normas Técnicas. Disponível em: http://www.abnt.org.br/m3.asp?cod_pagina=929. Acesso em 12 de Agosto de 2014.

[2] Ananias, R. R. T. Estudo comparativo entre ferramentas de Gerência de Requisitos. Trabalho de Graduação. Universidade Federal de Pernambuco, Recife, 2009.

[3] Ferramentas de análise estática. Disponível em http://msdn.microsoft.com/pt-br/magazine/dd263071.aspx. Acesso em 2 de Agosto de 2014.

[4] Fowler, M. UML Distilled - Third Edition. Addison Wesley. 2004.

[5] Gamma, E.; Helm, R.; Jonhson, R.; Vilssides, J. Design Patterns. Addison Wesley Longman. Editora Bookman. 2000 (Reimpressão 2008).

[6] Guerra, A.; Colombo, R. Qualidade de Produtos de Software. 2009.

[7] Leffingwell, D.; Widrig, D. Managing Software Requirements: A Unified Aproach – Addison-Wesley object technology series. Addison Wesley. 2000.

[8] Magalhães, M. N.; Lima, A. C. P. Noções de Probabilidade e Estatística – 6ª Edição, Editora da Universidade de São Paulo. 2004.

[9] Pezzè , M.; Young, M. Teste e Análise de Software: Processos, Princípios e Técnicas. Editora Bookman. 2008.

[10] Portal da Educação: Artigos de Informática - XML. Disponível em http://www.portaleducacao.com.br/informatica/artigos/4310/xml#!1. Acesso em 23 de Maio de 2014.

[11] PUC-RIO: Árvore de Decisão. Disponível em http://www.maxwell.vrac.puc-rio.br/3710/3710_4.PDF. Acesso em 30 de Junho de 2014.

[12] SOMMERVILLE, I. ENGENHARIA DE SOFTWARE. PEARSON EDUCATION DO BRASIL. 2003.

[13] TESTE DE DESEMPENHO: CONCEITOS, OBJETIVOS E APLICAÇÕES. DISPONÍVEL EM HTTP://WWW.LINHADECODIGO.COM.BR/ARTIGO/3256/TESTE-DE-DESEMPENHO-CONCEITOS-OBJETIVOS-E-APLICACAO-PARTE-1.ASPX. ACESSO EM 3 DE JUNHO DE 2014.

[14] VIEIRA, A. D.; BOURBON, B. C.; LUZIA, C.; BRANDÃO, H.; ARRAIS, T. UTILIZAÇÃO DA ISSO 12119 NA CONSTRUÇÃO DE TESTES NA TÉCNICA TEST-DRIVEN DEVELOPMENT. 2004. CENTRO DE INFORMÁTICA, UNIVERSIDADE FEDERAL DE PERNAMBUCO, RECIFE.

[15] XTREAM. DISPONÍVEL EM HTTP://XSTREAM.CODEHAUS.ORG/. ACESSO EM 23 DE MAIO DE 2014.