Joenio Marques da Costa

Extração de Informações de Dependência entre Módulos de Programas C/C++

Salvador-Bahia

2009

Joenio Marques da Costa

Extração de Informações de Dependência entre Módulos de Programas C/C++

Monografia apresentada ao Curso de graduação em Bacharelado em Informática, Universidade Católica do Salvador, como requisito parcial para obtenção do grau de Bacharel em Informática.

Orientador:
Prof. M.Sc. Antonio A. S. Terceiro

Co-orientador:
Prof(a). M.Sc. Maria Mesquita Mota

UCSAL - UNIVERSIDADE CATÓLICA DO SALVADOR
FACULDADE DE INFORMÁTICA

Salvador-Bahia

2009

Monografia sob o título *'Extração de Informações de Dependência entre Módulos de Programas C/C++'*, apresentada por Joenio Marques da Costa como requisito parcial para obtenção do grau de Bacharel em Informática, e aprovada em 13 de Junho de 2009, em Salvador-Bahia, pela banca examinadora constituída por:

Prof(a). M.Sc. Maria Mesquita Mota
Universidade Católica do Salvador

Prof. Antonio Claudio Pedreira Neiva
Universidade Católica do Salvador

Prof. Osvaldo Requião Melo
Universidade Católica do Salvador

RESUMO

A arquitetura de software de um programa de computador representa a estrutura básica e os relacionamentos externamente visíveis de seus componentes. A documentação desta arquitetura é de suma importancia para acompanhar e avaliar a evolução de um sistema, bem como para medir seus atributos de modularidade como acoplamento e coesão. Uma ferramenta capaz de extrair esta documentação automaticamente possibilita acompanhar e medir esses atributos durante a evolução de um projeto de software. Este trabalho apresenta a implementação de uma ferramenta para extração automática de informações de dependência entre módulos de programas escritos em C/C++ com foco em seus atributos de modularidade visando o acompanhamento da evolução do sistema. Ao final do trabalho é feita uma avaliação desta ferramenta a partir de um estudo de caso.

Palavras-chave: engenharia de software, métricas, arquitetura de software, acoplamento, coesão, C, C++, dependência.

ABSTRACT

The software architecture of a computer program represents the basic structure and the externally visible relationships of those components. The documentation of this architecture is of great importance to monitor and evaluate the evolution of a system and to measure their attributes of modularity as coupling and cohesion. A tool capable of extracting this documentation automatically enables monitor and measure these attributes during the evolution of a software project. This paper presents the implementation of a tool for automatic extraction of information of dependence between modules for programs written in C/C++ with focus on attributes of modularity to monitor the evolution of the system. At the end of this paper is done an evaluation of this tool through a case study.

Keywords: software engineering, metrics, software architecture, coupling, cohesion, C, C++, depedency.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

OO	Orientação a Objetos,	p. 13
WMC	Weighted Methods per Class,	p. 13
DIT	Depth of the Inheritance Tree,	p. 13
NOC	Number of Children,	p. 13
CBO	Coupling Between Objects classes,	p. 13
RFC	Response For a Class,	p. 13
LCOM	Lack of Cohesion in Methods,	p. 13
GCC	GNU C Compiler,	p. 18
IDL	Interface Definition Language,	p. 19
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language,	p. 19
HTML	HyperText Markup Language,	p. 19
RTF	Rich Text Format,	p. 19
PDF	Portable Document Format,	p. 19
GPL	GNU Public License,	p. 19
API	Application Programming Interface,	p. 19
GTK	GIMP Tool Kit,	p. 31

1 INTRODUÇÃO

Arquitetura de software de um programa é a estrutura que define as propriedades externamente visíveis e o relacionamento entre os grandes componentes estruturais de um sistema (PRESSMAN, 2005). Esta arquitetura pode estar ou não documentada em forma de diagramas, grafos, tabelas ou documentos. Todo software possui uma arquitetura, ainda que esta não esteja documentada. Ter esta arquitetura documentada é útil para extrair métricas de modularidade ou para estudar o impacto de possíveis alterações de um projeto (TERCEIRO, 2008).

A arquitetura de grandes projetos de software raramente são documentados e quando são usualmente estão desatualizados (HASSAN; JIANG; HOLT, 2005). Ter a arquitetura documentada e atualizada é de suma importância para o acompanhamento do projeto além de servir como forma de apoio para entrada de novos desenvolvedores.

Existem hoje diversas ferramentas capazes de extrair do código fonte ou do código objeto a arquitetura de software de um sistema, durante a elaboração deste trabalho algumas ferramentas foram pesquisadas (HASSAN; JIANG; HOLT, 2005) e entre as opções encontradas a maioria faz uso do código objeto, ou seja, dependem da compilação do código fonte.

Uma desvantagem em se utilizar código objeto é que fica impossível analisar projetos que não compilem mais, seja por falhas no código ou por possuir dependencias não satisfeitas, além de gerar perda de informação, como acontece com macros em projetos C/C++ que são descartadas durante a compilação (HASSAN; JIANG; HOLT, 2005).

Este trabalho baseia-se na implementação de uma ferramenta que possibilite a extração de informação de dependências entre módulos de programas C/C++ baseada em código fonte.

Este trabalho está organizado da seguinte forma: no capítulo 2 são abordados conceitos básicos de arquitetura de software e atributos de modularidade como acoplamento e coesão; no capítulo 3 será demonstrado como a ferramenta foi implementada; no capítulo 4 os resultados obtidos pela ferramenta são avaliados com um estudo de caso; o capítulo 5 apresenta as conclusões e sugestões para trabalhos futuros.

2 CONCEITOS

Neste capítulo serão apresentados os conceitos fundamentais utilizados neste trabalho para a avaliação dos resultados gerados pela ferramenta desenvolvida.

2.1 ARQUITETURA DE SOFTWARE

Arquitetura de software de um programa é a estrutura que define as propriedades externamente visíveis e o relacionamento entre os grandes componentes estruturais do sistema (PRESSMAN, 2005).

Pressman (2005) destaca 3 razões principais sobre a importância da arquitetura de software:

- Representações da arquitetura de software constituem um facilitador da comunicação entre todas as partes interessadas no desenvolvimento de um sistema.
- A arquitetura destaca decisões iniciais de projeto que terão um impacto profundo em todo o trabalho de engenharia de software que se segue.
- A arquitetura compõe uma representação de simples entendimento de como o sistema é estruturado e como se interrelacionam os seus componentes.

Apesar de Pressman, aqui, destacar a importância da arquitetura de software nas decisões iniciais do projeto, ela é de suma importância durante todo o processo de desenvolvimento de um projeto.

Neste trabalho a documentação da arquitetura de software de um programa será utilizada com o objetivo de calcular métricas que possibilitem avaliar a qualidade do sistema em relação aos seus atributos de modularidade.

2.2 ATRIBUTOS DE MODULARIDADE

A medição é um elemento fundamental para qualquer processo de engenharia (PRESSMAN, 2005). As métricas de software são um modo poderoso de prover indicadores de modularidade da arquitetura de um sistema(SANT'ANNA, 2008).

Um dos conjuntos de métricas de software OO mais amplamente referenciado foi proposto por Chidamber e Kemerer (PRESSMAN, 2005). Eles propuseram seis métricas:

- **WMC** - Métodos ponderados por classe (weighted methods per class).
- **DIT** - Profundidade de árvore de herança (depth of the inheritance tree).
- **NOC** - Número de filhos (number of children).
- **CBO** - Acoplamento entre as classes de objetos (coupling between objects classes).
- **RFC** - Resposta de uma classe (response for a class).
- **LCOM** - Falta de coesão em métodos (lack of cohesion in methods).

Este trabalho fundamenta-se nas métricas CBO e LCOM, que são indicadores, respectivamente, de acoplamento e coesão, e refletem alguns aspectos da modularidade de um projeto de software.

2.2.1 ACOPLAMENTO

Acoplamento é a medida de conexões entre os componentes de um sistema, ele representa o nível de interdependências entre os módulos de um sistema, quanto maior o acoplamento maior a complexidade.

A métrica para cálculo de acoplamento utilizada neste trabalho será CBO (acoplamento entre as classes de objetos), esta métrica é calculada contando-se o número de colaborações de um módulo com as outras entidades do sistema. Um grafo simples para cálculo desta métrica é apresentado na figura 2.1, onde: *module1* colabora com dois outros módulos (*module2* e *module3*), então $CBO(module1) = 2$.

Valores de CBO altos podem significar que a reusabilidade de um módulo é pequena (PRESSMAN, 2005), além de implicar que a manutenção e os testes serão mais complexos de ser feitos. Em geral o ideal é que os valores de CBO sejam baixos.

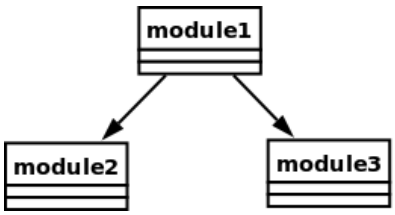


Figura 2.1: Grafo de colaboração exemplo para cálculo de CBO

2.2.2 COESÃO

Coesão é a medida que define o quanto um módulo de um programa está focado em solucionar um único problema. Pressman (2005) define coesão da seguinte forma:

> *coesão* implica que um componente ou classe encapsule somente os atributos e operações muito relacionados entre si e com a classe ou componente propriamente dito. (PRESSMAN, 2005)

É desejável que a coesão dos componentes de um sistema seja alta, isto garante que o acoplamento do sistema seja baixo e como consequência a complexidade do sistema cai facilitando a manutenção e o desenvolvimento.

A métrica para cálculo de coesão utilizada neste trabalho será LCOM (falta de coesão em métodos), proposta por Chidamber e Kemerer (1993). LCOM é a diferença entre o número de métodos num mesmo componente que compartilham atributos e o número de métodos que não compartilham atributos, se nenhum método compartilha atributos então $LCOM = 0$.

A figura 2.2 demonstra a representação interna de um módulo, onde o atributo *nome* é compartilhado entre dois métodos (*imprime* e *exporta*) e o método *aniversario* não compartilha nenhum atributo com outro método, então $LCOM = 2 - 1 = 1$.

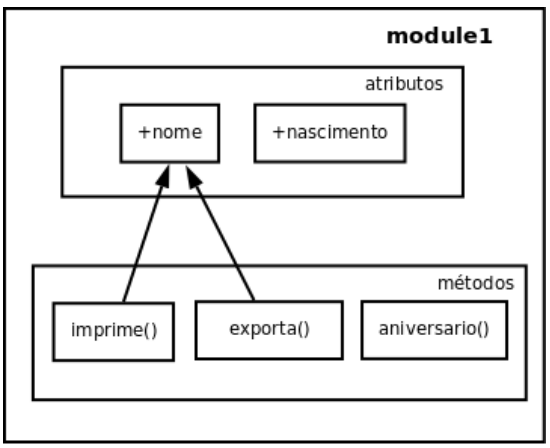


Figura 2.2: Exemplo de troca de mensagens internas de module1

LCOM mede a falta de coesão, então valores baixos significam boa coesão, e valores altos significam que métodos podem estar acoplados uns aos outros através de atributos, e que é melhor dividir o módulo em submódulos(MÄKELÄ; LEPPÄNEN, 2000).

Após a publicação de Chidamber e Kemerer (1993) para o cálculo de LCOM vários autores propuseram modificações e melhorias deste método, as mais significantes incluindo a definição original são (LAKSHMINARAYANA; S.NEWMAN, 1998):

- A definição original por Chidamber e Kemerer (1993);
- A definição de Li e Henry (1993);
- A redefinição do LCOM de Li e Henry por Hitz e Montazeri (2001); e
- A redefinição da proposta original pelos próprios Chidamber e Kemerer.

Uma das criticas ao LCOM de Chidamber e Kemerer (1993) foi feita por Gupta (1997) onde ele diz:

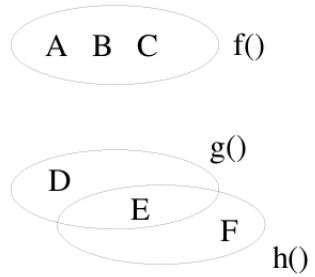
> LCOM não é um caminho válido para medir coesão de classes, sua definição é baseada em iterações entre método e dados, o que pode não ser um bom caminho para definir coesão no mundo da orientação a objetos. Além disso, para classes muito diferentes são dados valores de LCOM iguais. (GUPTA, 1997)

O cálculo de LCOM, de acordo com a definição original, é feito da seguinte forma: considere C1 uma classe com M1, M2, ..., Mn métodos e Ii o conjunto de atributos (variáveis de instância) de C1 sendo usado pelo método Mi. Para cada método Mi de C1 deve existir um grupo de atributos Ii correspondente. LCOM é número distinto de grupos formados pela intersecção dos 'n' conjuntos de Ii.

Na classe da figura 2.3 existem 2 pares de métodos que não compartilham nenhuma variável ($< f,g >$ e $< f,h >$) e exatamente um par que compartilha a variável E ($< g,h >$). Portanto, $LCOM = 2 - 1 = 1$.

Hitz e Montazeri (2001) fizeram estudos sobre LCOM e chegaram a conclusão que apesar do cálculo ser baseado numa idéia muito sensata ele traz algumas anomalias nos resultados, como exemplifica a figura 2.4, onde os autores concluem que os 2 casos são igualmente não coesivos e deveriam ter valores de LCOM iguais, mas o cálculo de LCOM traz resultados diferentes.

Eles então basearam-se na idéia de Lakshminarayana e S.Newman (1998) para propor um novo cálculo para LCOM:




```
class X {
        int A, B, C, D, E, F;
        void f () { ... uses A, B, C ... }
        void g() { ... uses D, E ... }
        void h() { ... uses E, F ... }
};
```

Figura 2.3: Exemplo: LCOM (HITZ; MONTAZERI, 2001)

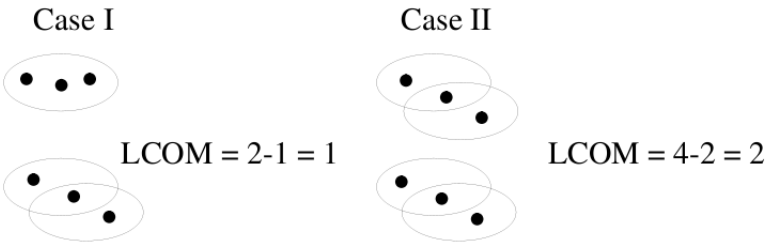


Figura 2.4: Exemplo de anomalia no uso da mátrica LCOM (HITZ; MONTAZERI, 2001)

> LCOM é o número de grupos de métodos locais distintos, onde cada grupo tem um ou mais métodos de classe locais, e ao menos 2 desses métodos acessam pelo menos um atributo de classe em comum; o número de atributos comuns pode variar de 0 à N (onde N é um inteiro maior que 0). (LAKSHMINARAYANA; S.NEWMAN, 1998)

Com esta definição, um valor $k > 1$ para $LCOM(X)$ sinaliza a possibilidade de dividir X em k módulos menores e mais coesos. Aplicando esta definição aos casos I e II da figura acima chega-se a um mesmo resultado, $LCOM = 2$, que de acordo com Hitz e Montazeri (2001) é mais coerente.

Com base nesta proposta de cálculo de Lakshminarayana e S.Newman (1998), Hitz e Montazeri (2001) propuseram uma nova definição usando representação de teoria dos grafos, como se segue:

Seja X uma classe, IX um grupo de variáveis de instancia de X, e MX um grupo

de seus métodos. Considere um simples grafo indireto $GX(V,E)$, onde:

$$V = MX$$

$$E = \{< m,n >\in V \times V \mid \exists i \in IX : (m\ acessos\ a\ i)(n\ acessos\ a\ i)\}$$

LCOM(X) é então definido como o número de componentes conectados de GX $(1 \leq LCOM(X) \leq |MX|)$.

Esta definição de Hitz e Montazeri (2001) para coesão conhecida como LCOM4 será junto a métrica de acoplamento CBO por Chidamber e Kemerer (1993) as duas métricas abordadas neste trabalho no capítulo 4.

3 IMPLEMENTAÇÃO DO EXTRATOR

Neste capítulo será apresentado como o extrator foi implementado e quais ferramentas foram utilizadas como base para a sua implementação.

3.1 EGYPT

O egypt é um Software Livre desenvolvido por Andreas Gustafsson[1] com o objetivo de gerar grafos de chamada entre funções de programas escritos em C, ele funciona lendo os arquivos intermediários gerados pelo GCC e os converte num grafo de colaboração no formato usado pelo Graphviz[2], um programa para visualização de grafos.

Em Janeiro de 2009 o egypt começou a ser restruturado por Antonio Terceiro o qual o tem mantido em [3]. As principais mudanças sofridas pelo egypt deste então foram (TERCEIRO; CHAVEZ, 2009):

- Detecção de uso de variáveis, para identificar que função usa qual variável.
- Opção para agrupar chamada e uso de variáveis por módulo. Com isto é possível ter uma visão de dependência entre módulos.
- Refatoração do script egypt em um design orientado a objetos, para permitir diferentes módulos de extração e relatório.
- Geração de relatório de métricas, como coesão e acoplamento por exemplo.

As mudanças implementadas no egypt alteraram sua estrutura original, na figura 3.1 é apresentado um fluxograma de funcionamento do egypt após a refatoração.

[1] http://www.gson.org/egypt
[2] http://www.graphviz.org
[3] http://github.com/terceiro/egypt

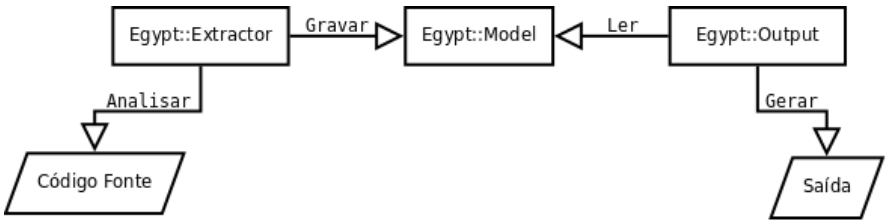


Figura 3.1: Fluxograma de funcionamento do egypt

Egypt::Extractor Extrator baseado nos arquivos intermediários do GCC

Egypt::Model Armazena os dados obtidos pelo extrator

Egypt::Output Gera saída com os dados de Egypt::Model

A atual implementação do egypt conta com apenas um tipo de saída, implementada pela classe *Egypt::Output::DOT* que gera saída no formato do Graphviz.

Para a geração de relatório de métricas o egypt conta ainda com a classe *Egypt::Metrics*.

3.2 DOXYGEN (E A SUA API)

Doxygen[4] é um sistema de documentação para C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP e C#. Com ele é possível gerar documentação em HTML, RTF, PostScript, PDF, LaTeX e man pages. Além de extrair a documentação existente no código fonte, o Doxygen também extrai informações de hierarquia e colaboração entre os módulos do projeto. É baseado nesta capacidade de extrair informações de hierarquia e colaboração que o Doxygen foi escolhido como base para implementação do extrator.

O Doxygen é Software Livre e está disponível sob a GPLv2 em [5]. Junto ao código fonte deste programa existe um pequeno exemplo chamado doxyapp que implementa uma ferramenta para análise de código fonte bem próximo as necessidades deste projeto e foi utilizado como base inicial para a implementação.

Entre as inúmeras classes presentes na API (Interface de Programação de Aplicativos) do Doxygen é importante destacar as seguintes:

Doxygen Provê um namespace para variáveis e funções globais usadas pelo doxygen

CodeOutputInterface Interface de saída de trecho de código para os parsers

[4] http://www.doxygen.org
[5] https://doxygen.svn.sourceforge.net/svnroot/doxygen/trunk

MemberDef Definição de um membro ou símbolo de classe

FileDef Definição de um arquivo

A figura 3.2 mostra uma visão geral de como os arquivos fontes são processados pelo Doxygen internamente.

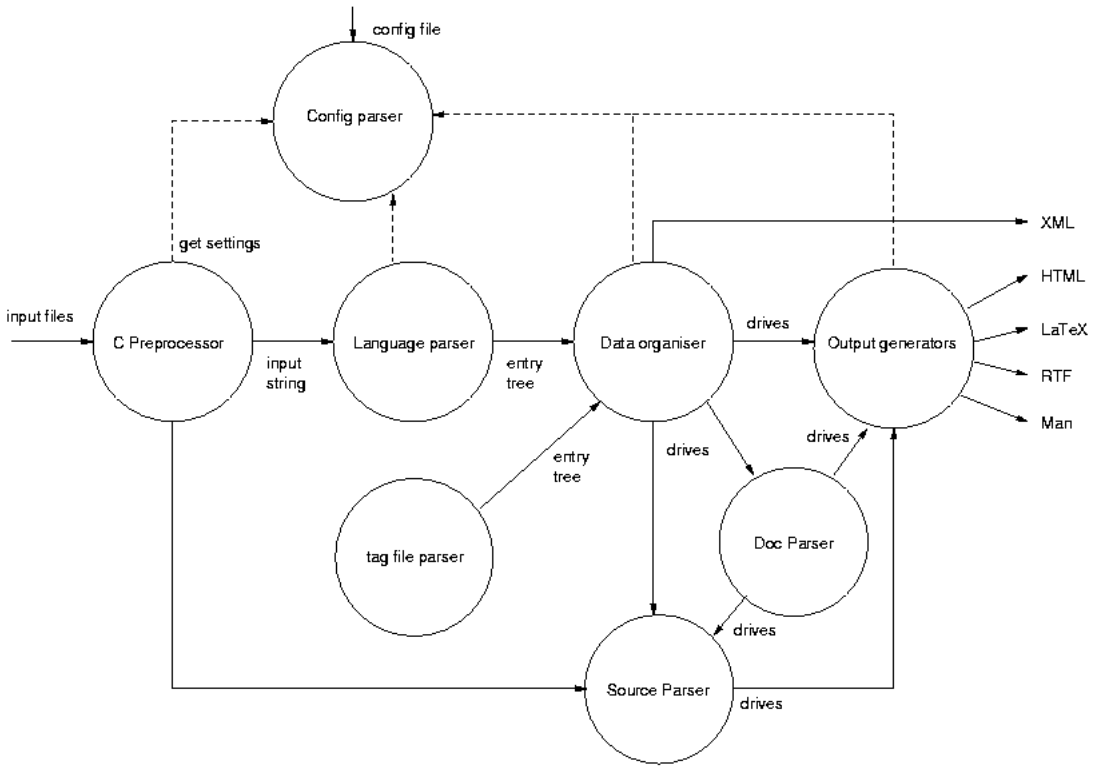


Figura 3.2: Fluxograma interno do doxygen(DOXYGEN'S...,)

3.3 IMPLEMENTAÇÃO DO EXTRATOR USANDO A API DO DOXYGEN

O Doxygen oferece todos os recursos necessários para analisar projetos C/C++ e extrair o uso de símbolos como funções e variáveis, que é o objetivo deste projeto, mas não possibilita que a saída gerada seja customizada, apenas oferece opções específicas como PDF e HTML. É necessário então adaptar o Doxygen para gerar uma saída num formato específico e apenas com as informações necessárias, para isto foi criado o doxyparse[6], um parser capaz de analisar projetos escritos em C/C++ e identificar onde os símbolos são declarados e utilizados dentro do código fonte do projeto.

[6]http://gitorious.org/doxygen/

Seguindo a interface *CodeOutputInterface* foi possível reaproveitar no doxyparse todo o poder que o Doxygen fornece e gerar a saída da forma desejada. A figura 3.3 apresenta um diagrama desta implementação.

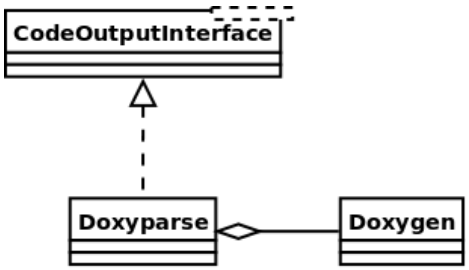


Figura 3.3: Diagrama da hierarquia de classes do doxyparse

O doxyparse reaproveita os recursos existentes no Doxygen para fazer a análise de código e gerar uma saída para o extrator a ser implementado no egypt. Ele redefine algums parâmetros de configuração do Doxygen para obter o comportamento desejado, como: analisar diretórios recursivamente, não gerar saida em LaTeXou HTML, extrair tanta informação quanto possível do código fonte, extrair informação de colaboração entre módulos e não gerar mensagens de aviso.

O doxyparse faz a análise dos fontes de um diretorio ou arquivo(s) passados como parâmetro via linha de comando, extrai destes fontes os símbolos encontrados e gera uma saída (exemplificada na figura 3.4) que será lida pelo egypt.

```
module module1.c
   function main in line 5
      uses function callback defined in module3.c
      uses function say_bye defined in module2.c
      uses function say_hello defined in module2.c
      uses variable variable defined in module3.c
```

Figura 3.4: Exemplo de saída do doxyparse

Com a saída gerada pelo doxyparse o egypt precisa agora de um extrator que entenda estes dados, extraia as informações sobre os símbolos e gere uma saída no formato do Graphviz que irá então gerar o grafo de colaboração entre os módulos.

O atual extrator egypt entende apenas código objeto gerado pelo GCC e não é capaz de entender a saída da figura 3.4 gerada pelo doxyparse. Para isto é necessário refatorar a atual implementação do egypt para possibilitar a implementação de um novo extrator. Na figura 3.5 está o desenho desta nova estrutura após a refatoração.

Abaixo uma breve descrição de cada classe presente na figura 3.5:

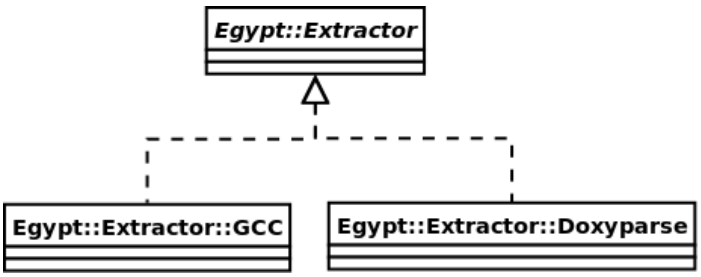


Figura 3.5: Diagrama da hierarquia de classes do extrator do egypt

Egypt::Extractor Interface padrão para os extratores.

Egypt::Extractor::GCC Extrator baseado nos arquivos intermediários do GCC.

Egypt::Extractor::Doxyparse Novo extrator baseado na saída do doxyparse.

Após a refatoração o *egypt* pode extrair informações usando 2 métodos diferentes, usando o extrator GCC que faz a extração baseada nos arquivos intermediários do GCC ou usando o extrator *Doxyparse* que faz a análise utilizando a saída do *doxyparse*.

Segue abaixo um exemplo de execução do *egypt* usando cada um dos extratores:

```
$ egypt --extractor Doxyparse <arquivos>
$ egypt --extractor GCC <arquivos>
```

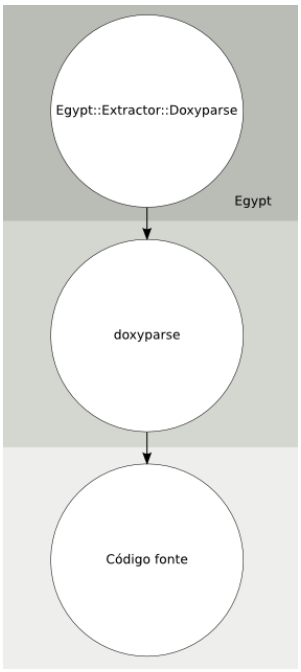


Figura 3.6: Relacionamento entre o egypt e doxyparse

A implementação do novo extrator para o *egypt* consistiu na criação de uma nova ferramenta chamada *doxyparse* e na implementação do extrator em si chamado *Egypt::Extractor::Doxyparse*. A figura 3.6 apresenta um diagrama com o relacionamento entre essas duas ferramentas.

4 AVALIAÇÃO

Com o extrator pronto é necessário avaliar se as informações extraídas estão corretas e se há diferenças em relação as informações extraídas pelo extrator original do egypt baseado em GCC.

4.1 PROCEDIMENTO

Em (TERCEIRO; CHAVEZ, 2009) foi feita uma análise, utilizando o extrator original do egypt, do projeto *ristretto*[1], um Software Livre escrito em C para visualização de imagens no ambiente Desktop Xfce[2]. As informações geradas por esta análise serão utilizadas aqui para comparação com as informações extraídas pelo novo extrator baseado no doxyparse.

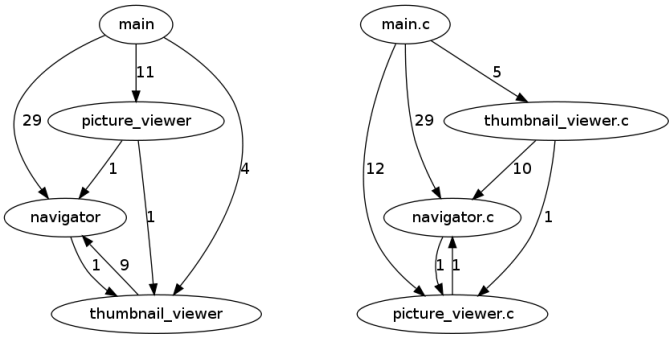
Além da comparação entre o extrator original e o novo extrator do egypt a própria saída gerada pelo doxyparse será avaliada em busca de inconsistências em relação a identificação dos símbolos encontrados no código fonte.

Por fim será feito resumo comparativo com as métricas de coesão e acomplamento calculadas pelo egypt usando cada um dos extratores com o objetivo de interpretar as possíveis diferenças nos números encontrados por cada um.

4.2 RESULTADOS

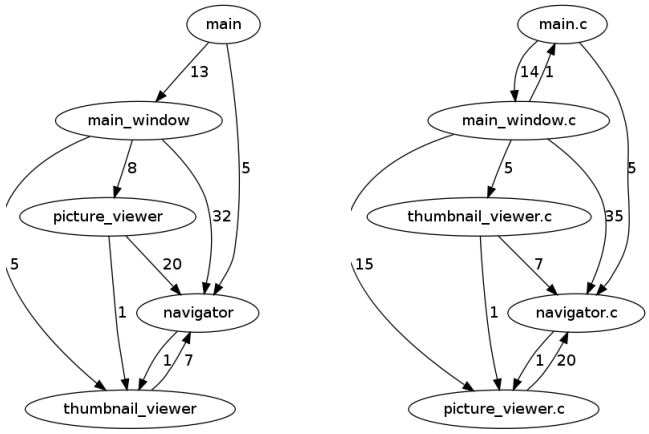
As diferenças encontradas entre os grafos de colaboração e entre as métricas de coesão e acoplamento serão apresentadas nas seções a seguir.

[1] http://goodies.xfce.org/projects/applications/ristretto
[2] http://www.xfce.org



(a) Doxyparse (b) GCC

Figura 4.1: Grafo de colaboração do ristretto 0.0.1



(a) Doxyparse (b) GCC

Figura 4.2: Grafo de colaboração do ristretto 0.0.11

4.2.1 GRAFOS DE COLABORAÇÃO

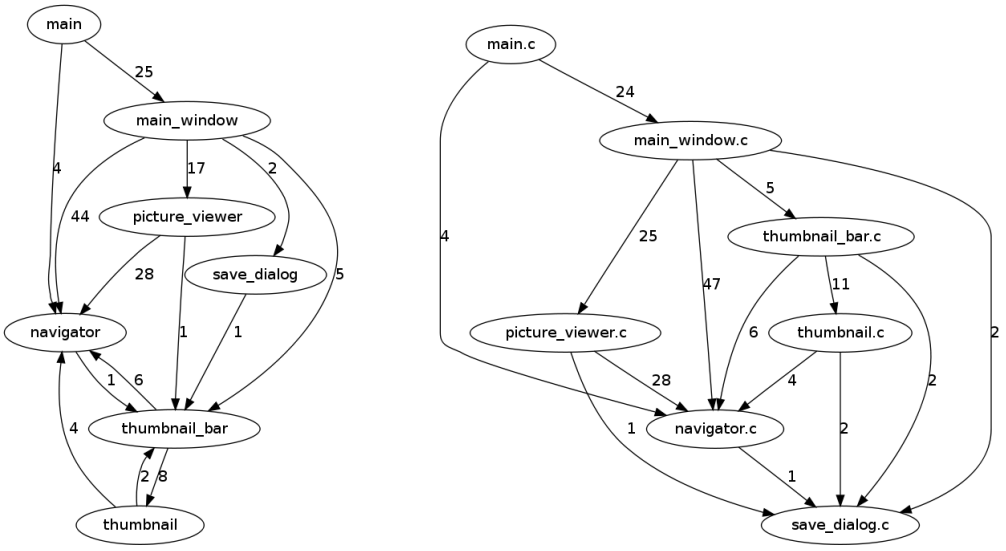
Os grafos gerados pelo egypt usando o extrator Doxyparse apresentaram diferenças em relação ao extrator GCC como demonstram as figuras 4.1, 4.2 e 4.3 referentes as versões 0.0.1, 0.0.11 e 0.0.21 do *ristretto*, respectivamente.

Avaliando cada um dos grafos encontramos as seguintes diferenças:

- Na figura 4.1(b) o extrator GCC encontra uma chamada do módulo *thumbnail_viewer* para o módulo *picture_viewer*, já na figura 4.1(a) o Doxyparse encontra uma chamada de *picture_viewer* para *thumbnail_viewer*.

- Na figura 4.2(b) o GCC encontra uma chamada do módulo *main_window* para o módulo *main*, já no Doxyparse não há nenhuma chamada para o módulo *main* como visto na figura 4.2(a).

- Na figura 4.3 temos outra diferença interessante, o GCC informa que o módulo *save_dialog*



(a) Doxyparse (b) GCC

Figura 4.3: Grafo de colaboração do ristretto 0.0.21

é chamado pelos módulos *main_window*, *thumbnail*, *navigator*, *picture_viewer* e *thumbnail_bar*, já o Doxyparse na figura4.3(a) mostra apenas uma chamada de *main_window* para *save_dialog*.

4.2.2 MÉTRICAS

O Doxyparse também apresentou diferenças em relação ao GCC no cálculo das métricas, na tabela 4.1 é apresentado um resumo comparativo entre as métricas de coesão e acoplamento gerados por cada extrator para cada versão do *ristretto*.

Nesta tabela os valores de *Falta de Coesão* e *Acoplamento* referem-se a média das métricas LCOM4 e CBO respectivamente.

Analisando a tabela é possível notar que a média dos valores de *Falta de Coesão* gerados pelo Doxyparse são maiores que os valores gerados pelo GCC em todas as versões do *ristretto*. E que o *Acoplamento* é menor em todas as versões.

4.3 DISCUSSÃO

A análise dos dados gerados pelo egypt usando cada um dos extratores para cada versão do *ristretto* foi importante pois revelou que existem divergencias entre os extratores. Isto indica que algum dos extratores está dando informações incorretas e é necessário investigar o que está causando essas diferenças. Nas seções a seguir cada uma das diferenças encontradas serão

Tabela 4.1: Resumo comparativo das métricas do ristretto

Extrator	GCC		Doxyparse	
Versão	Falta de Coesão	Acoplamento	Falta de Coesão	Acoplamento
0.0.1	4.75	2.75	5.00	1.25
0.0.2	5.75	2.75	6.00	1.25
0.0.3	6.00	2.75	6.00	1.25
0.0.4	6.25	2.75	6.25	1.25
0.0.5	6.25	2.75	6.25	1.25
0.0.6	7.60	3.00	7.60	1.40
0.0.7	7.60	3.00	7.60	1.40
0.0.8	7.00	3.00	7.20	1.40
0.0.9	7.20	3.00	7.40	1.40
0.0.10	7.60	3.00	8.00	1.40
0.0.11	7.60	3.00	8.00	1.40
0.0.12	7.60	3.00	8.00	1.40
0.0.13	7.80	3.00	8.20	1.40
0.0.14	8.00	3.00	8.40	1.40
0.0.15	8.00	3.00	8.80	1.40
0.0.16	7.16	3.16	7.83	1.50
0.0.17	7.00	3.16	7.83	1.50
0.0.18	7.50	3.00	8.33	1.50
0.0.19	7.83	3.00	8.66	1.50
0.0.20	8.83	3.00	10.00	1.50
0.0.21	8.28	3.00	9.14	1.42

avaliadas e corrigidas quando necessário.

4.3.1 GRAFOS DE COLABORAÇÃO

Ao verificar o código fonte do *ristretto 0.0.1* percebeu-se que nenhum dos 2 módulos (*picture_viewer* e *thumbnail_viewer*) se referenciam, então os dois extratores estão dando informações incorretas e os grafos demonstrados na figura 4.1 estão incorretos.

O problema pode estar na interpretação dos dados pelo egypt ou nos dados gerados pelo doxyparse. Após uma verificação a saída do doxyparse, exemplo abaixo na figura 4.4, foi possível notar que os dados estão corretos.

```
module main.c
   variable mime_dbase in line 28
   variable window_fullscreen in line 86
   variable viewer_scale in line 87
   variable menu_bar in line 88
   variable image_tool_bar in line 89
   variable app_tool_bar in line 90
   variable status_bar in line 91
   variable playing in line 92
   variable menu_item_play in line 93
   variable menu_item_pause in line 94
   variable main_hbox in line 96
   variable main_vbox1 in line 97
   variable thumbnail_viewer in line 98
   variable recent_manager in line 99
   variable xfce_rc in line 100
   variable thumbnail_viewer_orientation in line 101
   function cb_rstto_zoom_fit in line 31
      uses function rstto_picture_viewer_fit_scale defined in picture_viewer.c
   function cb_rstto_zoom_100 in line 33
      uses function rstto_picture_viewer_set_scale defined in picture_viewer.c
   function cb_rstto_zoom_in in line 35
      uses function rstto_picture_viewer_get_scale defined in picture_viewer.c
      uses function rstto_picture_viewer_set_scale defined in picture_viewer.c
   function cb_rstto_zoom_out in line 37
      uses function rstto_picture_viewer_get_scale defined in picture_viewer.c
      uses function rstto_picture_viewer_set_scale defined in picture_viewer.c
```

Figura 4.4: Trecho de saída do doxyparse para o ristretto 0.0.1

Então o problema está na interpretação do egypt aos dados gerados pelo doxyparse.

O egypt armazena as informações usando como chave, ou seja, como identificador único o nome do símbolo em questão mas alguns módulos possuem símbolos com o mesmo nome, isto faz com que o egypt confunda o uso e chamada destes símbolos.

A solução adotada para este problema foi armazenar o nome do módulo junto ao nome do símbolo, por exemplo ao invés de guardar apenas *parent_class* guarda-se *main::parent_class*. A figura 4.5 apresenta o grafo do *ristretto 0.0.1* atualizado após esta correção.

Na figura 4.2 referente ao *ristretto 0.0.11* o GCC diz que o módulo *main_window* chama *main*, já no Doxyparse não há esta chamada. E de fato, analisando o código fonte do *ristretto*

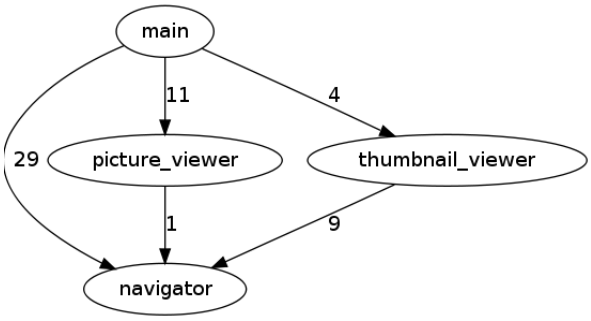


Figura 4.5: Grafo de colaboração do ristretto 0.0.1 atualizado

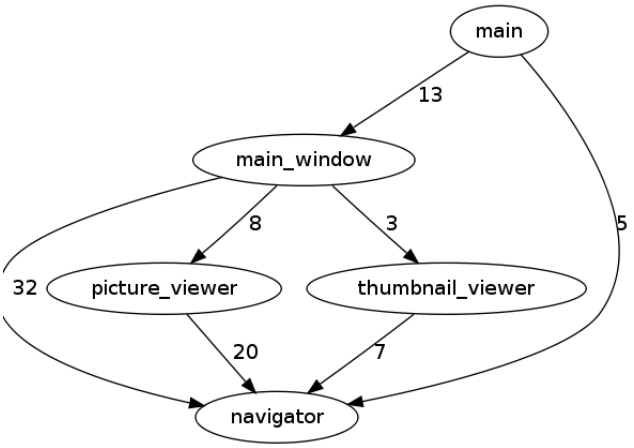


Figura 4.6: Grafo de colaboração do ristretto 0.0.11 atualizado

0.0.11 não foi encontrada nenhuma chamada de *main_window* para *main*, então o Doxyparse está correto neste caso. Apesar disso o grafo da figura 4.2(a) está errado pois foi gerado antes da correção do problema citado no parágrafo anterior, a figura 4.6 apresenta um grafo atualizado para o *ristretto 0.0.11*.

Na figura 4.3(b) o GCC diz que o módulo *save_dialog* é referenciado por vários outros módulos mas no código fonte do *ristretto 0.0.21* não foi encontrada todas essas chamadas, então o Doxyparse está correto ao identificar que o *save_dialog* é chamado apenas por *main_window*. Mas a chamada de *save_dialog* para *thumbnail_bar* dado pelo Doxygen está incorreta, não há esta chamada nos fontes do projeto. Este problema foi corrigido pela mesma solução dos problemas citados nos parágrafos anteriores, a figura 4.7 mostra o grafo do *ristretto 0.0.21* atualizado.

Além destas correções aplicadas ao egypt foi necessário corrigir um problema no doxyparse referente a identificação de símbolos estáticos, neste estudo de caso o doxyparse apresentou inconsistências em relação ao uso do simbolo *parent_class*, uma variável estática definida por todos os módulos do *ristretto*. O doxyparse registra a declaração deste símbolo apenas na primeira ocorrência dele, e nas demais ocorrências de uso e chamada nos diversos módulos o doxyparse faz referência ao primeiro módulo onde o símbolo foi encontrado.

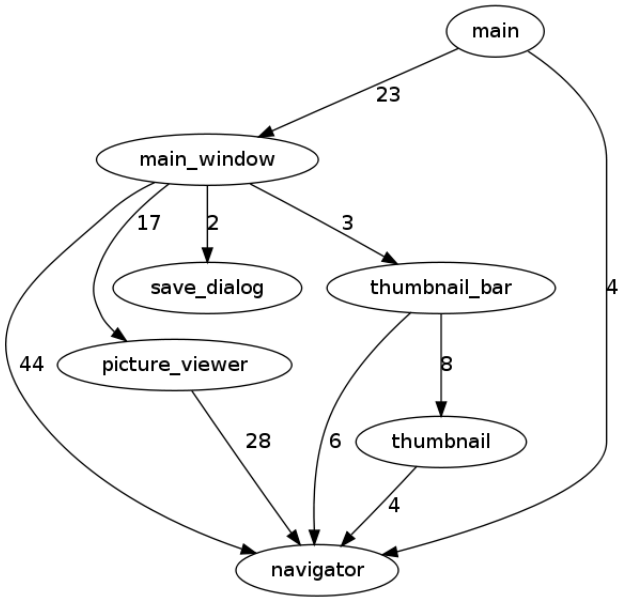


Figura 4.7: Grafo de colaboração do ristretto 0.0.21 atualizado

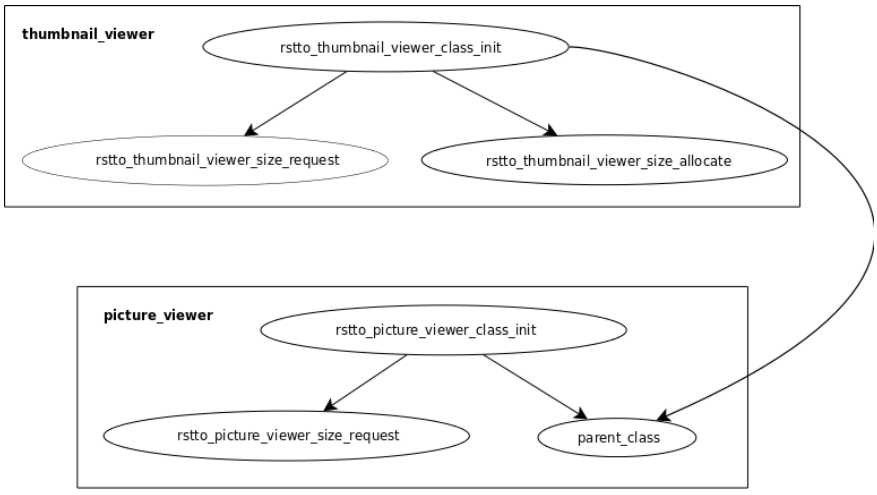


Figura 4.8: Grafo de colaboração de exemplo do ristretto

Tabela 4.2: Métricas do ristretto 0.0.1

Extrator	GCC		Doxyparse	
Módulo	Falta de Coesão	Acoplamento	Falta de Coesão	Acoplamento
main	1	4	1	3
navigator	13	2	14	0
picture_viewer	2	2	2	1
thumbnail_viewer	3	3	3	1

Este problema é exemplificado na figura 4.8, onde 2 módulos definem o símbolo *parent_class* e o utilizam internamente, mas o doxyparse o registra em apenas um módulo (*picture_viewer*) e referencia este módulo em todos os outros (*thumbnail_viewer*) que utilizam o símbolo. Isto foi solucionado ignorando as chamadas aos símbolos estáticos fora do módulo sendo analisado, só se considera chamada a símbolos estáticos se eles estiverem no mesmo módulo que o símbolo.

4.3.2 MÉTRICAS

Analisando a tabela 4.1 é possível notar que o acoplamento calculado com base no Doxyparse se mantém menor do que o cálculo baseado no GCC em todas as versões do *ristretto*, já a métrica de falta de coesão se mantém ligeiramente maior também em todas as versões.

Na tabela 4.2 são apresentadas métricas para cada módulo do *ristretto 0.0.1*. Esta versão será utilizada para verificar as diferenças encontradas entre os dois extratores.

A diferença no acoplamento do módulo *navigator* entre o Doxyparse e o GCC é claramente verificado pela figura 4.5 onde verifica-se que este módulo não chama nenhum outro módulo, portanto tem acoplamento igual a 0 (zero).

O GCC diz que o módulo *main* tem acoplamento igual a 4, mas o grafo de colaboração do ristretto 0.0.1 na figura 4.1(b) não apresenta colaboração com 4 módulos, mas sim com 3 e de acordo com o cálculo, o acoplamento de um dado módulo é o número de outros módulos com os quais ele colabora, esta informação está incorreta.

Avaliando o módulo do egypt responsável pelo cálculo de métricas verificou-se que o cálculo de acoplamento estava incorreto, pois considerava métodos externos como chamadas a métodos da biblioteca GTK (GIMP Tool Kit). O Doxyparse não foi afetado por este erro pois a chamada a métodos externos são desconsiderados pelo doxyparse.

O egypt foi alterado para considerar apenas métodos definidos no escopo do projeto e o valor de acoplamento para o módulo *main* do *ristretto 0.0.1* ficou correto. Esta correção corrigiu

também as diferenças para acoplamento dos módulos *picture_viewer* e *thumbnail_viewer*.

Com esta correção as médias de acoplamento obtidas pelo extrator GCC em todas as versões do *ristretto* diminuiram, mas ainda não se igualaram aos valores obtidos pelo Doxyparse. Mas como o foco deste trabalho é no extrator Doxyparse a investigação detalhada deste problema foge ao escopo do trabalho. Na tabela B.1 no apêndice B é apresentado um resumo comparativo das métricas atualizadas.

5 CONCLUSÃO

Os resultados obtidos foram satisfatórios e atingiram o objetivo inicial que foi possibilitar extração de informação de dependência entre módulos de programas C/C++ sem necessidade de compilar o software em questão, além de um novo extrator o *egypt* ganhou correção de erro no cálculo da métrica CBO.

Além dos resultados práticos obtidos com a implementação, o estudo de caso possibilitou ter uma visão geral da evolução do projeto *ristretto* e de seus níveis de acoplamento e coesão ao longo das versões do projeto.

No apêndice A é possível verificar a evolução da arquitetura do ristretto em cada versão, é possível notar que em alguns momentos a arquitetura do projeto sofre mudanças estruturais pelo surgimento de novos módulos, e em outros momentos entre uma versão a outra permanece inalterada apenas com mudanças internas nos módulos onde o que se altera é apenas o número de colaboração entre os módulos existentes.

5.1 TRABALHOS FUTUROS

Testar egypt em outras linguagens de programação. O Doxygen suporta várias linguagens de programação além de C/C++. Testar o extrator Doxyparse em projetos escritos nessas linguagens e verificar se o egypt e o doxyparse funcionam bem.

Implementar novo extrator baseado em Natural Docs. O Natural Docs[1] é uma ferramenta semelhante ao Doxygen implementada em Perl com suporte a algumas linguagens não suportadas pelo Doxygen como Perl, Ruby e ActionScript.

Implementar no doxyparse detecção de chamada indireta. O doxyparse identifica chamadas indiretas mas não as diferencia de chamadas diretas.

Armazenar símbolos externos ao projeto. Os símbolos externos encontrados pelo doxyparse

[1] http://www.naturaldocs.org/

são completamente descartados, é importante ter estas referencias para calcular outros tipos de métricas.

REFERÊNCIAS BIBLIOGRÁFICAS

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. 1993.

DOXYGEN'S Internals. [S.l.]. Acessado em 02 de Junho de 2009. Disponível em: <http://www.stack.nl/ dimitri/doxygen/arch.html>.

GUPTA, B. S. *A Critique of Cohesion Measures in the Object-Oriented Paradigm - Gupta*. Dissertação (Mestrado) — Michigan Technological University, March 1997. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.3710&rep=rep1&type=pdf>.

HASSAN, A. E.; JIANG, Z. M.; HOLT, R. C. Source versus object code extraction for recovering software architecture. In: *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE'05)*. [S.l.: s.n.], 2005.

HITZ, M.; MONTAZERI, B. Measuring coupling and cohesion in object-oriented systems. In: *Symposium on Applied Corporate Computing*. Monterrey, Mexico: [s.n.], 2001. Disponível em: <http://www.isys.uni-klu.ac.at/PDF/1995-0043-MHBM.pdf>.

LAKSHMINARAYANA, A.; S.NEWMAN, T. *Principal Component Analysis of LCOM metrics - Anuradha and Timothy*. [S.l.], 1998. Disponível em: <http://www.cs.uah.edu/tech-reports/TR-UAH-CS-1999-01.pdf>.

LI, W.; HENRY, S. Maintenance metrics for the object oriented paradigm. In: . [s.n.], 1993. p. 52–60. Disponível em: <http://dx.doi.org/10.1109/METRIC.1993.263801>.

MÄKELÄ, S.; LEPPÄNEN, V. Observations on lack of cohesion metrics - sami and ville. In: *International Conference on Computer Systems and Technologies - CompSysTech'06*. [S.l.: s.n.], 2000.

PRESSMAN, R. S. *Engenharia de Software*. sexta. [S.l.]: Makron Books, 2005.

SANT'ANNA, C. N. *On the Modularity of Aspect-Oriented Design: A Concern-Driven Measurement Approach*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, April 2008.

TERCEIRO, A.; CHAVEZ, C. Structural complexity evolution in free software projects: A case study. 2009.

TERCEIRO, A. A. S. *Verificando as Leis de Crescimento Contínuo e Complexidade Crescente em um projeto de software livre de pequeno porte*. Salvador-BA-Brasil, 2008. [Online; acessado 15 de Janeiro de 2009]. Disponível em: <http://disciplinas.dcc.ufba.br/pub/MATA26/TrabalhoTerceiro/mata26-terceiro-projeto-piloto.pdf>.

APÊNDICE A – EVOLUÇÃO DO RISTRETTO

Grafos de colaboração do ristretto da versão 0.0.1 até a versão 0.0.21 gerados com os dados extraidos pelo Doxyparse.

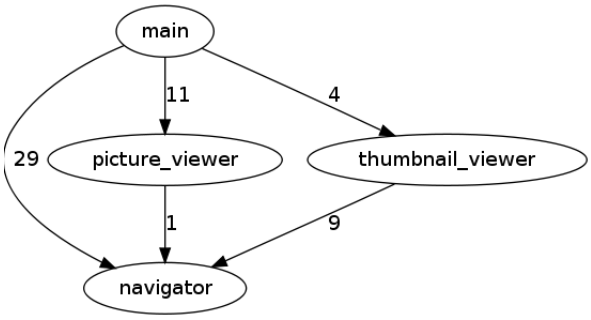


Figura A.1: ristretto 0.0.1

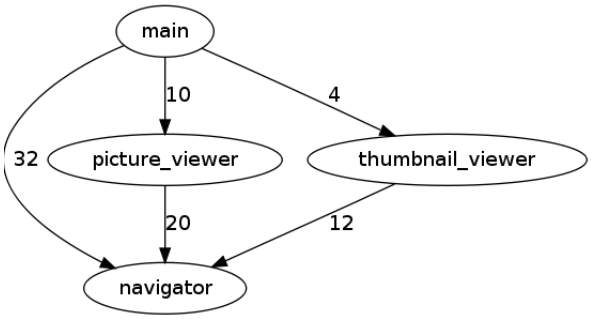


Figura A.2: ristretto 0.0.2

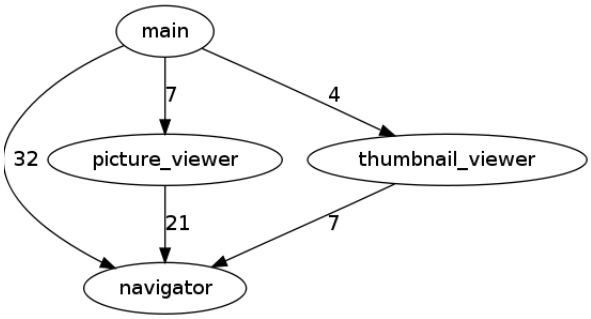


Figura A.3: ristretto 0.0.3

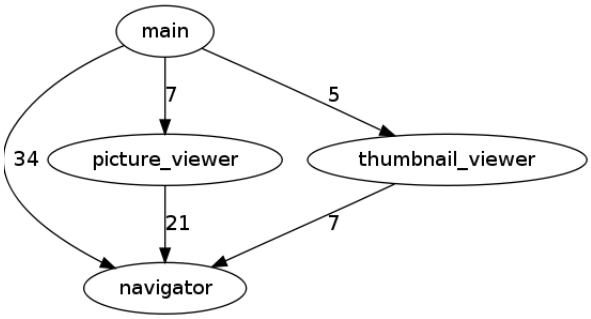


Figura A.4: ristretto 0.0.4

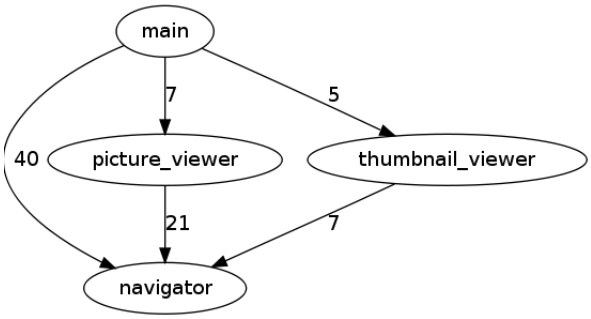


Figura A.5: ristretto 0.0.5

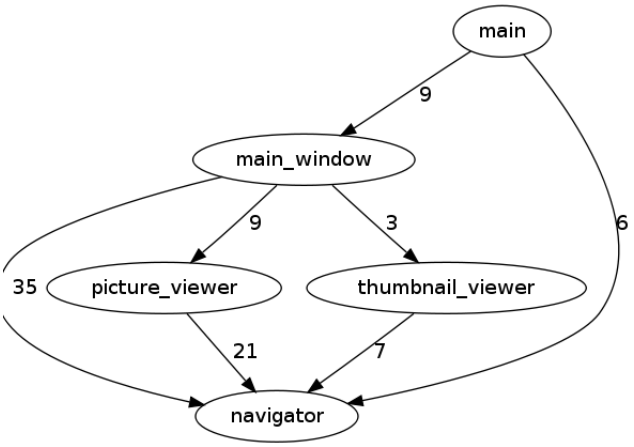


Figura A.6: ristretto 0.0.6

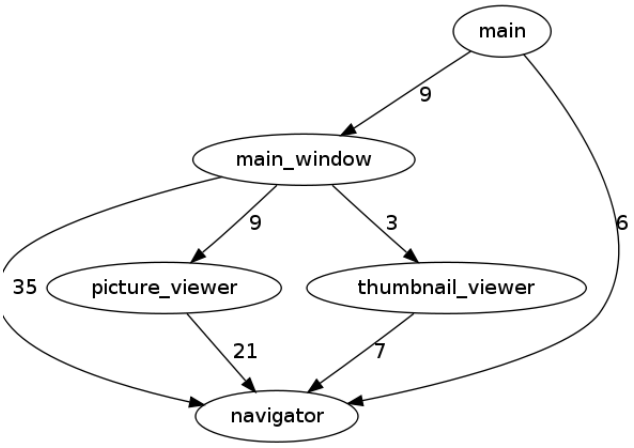


Figura A.7: ristretto 0.0.7

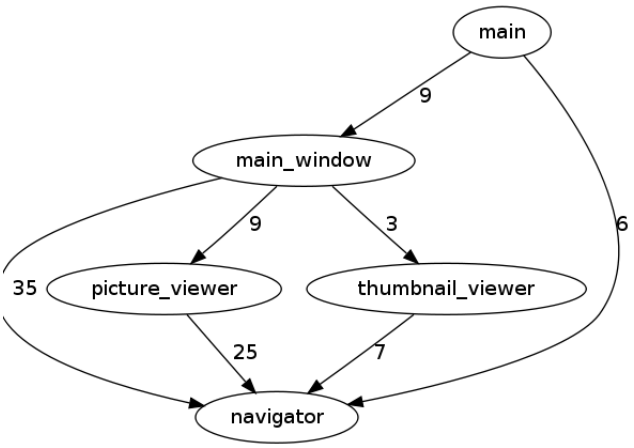


Figura A.8: ristretto 0.0.8

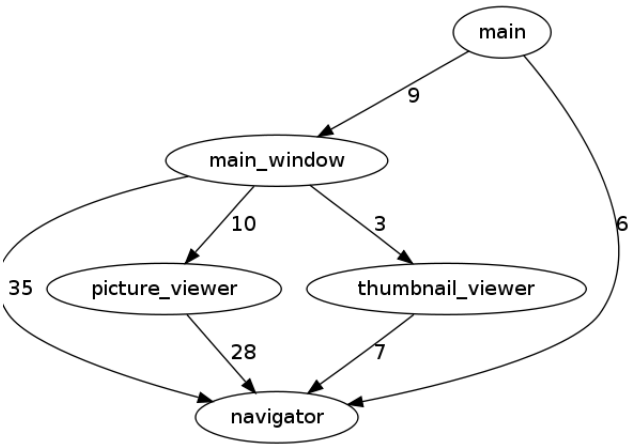


Figura A.9: ristretto 0.0.9

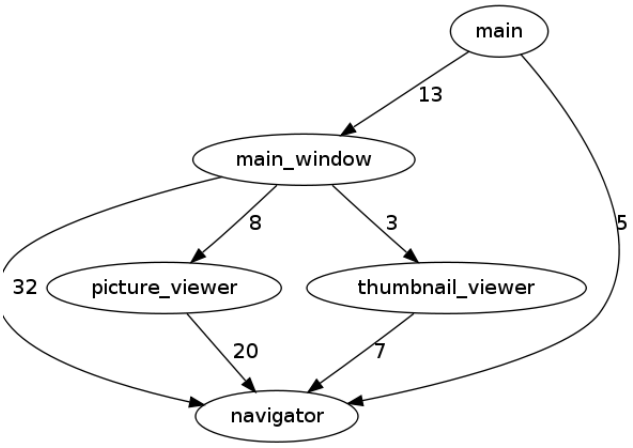


Figura A.10: ristretto 0.0.10

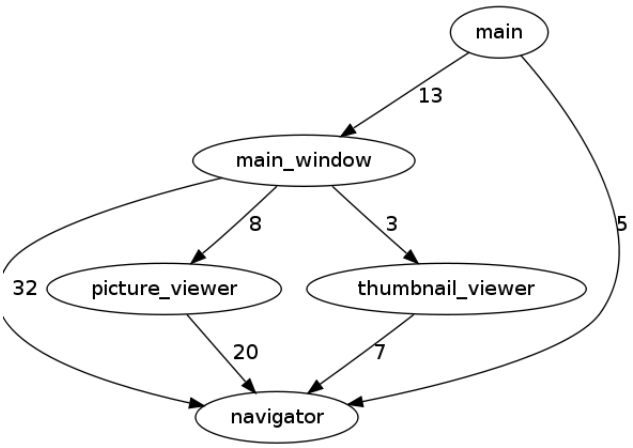


Figura A.11: ristretto 0.0.11

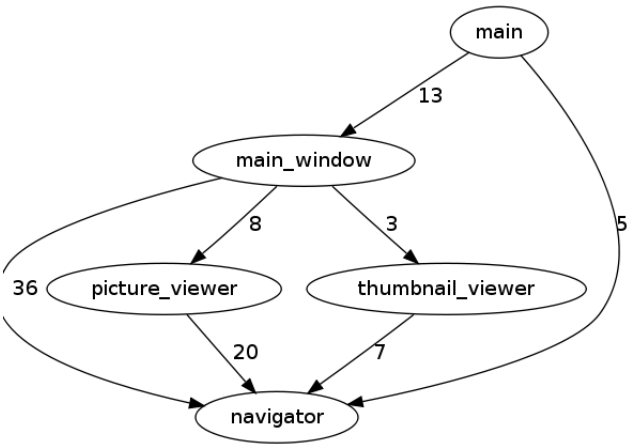


Figura A.12: ristretto 0.0.12

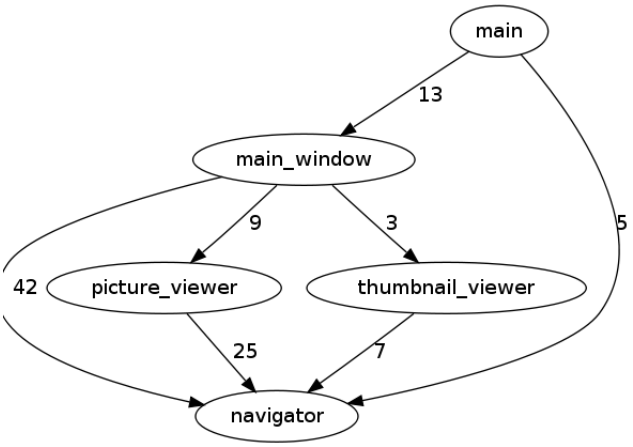


Figura A.13: ristretto 0.0.13

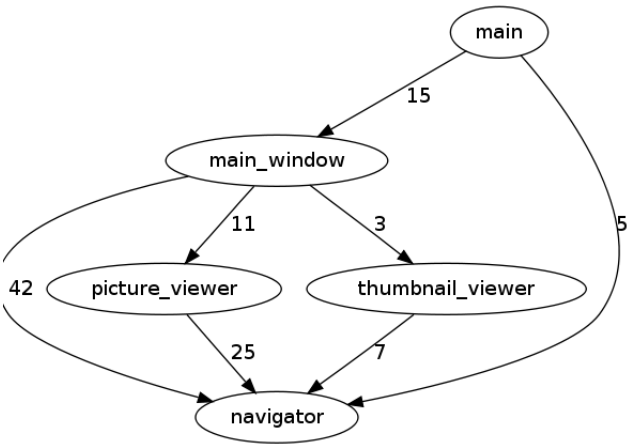


Figura A.14: ristretto 0.0.14

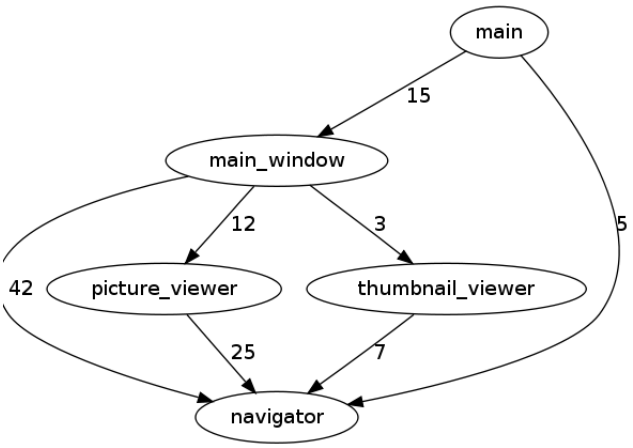


Figura A.15: ristretto 0.0.15

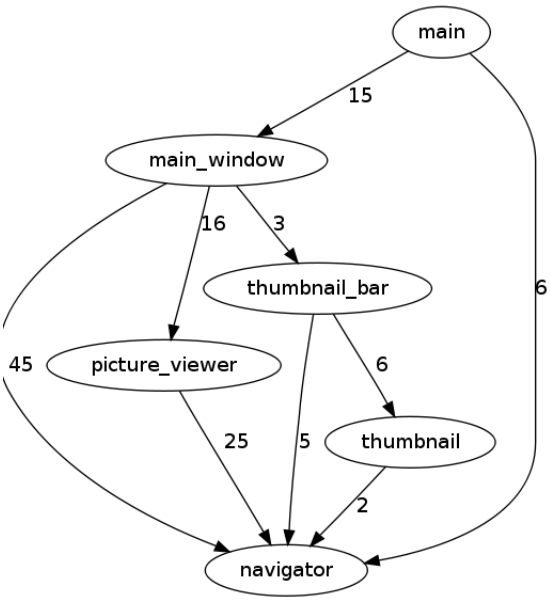


Figura A.16: ristretto 0.0.16

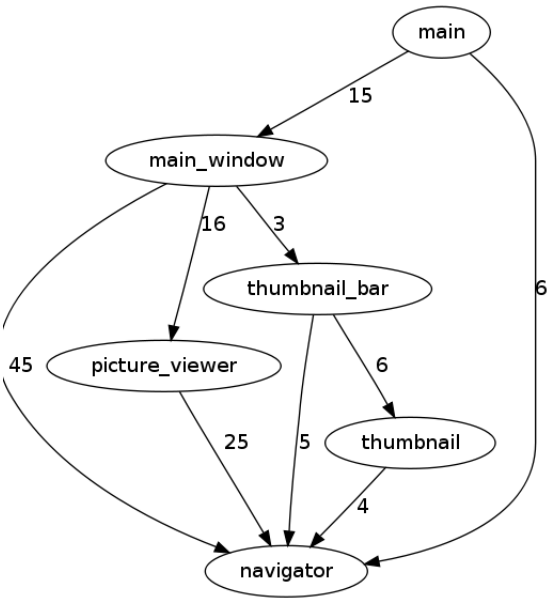


Figura A.17: ristretto 0.0.17

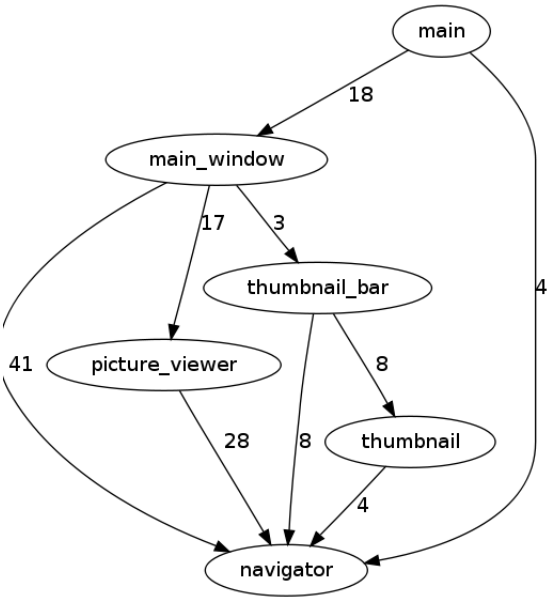


Figura A.18: ristretto 0.0.18

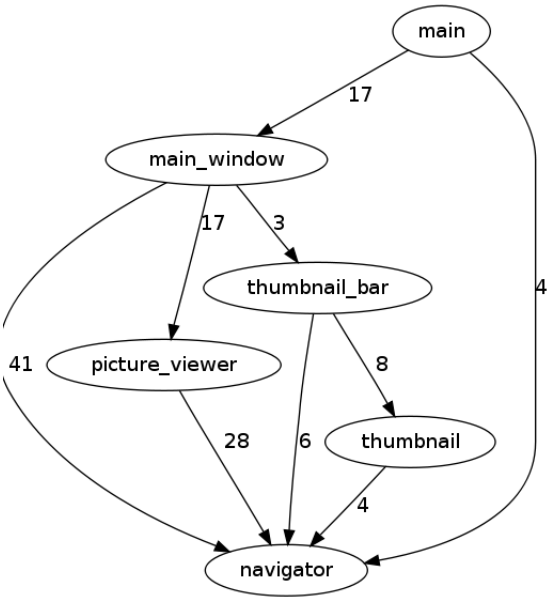


Figura A.19: ristretto 0.0.19

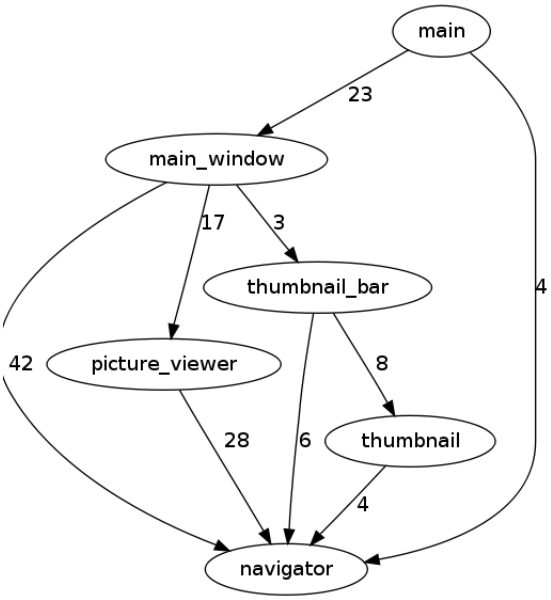


Figura A.20: ristretto 0.0.20

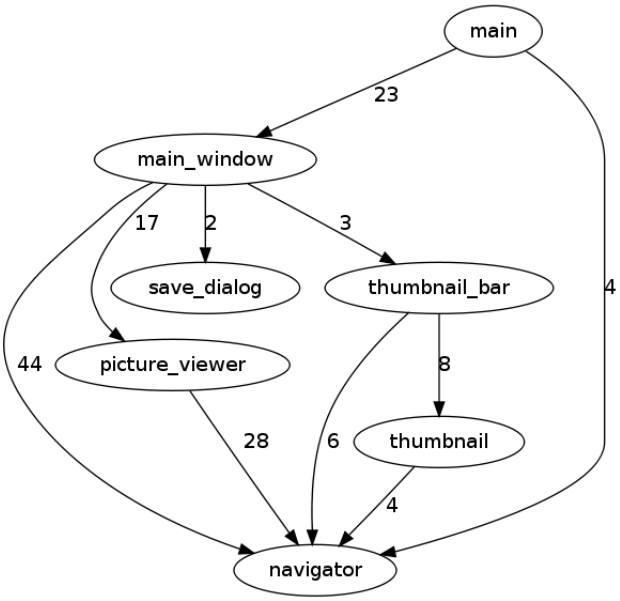


Figura A.21: ristretto 0.0.21

APÊNDICE B – RESUMO DE MÉTRICAS DO RISTRETTO

Tabela B.1: Resumo comparativo das métricas do ristretto atualizado

Extrator	GCC		Doxyparse	
Versão	Falta de Coesão	Acoplamento	Falta de Coesão	Acoplamento
0.0.1	4.75	1.25	5.00	1.25
0.0.2	5.75	1.25	6.00	1.25
0.0.3	6.00	1.25	6.00	1.25
0.0.4	6.25	1.25	6.25	1.25
0.0.5	6.25	1.25	6.25	1.25
0.0.6	7.60	2.20	7.60	1.40
0.0.7	7.60	2.20	7.60	1.40
0.0.8	7.00	2.20	7.20	1.40
0.0.9	7.20	2.20	7.40	1.40
0.0.10	7.60	2.20	8.00	1.40
0.0.11	7.60	2.20	8.00	1.40
0.0.12	7.60	2.20	8.00	1.40
0.0.13	7.80	2.20	8.20	1.40
0.0.14	8.00	2.20	8.40	1.40
0.0.15	8.00	2.20	8.80	1.40
0.0.16	7.16	2.33	7.83	1.50
0.0.17	7.00	2.33	7.83	1.50
0.0.18	7.50	2.16	8.33	1.50
0.0.19	7.83	2.16	8.66	1.50
0.0.20	8.83	2.16	10.00	1.50
0.0.21	8.28	2.00	9.14	1.42