UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

EDUARDO VINICIUS KEMPF

I.CODE.READER: UMA FERRAMENTA DE EXTRAÇÃO E GERENCIAMENTO DA ARQUITETURA DE SOFTWARE

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO - PR

2015

EDUARDO VINICIUS KEMPF

I.CODE.READER: UMA FERRAMENTA DE EXTRAÇÃO E GERENCIAMENTO DA ARQUITETURA DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Tecnólogo em Tecnologia em Sistemas para Internet.

Orientador: Me. Filipe Roseiro Côgo

Coorientador: Me. Igor Scaliante Wiese

CAMPO MOURÃO - PR

2015

RESUMO

KEMPF, Eduardo Vinicius. I.CODE.READER: UMA FERRAMENTA DE EXTRAÇÃO E GERENCIAMENTO DA ARQUITETURA DE SOFTWARE. 31 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2015.

Compreender a estrutura de um software desconhecido não é uma tarefa simples de ser feita. Com o intuito de auxiliar na compreensão da arquitetura de um software, ferramentas que são capazes de analisar o código fonte e moldar a estrutura do sistema vem sendo desenvolvidas com o passar do tempo. Porém muitas vezes essas ferramentas são inflexíveis, e não fornecerem meios de gerenciar a estrutura gerada, obrigando o usuário a ficar preso a elas ou a utilizar ferramentas complementares. Neste trabalho foi desenvolvida uma ferramenta que é capaz de extrair e modificar a estrutura diretamente do código fonte. Deste modo, criar espaço para que pesquisadores possam estudar arquiteturas de diversos softwares; facilitando o entendimento do funcionamento do sistema; além de prover a descoberta de possíveis modificações ao sistema.

Palavras-chave: Arquitetura de software, Extração, Gerenciamento

ABSTRACT

KEMPF, Eduardo Vinicius. . 31 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2015.

Understand the structure of an unknown software is never an easy task to be done. With the help of purpose in understanding the architecture of a software tools that are able to analyze the source code and shape the system structure has been developed over time. But often these tools are inflexible in addition to not provide ways to manage the generated structure, forcing the user to get stuck to them or to use additional tools. This work was developed a tool that is able to extract the structure directly from the source code, plus the user can make changes in the structure from the tool. Thereby paving the way for researchers to study various software architectures; facilitating the understanding of system operation; besides providing the discovery of possible modifications to the system.

Keywords: Software architecture, Extraction, Management

LISTA DE FIGURAS

FIGURA 1 – Exemplo de representação feita pela proposta de Ramírez et al. (2015) ... 7
FIGURA 2 – Exemplo de arquitetura gerada pelo SArF Map 8
FIGURA 3 – Exemplo de arquitetura gerada pelo SAVE 9
FIGURA 4 – Comunicação entre os módulos desenvolvidos no sistema 12
FIGURA 5 – Diagrama resumido das classes do módulo 13
FIGURA 6 – Diagrama resumido das classes do módulo de criação da estrutura 15
FIGURA 7 – Diagrama resumido das classes do módulo de criação da estrutura 17
FIGURA 8 – Diagrama do código fonte que será utilizado no exemplo 21
FIGURA 9 – Informando o caminho do código fonte ao sistema 22
FIGURA 10 – Arquitetura Gerada pelo `i.code.reader` para o código fonte de exemplo 22
FIGURA 11 – Código para realizar uma separação de componentes 24
FIGURA 12 – Arquitetura gerada após a separação dos dois componentes 24
FIGURA 13 – Código responsável por realizar a união de dois componentes 25
FIGURA 14 – Arquitetura gerada ao unir os componentes 26
FIGURA 15 – Código responsável por criar um novo componente, mover classes a um outro componente e remover um componente 26
FIGURA 16 – Arquitetura gerada após a excução do código 27

SUMÁRIO

1 INTRODUÇÃO 1
2 REFERENCIAL TEÓRICO 4
2.1 ARQUITETURA DE SOFTWARE 4
2.2 ENGENHARIA REVERSA 5
2.3 REPRESENTAÇÃO DA ARQUITETURA DO SOFTWARE 6
2.3.1 o meta-modelo 6
2.4 TRABALHOS RELACIONADOS 7
2.5 TECNOLOGIAS UTILIZADAS 9
3 A FERRAMENTA: I.CODE.READER 11
3.1 APRESENTAÇÃO SISTEMA DE EXTRAÇÃO DE ESTRUTURAS 11
3.2 A ARQUITETURA DA FERRAMENTA 12
3.2.1 Módulo de coleta de dados do código fonte 13
3.2.2 Módulo de construção da estrutura do software 15
3.2.3 Módulo de gerenciamento da estrutura do software 17
3.3 FUNCIONAMENTO DA FERRAMENTA 18
3.4 EXEMPLO DE USO DA FERRAMENTA 20
4 CONCLUSÕES GERAIS 28
REFERÊNCIAS 30

1 INTRODUÇÃO

O estudo formal da arquitetura de software tem sido uma ferramenta poderosa para a engenharia de software identificar possíveis problemas estruturais presentes em sistemas de computação. Com o passar dos anos este processo vem ganhando cada vez mais importância, uma vez que além de auxiliar a identificar problemas de desenvolvimento do software, ele também é capaz de ajudar a novos desenvolvedores a entender o funcionamento do sistema de uma forma mais eficaz do que analisar o código manualmente (SHAW; GARLAN, 1996; BASS et al., 2012).

Atualmente, diversos pesquisadores também se utilizam das arquiteturas de software, como base para analisar se as métricas do software estão de acordo com as normas recomendadas para o desenvolvimento de software. Para se obter determinadas métricas do sistema é necessário que se conheça a arquitetura geral do projeto.

No entanto, os benefícios que este estudo é capaz de trazer para o desenvolvimento de software, em diversas vezes é deixado de lado, uma vez que muitos sistemas não possuem a estrutura devidamente documentada, ou a estrutura que está documentada não é clara, ou até mesmo as manutenções e modificações que o sistema passa durante seus ciclos de vida não são repassadas ao documento estrutural. Desse modo a estrutura documentada não representa a real situação do software fazendo com que os desenvolvedores fiquem confusos em relação a real situação do software (KNODEL et al., 2006).

O problema relacionado a documentação fica evidente quando os sistemas necessitam de manutenção. Neste caso desenvolvedores e pesquisadores que não possuem o conhecimento do software a ser analisado são obrigados a procurar outras formas de compreender o software rapidamente. Uma das possíveis soluções é entrar em contato com um especialista do software, porém este método pode falhar, uma vez que o especialista pode estar indisponível para contato. Assim para evitar o esforço de aprender o funcionamento do software manualmente, buscam-se ferramentas automatizadas que são capazes de obter a estrutura do software de forma detalhada. Com essas ferramentas é possível analisar códigos com milhares de linhas e construir

sua estrutura de uma forma mais rápida e eficiente.

Para realizar essa importação os sistemas fazem uso da engenharia reversa, uma vez que essas ferramentas buscam as informações do funcionamento do sistema diretamente no código ou estrutura do sistema. Para assim então, repassar as informações referentes ao software analisado para o usuário final.

No entanto, a maioria dessas ferramentas limitam os dados extraídos do código, ou seja não permitem que os dados recuperados sejam modificados de forma livre pelo usuário, forçando os dados a serem utilizados dentro dos próprios sistemas ou exigindo que sejam utilizados outros softwares auxiliares, assim complicando o uso de tais ferramentas para que pessoas que desejem estudar a estrutura arquitetônica do sistema de forma mais profunda, ou até mesmo realizar a integração desta ferramenta de extração ao seu sistema.

Outro problema geralmente encontrado por quem deseja estudar a estrutura de sistemas que já tem um tempo considerável em um ambiente de produção, está no fato de que as ferramentas atualmente disponíveis no mercado fazem somente o processo de extração dos dados. Deixando de lado o suporte para gerenciar a arquitetura gerada. Desta forma, os desenvolvedores devem utilizar outras ferramentas que sejam capazes de interpretar a estrutura gerada pelo sistema. Para então poder realizar modificações na estrutura do sistema. Entretanto, esse processo acaba demandando uma carga de trabalho extra sobre essas pessoas, uma vez que teriam de trabalhar com duas ferramentas diferentes para realizar seus estudos.

Mais uma dificuldade normalmente encontrada pelos usuários destas ferramentas está em que algumas vezes elas não são capazes de fazer a representação de todas as conexões e componentes que fazem parte da estrutura gerada, ou por omitir informações desejadas pelos usuários, como uma dependência que é criada dentro de um método específico, fazendo assim que usuário precise buscar manualmente as informações restantes que deseja

Observando tais dificuldades encontradas por pesquisadores e até mesmo pessoas ligadas a engenharia de software presentes em empresas, que desejam realizar estudos sobre a arquitetura de software, o presente trabalho visa implementar uma ferramenta, que seja capaz de a partir do código fonte de um determinado sistema, extrair sua arquitetura e realizar operações estruturais (adicionar, remover e modificar componentes conforme sua necessidade) sobre essa estrutura. Para alcançar este objetivo o processo de desenvolvimento foi separado nas seguintes etapas:

- Identificar um modelo de arquitetura de software que já tenha sido estudada por outros autores e se encaixe no perfil necessário na ferramenta desenvolvida;

- Implementar um módulo que seja capaz de ler os dados do código fonte do software;
- Implementar um módulo de geração da arquitetura;
- Implementar o módulo de gerenciamento da arquitetura gerada;
- Por fim avaliar a ferramenta desenvolvida através de estudos de exemplos de arquiteturas de software, conferindo se o sistema foi capaz de importar e gerenciar devidamente a estrutura.

O restante deste trabalho de conclusão de curso está dividido em 4 capítulos. No Capítulo 2 são apresentados trabalhos que vem sendo desenvolvidos em relação ao estudo proposto. No Capítulo 3 está descrito como foi desenvolvida a ferramenta proposta neste trabalho de conclusão, bem como exemplos de utilização da ferramenta proposta. O Capítulo 4 contém as conclusões acerca deste estudo.

2 REFERENCIAL TEÓRICO

2.1 ARQUITETURA DE SOFTWARE

O sistemas de software a cada ano que passam ganham mais importância na vida das pessoas, seja no trabalho ou na vida pessoal. Por este motivo os sistemas normalmente buscam desenvolver sempre melhorar para atender as necessidades das pessoas. Porém para que o sistema possa continuarem a ganhar importância e usuários novos, os desenvolvedores devem conhecer como o sistema funciona e toda sua arquitetura, para que então possam realizar as modificações necessárias no sistema.

No momento existem diversos conceitos de arquitetura de software. O que foi utilizado na produção deste trabalho de conclusão de curso é o seguinte: A Arquitetura de software se preocupa em representar os principais componentes de um software e especificar os relacionamentos que ocorrem entre eles (SHAW; GARLAN, 1996; BASS et al., 2012). Desse modo a arquitetura de software para o sistema desenvolvido tem a funcionalidade de representar as conexões e componentes presentes no sistema, buscando essas informações diretamente no código.

Possuir um projeto estrutural do sistema pode auxiliar de diversas formas, uma vez que possibilita a reutilização de componentes do sistema, tornar o sistema escalável, possibilita estudos de métricas sobre o sistema, facilita a manutenção do software, entre outros tantos benefícios.

Para poder compreender uma estrutura de software um desenvolvedor ou pesquisador pode necessitar de diversas visões ou perspectivas estruturais diferentes sobre o sistema que ao mesmo tempo não sejam iguais, mas possa cobrir pontos que a outra perspectiva não foi capaz de cobrir (KRUCHTEN, 1995; RIVA; RODRIGUEZ, 2002). Com o programador possuindo essas diversas visões sobre o sistema fica mais fácil de se compreender o sistema, ou alguma funcionalidade presente no sistema. Os estudos referentes a arquitetura normalmente buscam representar dois tipos de arquiteturas: Arquitetura Estática e Arquitetura Dinâmica, tais arquiteturas são descritas a seguir (HOFMEISTER et al., 1999; KRUCHTEN, 1995).

A representação estática tem como objetivo demonstrar como o código está moldado, sem levar em consideração possíveis modificações que o sistema possa sofrer durante o processo de execução. Essa estrutura pode ser recuperada através da análise do código fonte do sistema, ou através do estudo de diagramas estruturais da documentação do sistema.

A representação estática tem como principal vantagem o fato de ela poder ser recuperada do sistema sem o mesmo estar em execução, ou seja basta ter em mãos o código fonte do software ou sua documentação para obter-se a estrutura do software. No sistema proposto neste trabalho de conclusão de curso optou-se por utilizar esta forma de recuperar a arquitetura, uma vez que será trabalhada a extração da estrutura do código diretamente do código fonte do sistema a ser analisado (BRIAND et al., 2006).

O objetivo da arquitetura dinâmica é representar os componentes e relacionamentos do software analisados em tempo de execução. Ou seja, o sistema responsável por realizar a extração da arquitetura irá analisar o comportamento do código enquanto o sistema é executado. Assim verificando como o código é executado e o passo a passo que ele segue no sistema, e então gerando um relatório detalhado desta execução.

As vantagens em utilizar este método estão no fato de que o sistema é capaz de capturar relacionamentos criados através de chamadas dinâmicas, a identificação de objetos definidos por reflexão ou objetos que possuam vinculação tardia (YAN et al., 2004). Os problemas relacionados a este tipo de estrutura estão por conta da alta escalabilidade, uma vez que sistemas de pequeno porte podem tem milhares de objetos em tempo de execução (ABI-ANTOUN; ALDRICH, 2008). Outro problema está no fato de que modelos dinâmicos não fazem distinção entre objetos de mais baixo nível de objetos de maior valor arquitetural.

2.2 ENGENHARIA REVERSA

A engenharia reversa é uma série de tarefas que possibilitam, com base em um software já elaborado, retirar todos os conceitos ali implementados. Esses conceitos podem ser padrões arquiteturais, diagramas de classes, a arquitetura do sistema, enfim, qualquer informação que contribua para um entendimento do sistema desenvolvido (EILAM, 2011).

Os processos utilizados pela engenharia reversa devem ser capazes de derivar representações do projeto de software e informações sobre a estrutura de dados (a partir do nível mais baixo de abstração). Além disso, também devem ser capazes de representar um projeto de software a partir do seu programa executável, gerando representações de sua arquitetura (NELSON, 2005).

A aplicação da engenharia reversa é útil no contexto de manutenção de softwares legados. Um sistema pode ser uma peça fundamental para o dia-a-dia de uma organização, porém precisa passar por manutenções para se atualizar às exigências legislativas ou adaptativas. A engenharia reversa entra em ação nestes casos para auxiliar os desenvolvedores a conhecer o funcionamento/arquitetura do software (EILAM, 2011).

2.3 REPRESENTAÇÃO DA ARQUITETURA DO SOFTWARE

Para a representação da arquitetura de software comumente se é utilizada as linguagens de descrição arquitetural (ADLs) que tem o objetivo de especificar uma arquitetura de software e/ou hardware por meio de uma linguagem formal e/ou através de gráficos (ALDRICH et al., 2002). De modo geral, o ponto positivo das ADLs está na possibilidade de testar se a arquitetura está bem idealizada ou se há algum problema arquitetural no software antes mesmo de implementa-lo. Porém, a dificuldade em aprender a uma determinada ADL pode ser elevada. Existem diversas Linguagens estruturais já conhecidas, tais como byADL, ACME, ArchJava, etc.

A linguagem ArchJava tem como objetivo definir uma arquitetura, que se estende Java com arquitetura de abstrações, como componentes e conectores (ALDRICH et al., 2002). A limitação desta ADL está no fato de se o desenvolvedor desejar utilizar esta ADL em seus projetos, será obrigado à migrar seu código para o Java. Caso não utilize está linguagem em seu sistema. Porém está ADL não dá a possibilidade do usuário realizar o gerenciamento da arquitetura.

Outra liguagem de representação existente é a COTS (Components Off-The-Shelf). O COTS já implementa funcionalidades especificas que tem o objetivo de representar os requisitos funcionais gerais do software (MARTENS et al., 2010). Além disso, a COTS capaz de representar os componentes da arquitetura do software analisado, exibindo os componentes e conexões presentes no sistema.

Outra maneira de fazer a representação da arquitetura de um software é fazer a utilização de meta-modelos. Neste trabalho em especial será utilizado o meta-modelo apresentado por Ramírez et al. (2015). A seguir será detalhado o meta-modelo citado.

2.3.1 O META-MODELO

Para a representação da arquitetura de software utilizada como base neste trabalho de conclusão de curso será utilizado o meta-modelo para à representação arquitetural de um sis-

tema proposta por Ramírez et al. (2015). Este modelo de representação tem como objetivo representar a arquitetura do software através de uma árvore genética, separando cada componente com sua devida composição(classes e conexões). A Figura 1 demonstra como é feita a representação arquitetural, onde a primeira parte é representado como o código está estruturado, e a segunda é como o meta-modelo da arquitetura ficará utilizando-se a proposta de Ramírez et al. (2015).

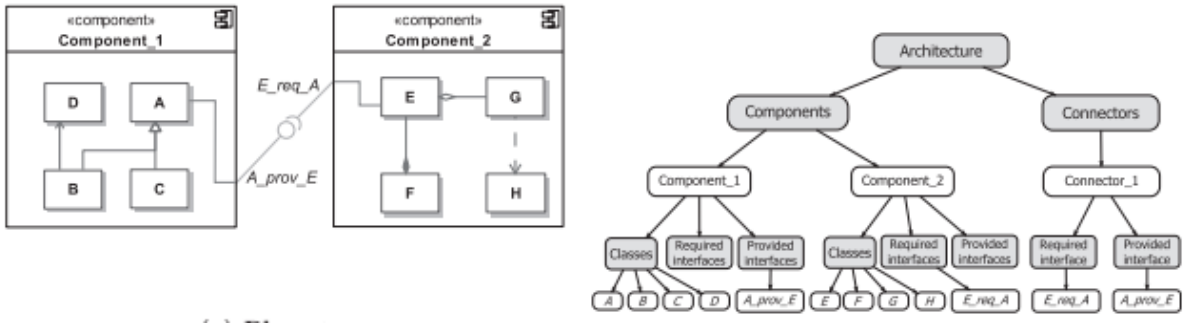

Figura 1: Exemplo de representação feita pela proposta de Ramírez et al. (2015)
Fonte: Adaptado de Ramírez et al. (2015)

Como é possível observar na Figura 1 o sistema de representação utilizado por Ramírez et al. (2015) representa cada pacote do sistema como um componente. E dentro deste componente eles buscam representar as classes que fazem parte deste componente, bem como as conexões das classes que o compõem. Essas conexões são dividas em dois tipos, `Provided`‘(Provido) e `Required`(Requerido), onde a ligação do tipo provido representa que a classe do componente está fornecendo uma ligação a outra classe. No caso do exemplo da Figura 1 uma representação desta ligação é onde a classe `A` está provendo uma ligação com a classe `E`. Já a ligação do tipo requerido representa que uma classe do componente requer uma ligação a outra classe. Na Figura 1, é possível visualizar um exemplo deste tipo de conexão, onde a `E` está requerendo uma ligação com a classe `A`.

Outra característica que é possível ser observada nesta estrutura, está no fato de ela ter um ramo da árvore é dedicado a representar todas os conectores existentes na arquitetura. Onde um conector representa a junção entre a conexão requerida e à conexão provida entre as classes.

2.4 TRABALHOS RELACIONADOS

Nesta seção serão descritos trabalhos relacionados a ferramenta implementada neste trabalho de conclusão. Apresentando suas características e diferenças ou melhoras que foram incluídas no i.code.reader.

A abordagem da ferramenta SArF Map recupera a arquitetura estática de um software e faz a representação gráfica desta estrutura para o usuário, dividindo ela em camadas e módulos (KOBAYASHI et al., 2013). A Figura 3 ilustra um exemplo de arquitetura gerada pelo SArF. Porém o SArF Map faz o trabalho de extração de dados e apenas apresenta esses dados ao usuário, não permitindo qualquer modificação na estrutura, diferentemente do sistema proposto neste trabalho, onde o usuário terá a possibilidade de alterar a arquitetura extraída e estudar se suas modificações são benéficas.

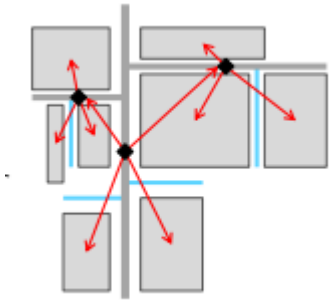

Figura 2: Exemplo de arquitetura gerada pelo SArF Map
Fonte: Adaptado de Kobayashi et al. (2013)

Os Reflection Mode (RM) é outra abordagem utilizada para a extração de arquitetura estática a partir do código (MURPHY et al., 1995). A técnica RM consiste que os desenvolvedores tenham de fornecer uma estrutura planejada de alto nível e um mapeamento entre o modelo declarativo e o modelo do código-fonte. Uma ferramenta baseada em RM como a SAVE (*Software Architeture Visualization and Evaluation*) (KNODEL; POPESCU, 2007), destaca as relações ausentes, divergentes ou convergentes entre o modelo de alto nível e o modelo do código fonte. Porém, isso acaba acarretando em uma carga de trabalho extra, uma vez que os desenvolvedores teriam de criar anotações no código para que o sistema possa executar as verificações necessária. Essa é a principal diferença entre esta abordagem utilizada no SAVE e a do sistema proposto, logo que o software irá retirar essas informações diretamente do código sem a necessidade de trabalho extra do programador criar anotações em seu código.

Uma outra abordagem simples de recuperação estática da arquitetura é conhecida como Womble (JACKSON; WAINGOLD, 1999). Ela funciona de forma semelhante ao RM, porém essa feramenta não faz uso da reflexão para identificar a estrutura. Nesta ferramenta especificamente, o desenvolvedor também descreve as anotações e o sistema percorre através da mineração do código buscando as anotações para assim identificar a estrutura. Outro fator problemático é que o Womble não tem meios de sumarizar o código fazendo com que uma pequena aplicação possua milhares de objetos.

Outra ferramenta que segue o método de extração de dados de forma estática é o *Au-*

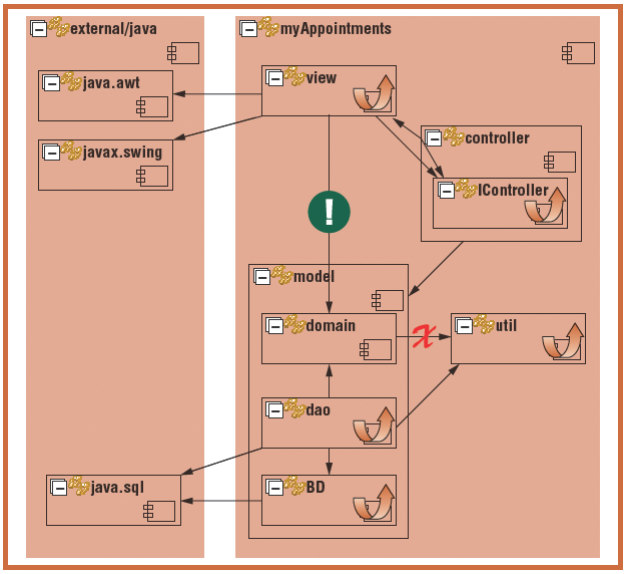

Figura 3: Exemplo de arquitetura gerada pelo SAVE
Fonte: Adaptado de Knodel e Popescu (2007)

tomated Software Architecture Extraction Using Graph-based Clustering (CHARGO, 2013). Nesta abordagem, a ferramenta extrai os dados do código fonte e divide a arquitetura recuperada em módulos de acordo com a função das classes em grafos, com o objetivo de ter uma representação mais clara da estrutura do software para o usuário de como o sistema funciona. Porém ela entra no mesmo caso da ferramenta SArF, onde o usuário seria obrigado a utilizar os dados que foram passados a ele e não teria como modificá-los.

2.5 TECNOLOGIAS UTILIZADAS

Nesta seção serão abordadas as tecnologias escolhidas para o desenvolvimento da ferramenta proposta neste trabalho, com o objetivo de ter agilidade em sua elaboração e explorar ao máximo outras tecnologias disponíveis, visando ter os melhores resultados possíveis.

A linguagem de programação que foi selecionada para a implementação da ferramenta foi o `Java`[1], uma vez que o conhecimento na sintaxe da linguagem é satisfatório, outro fator que foi observado é a grande quantidade de componentes disponíveis para a linguagem que podem auxiliar no processo de desenvolvimento.

Para realizar a extração de dados do código fonte optou-se por utilizar a biblioteca `JAVA PARSER`[2], um projeto que é capaz de ler todos os elementos que fazem parte de uma classe `Java`. Para resolver as dependências necessárias no projeto foi escolhido o `Maven`[3], uma

[1] Disponível em https://www.java.com/
[2] Disponível em http://javaparser.github.io/javaparser/
[3] Disponível em: https://maven.apache.org/

vez que basta o usuário ter uma ligação com a internet para que o sistema realize o download das bibliotecas necessárias e consiga executar o sistema.

3 A FERRAMENTA: I.CODE.READER

3.1 APRESENTAÇÃO SISTEMA DE EXTRAÇÃO DE ESTRUTURAS

Com o intuito de implementar e avaliar a abordagem que foi proposta neste trabalho, foi desenvolvido uma ferramenta denominada de `i.code.reader`[1]. Está aplicação é capaz de extrair os dados referentes ao código de um determinado sistema, que foi implementado na linguagem `Java`. E com essas informações extraídas apresentar o sistema em uma estrutura, a fim de representar como o sistema foi elaborado.

O principal objetivo do sistema é representar o algoritmo desenvolvido no projeto em uma estrutura. Deste modo, fazer que seja possível encontrar problemas estruturais de código de uma forma mais eficaz por exemplo.

Para determinar a estrutura do algoritmo é preciso que o desenvolvedor tenha em mãos o código fonte do sistema e esse código seja desenvolvido em `JAVA`, para que ele assim informe ao programa extrator o pacote raiz do sistema. Com as informações que o sistema conseguir coletar do código passado a ele será possível de se moldar a estrutura proposta.

De forma geral, os dados coletados pelo sistema do código fonte original são os pacotes do sistemas, as classes presentes em cada um destes pacotes, as heranças, caso exista, de uma classe a outra classe do sistema, as implementações de interfaces que a classe possui, além das referências que a classe faz às demais classes do projeto. Essa categoria engloba as declarações de atributos, tipos de parâmetros de métodos, variáveis declaradas em métodos e retorno de métodos, que referencie uma outra classe presente no projeto. Com esses dados extraídos o sistema é capaz de determinar se a referência é feita internamente em um pacote ou a classe tem uma referência feita a um pacote que não seja o que ela está, considerando assim ela uma referência externa.

Os dados extraídos do código fonte são salvos somente na memória do computador, foi optado realizar a execução do projeto desse modo para buscar uma maior agilidade no processo

[1] https://github.com/edukempf/code.icode.reader

de execução do sistema. Outra característica do sistema é que ele faz a análise do código a nível de classe.

Os detalhes da arquitetura, características sobre a extração de dados, criação e gerenciamento da estrutura são apresentados nas seções a seguir.

3.2 A ARQUITETURA DA FERRAMENTA

A arquitetura do projeto foi feita com o objetivo de que o sistema seja acoplado a outra ferramenta sem muitas dificuldades. A Figura 4 demonstra a maneira como os módulos do sistema foram divididos e como estão sendo interligados. Tais módulos serão introduzidos nesta seção e detalhados nas seções subsequentes.

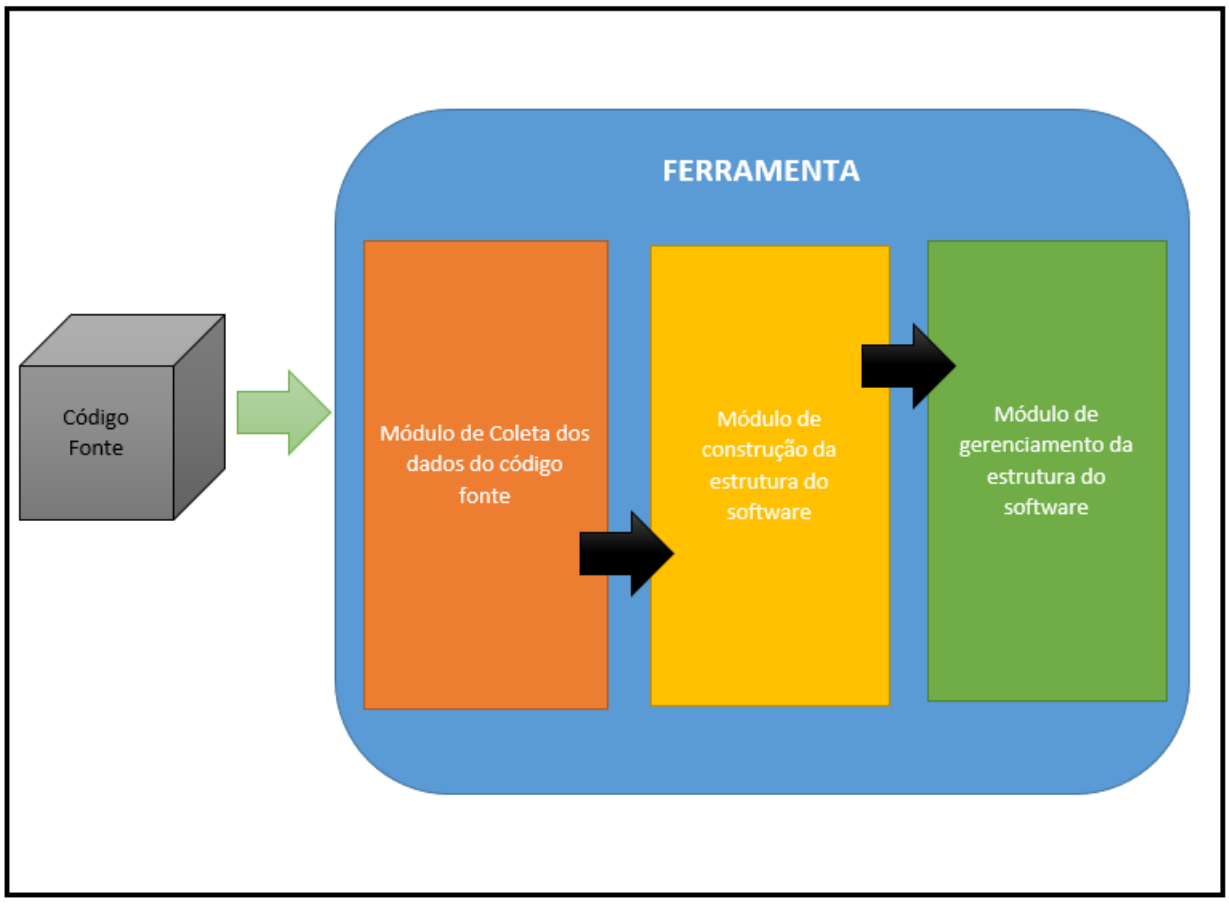

Figura 4: Comunicação entre os módulos desenvolvidos no sistema
Fonte: Autoria do autor

O módulo de coleta de dados é responsável por realizar a leitura do código fonte, através do caminho do pacote raiz do sistema que foi passado pelo usuário. Os dados obtidos são colocados em memória e ficam disponíveis para os módulos seguintes do sistema fazerem uso deles.

Logo que os dados foram coletados, é possível recriar a estrutura do sistema, e com isso entra em ação o módulo de criação da estrutura, onde os dados coletados serão analisados e a estrutura geral será moldada.

O último módulo do sistema é responsável por realizar o gerenciamento da estrutura obtida. Através dele, o usuário é capaz de modificar a arquitetura do sistema da forma que ele necessite para analisar o resultado obtido.

3.2.1 MÓDULO DE COLETA DE DADOS DO CÓDIGO FONTE

O módulo de coleta de dados tem a função de extrair os dados necessários do projeto diretamente do código fonte. É através deste módulo que se torna possível a obtenção dos dados do projeto para serem estudados. Para realizar está tarefa de coleta de dados o sistema utiliza uma biblioteca `JAVA` denominada `JAVAPARSER`, que faz a leitura do código e é capaz de identificar as características que compõem o código. A Figura 5 demonstra a estrutura de classes que compõem este módulo.

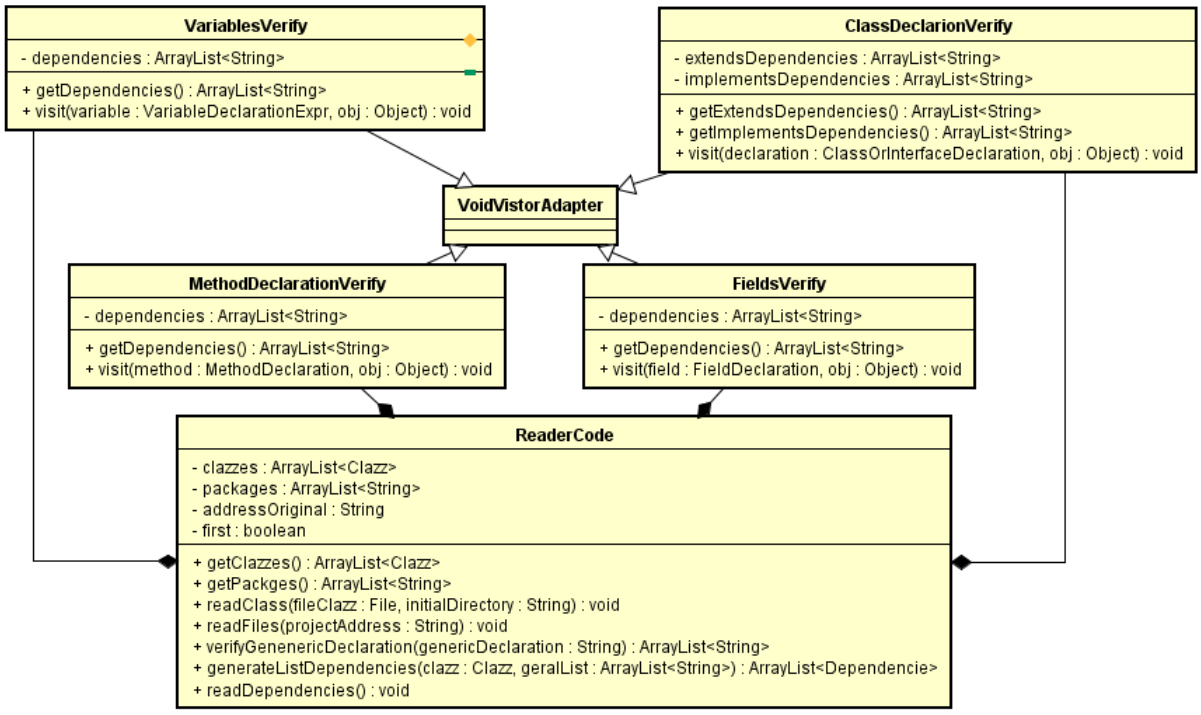

Figura 5: Diagrama resumido das classes do módulo
Fonte: Autoria do autor

As classes da Figura 5 tem as seguintes funcionalidades no sistema:

- `VoidVisitorAdapter` - Classe da biblioteca `JAVA PARSER`; é a partir desta classe que o sistema consegue efetuar a extração dos dados das classes;

- `ClassDeclarationVerify` - Esta classe tem a responsabilidade de verificar possíveis dependências de herança ou de interface da classe; ela faz isso analisando a declaração da classe;
- `FieldsVerify` - Esta classe está encarregada de verificar as referências feitas nos atributos das classes;
- `MethodDeclarationVerify` - Nesta classe, o objetivo é identificar as dependências geradas a partir das declarações dos métodos das classes analisadas;
- `VariablesVerify` - Nesta classe são buscadas as dependências geradas por variáveis que são declaradas dentro dos métodos do sistema;
- `ReaderCode` - Esta classe pode ser considerada o ”coração”do módulo uma vez que é ela quem gerencia o processo de extração dos dados.

Como dito anteriormente a classe `ReaderCode` é de fundamental importância para o módulo, sendo assim será detalhado o funcionamento de cada método da classe:

- `getClazzes` - Este método é responsável por retornar as informações das classes lidas durante o processo de extração dos dados;
- `getPackages` - Neste método o objetivo é retornar os pacotes que foram encontrados;
- `readClass` - Aqui são lidas as informações básicas da classe, como o pacote a qual ela pertence, o caminho que ela se encontra no momento e o nome dela. As informações lidas são salvas em um `ArrayList` na memória do computador;
- `readFiles` - Neste método o sistema percorre de forma recursiva as subpastas do caminho passado pelo usuário procurando por arquivos ”.java”, quando encontrado chama o método `readClass` para ler as informações. Quando ele encerra a etapa de leitura das classes ele invoca o método responsável por ler as dependências;
- `verifyGenenericDeclaration` - Este método é acionado somente quando se identifica uma declaração do tipo genérica, por exemplo `ArrayList<ClasseB>`, então é invocado este método para que ele identifique quais são as classes referenciadas nesta declaração e retornar estas dependências, caso ela seja uma considerada válida;
- `generateListDependencies` - Este método é responsável por fazer a filtragem das dependências lidas, ou seja ela irá receber uma lista de `Strings` que são como as dependências lidas pelo `JAVAPARSER` são retornadas, caso seja válida ele cria um elemento

do tipo `Dependencie`, informando se a dependência é feita a uma classe do mesmo pacote ou a um pacote externo.

- `readDependencies` - Neste método é onde são realizadas as leituras de dependências do projeto, utilizando as classes `MethodDeclarationVerify`, `FieldsVerify`, `ClassDeclarationVerify` e `VariablesVerify`. A partir do retorno de cada uma destas classes o sistema realiza a filtragem das dependências com o método `generateListDependencies` e salva as dependências de cada classe lida durante o processo de extração.

Com o processo de extração de dados finalizado o próximo passo invoca o módulo de criação de estrutura. Este módulo será explicado de forma mais detalhada na próxima seção.

3.2.2 MÓDULO DE CONSTRUÇÃO DA ESTRUTURA DO SOFTWARE

Este módulo está presente no sistema com o objetivo de gerar a estrutura geral do sistema analisado. Para isso o sistema utiliza os dados lidos no módulo anterior. Este módulo é composto por somente uma classe, como o mostrado na Figura 6.

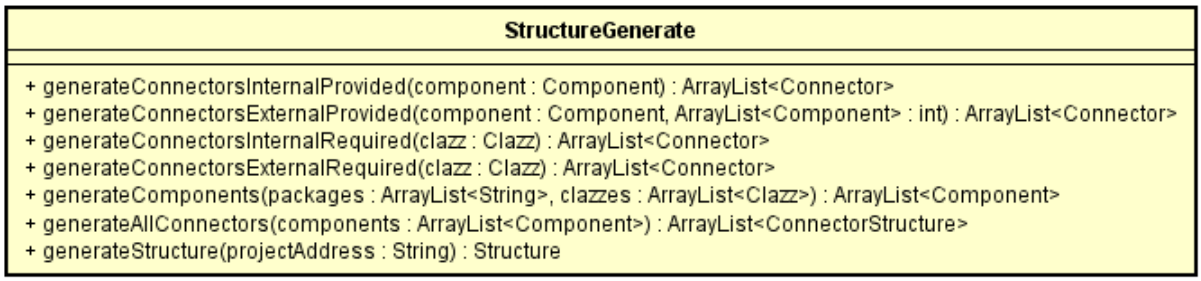

Figura 6: Diagrama resumido das classes do módulo de criação da estrutura
Fonte: Autoria do autor

Como o visto na Figura 6, o módulo possui vários métodos que são responsáveis pela análise e geração da estrutura. A seguir uma explicação de funcionalidade de cada um dos métodos implementados na classe:

- `generateConnectorsInternalProvided` - Este método é responsável por criar as conexões que são providas internamente no pacote. Para isso, o método irá receber por parâmetro o pacote a ser analisado, verificando as conexões que já são requeridas internamente e criar as novas conexões do tipo providas internamente;

- `generateConnectorsExternalProvided` - Neste método, o objetivo é criar as conexões que são providas de classes deste pacote a classes dos demais pacotes. Para isso o

método recebe por parâmetro o pacote a ser criado as conexões e os outros pacotes presentes no sistema. Com esses dados o método percorre os demais pacotes presentes no sistema. Em cada pacote que o sistema passar ele verificará se nele existem classes que dependam de classes presentes no pacote que deseja-se criar as conexões. Caso exista o sistema criará uma nova conexão no pacote do tipo provida externamente;

- `generateConnectorsInternalRequired` - Método responsável por criar as conexões que são requeridas internamente. Para realizar a tarefa, o método receberá a classe a ser analisada, verificando as dependências contidas na classe. Caso a dependência seja interna, ele criará uma conexão do tipo requerida internamente. Durante o processo o sistema também faz a distinção do tipo de dependência, se é de herança, interface ou de referências gerais (declaração e retorno de métodos, declaração de variáveis ou declarações de atributos);

- `generateConnectorsExternalRequired` - Este método faz um processo semelhante ao processo do método `generateConnectorsInternalRequired`, a única diferença durante o processo está no fato de ele analisar as dependências externas e criar as conexões do tipo requerida externamente;

- `generateComponents` - O processo realizado neste método consiste em criar os pacotes, com suas devidas classes e conexões. Para isso, o método receberá as classes e pacotes que foram lidos anteriormente, assim ele percorrerá os pacotes e separando as classes de acordo com o seu devido pacote e assim gerando as conexões internas e externas do pacote durante o processo com o auxílio dos métodos anteriores, ao fim do processo o método retornará a lista de componentes gerados;

- `generateAllConnectors` - Este método tem a tarefa de gerar a lista de todos os conectores presentes no sistema. Desse modo, o método irá receber os componentes da estrutura e percorrer todas as conexões presentes em cada pacote do sistema, assim que ele obtiver todas as informações necessárias ele irá retornar a lista das conexões;

- `generateStructure` - Este é o método responsável por gerar e retornar a estrutura do sistema a ser analisado. Para realizar tal tarefa, ele invoca os métodos anteriores para gerar os dados necessários para a criação da estrutura, cria o objeto do tipo `Struture` com as informações retornadas pelos outros métodos e então retorna este ojeto contendo a estrutura gerada para quem realizou a chamada do método.

3.2.3 MÓDULO DE GERENCIAMENTO DA ESTRUTURA DO SOFTWARE

o módulo de gerenciamento da estrutura do software foi criado com o objetivo de disponibilizar ao usuário uma forma de gerenciar a estrutura gerada de forma que ele possa realizar estudos de melhorias possíveis no sistema analisado, sem a necessidade de exportar a estrutura gerada para uma outra ferramenta.

Para realizar esta tarefa de gerenciamento, foi criada uma classe que é responsável por realizar as modificações na estrutura. A Figura 7 demonstra os métodos implementados pela classe. A seguir, será apresentado uma explicação detalhada da implementação de cada método da classe:

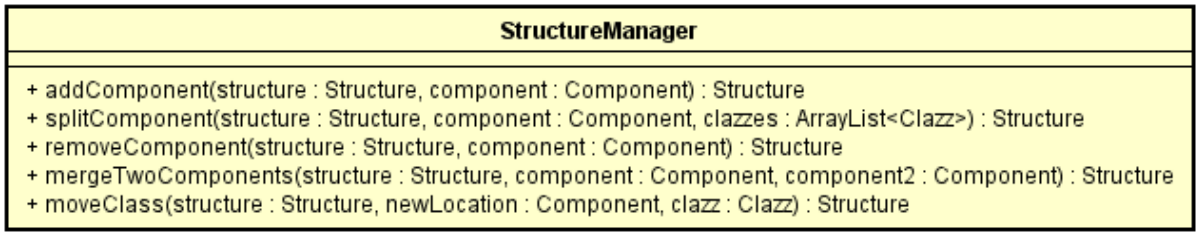

Figura 7: Diagrama resumido das classes do módulo de criação da estrutura
Fonte: Autoria do autor

- `addComponent` - Este método é responsável por realizar a adição de novos componentes à estrutura. Para isso, o usuário passa por parâmetro o componente a ser adicionado. O método é responsável por alocar este componente na estrutura e retornar a nova estrutura gerada para o usuário.

- `splitComponent` - A tarefa realizada por este método é fazer a divisão de um componente já existente. O método precisa receber por parâmetro o componente que será dividido e as classes que serão realocadas. Com esses dados o sistema cria dois novos componentes à estrutura e então realoca em um dos componetes criados as classes remanescentes do componente original e no outro as classes que foram solicitadas. Com as classes devidamente alocadas o sistema refaz as conexões destes dois novos componentes. A última etapa deste processo de divisão é excluir o componente original e retornar a nova estrutura gerada.

- `removeComponent` - O processo executado por este método é basicamente receber o componente que deve ser extraído, verificar se este componente possui ou não alguma classe dentro dele e, caso não possua, o sistema irá excluir e retornar a nova estrutura gerada. Caso exista alguma classe, ele irá somente retornar a estrutura original.

- `mergeTwoComponents` - É o método responsável por realizar a união entre dois componentes da estrutura. São necessários os dois parâmetros, que serão os dois componentes que irão ser unidos. Com esses dados o método irá criar um novo componente na estrutura para representar a união entre os dois componentes e então mover as classes destes dois componentes para esse novo componente. Após ter movido todas as classes, o sistema irá fazer as conexões deste novo componente. Assim que terminado esse processo o método excluirá os componentes antigos e retornar a nova estrutura.
- `moveClass` - Este método será responsável por retirar uma classe de um componente e adicionar ela a outro. Para que ele possa realizar este processo terá de ser passado a ele qual classe será movida, o componente que ela pertence atualmente e o novo componente que ela será alocada. O processamento do método será de remover a classe de seu componente antigo e alocar ela em seu novo local, porém após realizar está modificação é necessário que o sistema refaça as conexões dos dois componentes envolvidos. Com todo o processo de execução já finalizado o sistema irá retornar a nova estrutura.

3.3 FUNCIONAMENTO DA FERRAMENTA

Todo o processo tem início com a extração de dados do código fonte a ser estudado pela ferramenta. Para isso, o usuário terá de passar o caminho do pacote raiz do sistema que ele deseja obter os dados. A partir do momento que o sistema reconhece a pasta como válida ele irá percorrer as subpastas até identificar todos os pacotes e classes do projeto.

Logo após ele ter identificado esses dados do projeto o sistema passa a etapa de ler os dados internos de cada classe. Para realizar esta tarefa, é utilizada a API `JavaParser`, uma biblioteca que identifica os elementos que compõem uma classe JAVA. Nesta parte, ele busca as seguintes informações dentro do código do projeto:

- Verifica se a classe implementa uma interface existente dentro do projeto;
- Verifica se a classe herda alguma outra classe presente no projeto;
- Busca entre os atributos da classe algum que seja do tipo de alguma outra classe do sistema;
- Verifica se os métodos da classe tem algum parâmetro ou retorno do tipo de outra classe do projeto;
- A última parte deste processo consiste em verificar as variáveis declaradas dentro dos métodos que realizam referência a outra classe do sistema.

Em cada uma dessas etapas o sistema irá dividir as referências em três tipos diferentes que são: ”referências de Interfaces”, onde serão colocadas as referências a interfaces implementadas no sistema. O segundo tipo de referência vai estar o de ”referência de herança”, na qual as referências que representam as heranças que cada classe possui. E o último tipo que é ”referências gerais”, que correspondem as demais referências feitas no código, como as de criação de variáveis e atributos, além de retorno e parâmetros dos métodos.

Com esses dados o `i.code.reader` passa para a etapa de criação da arquitetura do sistema. Para isso, o sistema utiliza como base o meta-modelo para a representação da estrutura de um sistema, proposta por Ramírez et al. (2015), que foi apresentada na Seção 2.3 deste trabalho. Porém com o objetivo de aprimorar a representação da estrutura para o usuário foram feitas algumas modificações na forma de moldar a estrutura.

As modificações consistem em criar conexões para representar herança de classe e conexões de referência para representar quando uma classe referência outra classe de alguma forma, como em criação de atributos que são do tipo de uma classe do projeto, retorno de métodos ou criação de variáveis. Além das modificações citadas o sistema também faz uma distinção se as conexões são internas no pacote, quando uma classe referência outra classe em um mesmo pacote, ou em conexões externas, quando a classe referência a uma classe alocada em um pacote externo.

Desse modo, a primeira etapa para a criação dessa estrutura será a criação dos componentes. Para isso, o sistema percorre primeiramente os pacotes que fora passados a ele, e para cada um destes pacotes é criado um novo componente. Quando finalizada a criação de todos os componentes presentes na estrutura, o sistema passa para a etapa de separação das classes.

Na etapa de separação de classes, basicamente o sistema fica encarregado de separar as classes de acordo com o pacote que cada uma está alocada no projeto estudado. Para isso o `i.code.reader` percorrerá as classes que foram repassadas a ele e comparar o pacote a qual a classe pertence e posicionará a classe no componente que representa este pacote no sistema.

O próximo passo será a criação dos conexões internas e externas de cada classe. Essas conexões irão representar as referências que a classe possui a outras classes dentro de um mesmo componente, ou a referências feitas a classe que pertencem a outros pacotes do sistema. Para isso, o sistema percorre cada uma das dependências de cada uma das classes e compara se a classe referenciada pertence ao mesmo pacote. Caso pertença, é criada uma conexão do tipo interna. Se a referência for a uma classe de um pacote externo, o tipo do conector será externo. Tanto o conector interno quanto o externo serão criados como requeridos, que representa que a classe lida requer que a outra classe exista. O sistema faz esta comparação para todas as classes

lidas.

A próxima parte da geração da estrutura vai estar em criar as conexões do tipo providas. Estas conexões representam as classes que estão provendo uma ligação a outras classes, ou seja estas conexões são o inverso das requeridas, as conexões do tipo providas irão representar as classes que possuem dependências delas nas demais classes do sistema. Para criar tais conexões o sistema percorre todos as conexões internas e externas do tipo requeridas de cada classe. Ele verifica se a conexão é interna, em caso positivo, os sistema irá somente inverter a classe que prove e que requer, gerando assim uma nova conexão do tipo providas dentro do mesmo componente. Caso seja externa, o programa percorre os demais componentes do sistema, com o objetivo de encontrar o componente que a classe requerida pertence. Quando localizado, o sistema irá criar uma conexão do tipo providas no componente em que a classe requerida foi encontrada.

A última etapa do processo de criação da estrutura está na identificação de todos os conectores presentes do projeto. Neste processo o sistema irá percorrer todos os componentes que foram criados, porém agora buscando as conexões de cada um dos componentes. Toda vez que ele encontra uma conexão que ainda não foi adicionada à lista de conectores gerais, ele é adicionado à lista. Após terminar de adicionar todos os conectores, a estrutura já está pronta para ser analisada e estudada pelo usuário.

A última parte do sistema é a parte de gerenciamento da estrutura, porém essa parte depende de uma iteração com o usuário, uma vez que é ele quem decidi se vai ser necessário realizar alguma modificação ou não na estrutura. Considerando que haja modificação necessária, o usuário irá repassar ao sistema qual será a modificação necessária. No momento o sistema é capaz de realizar as seguintes alterações na estrutura: adicionar um novo pacote (componente), remover um pacote, mover uma classe de um pacote a outro, unir dois pacotes distintos e separar determinadas classes de um pacote em um novo pacote.

Assim que o usuário realizar todas as modificações que julgar necessárias ele poderá obter a nova estrutura para seus estudos, e caso o usuário julgue que sejam necessárias mais modificações na estrutura ele fazer as modificações necessárias repassando a estrutura e a modificação necessária ao sistema novamente.

3.4 EXEMPLO DE USO DA FERRAMENTA

Conforme descrito nas seções anteriores, o sistema é capaz de realizar a extração da arquitetura de um software a partir do código fonte do mesmo. Nesta seção, será apresentado

um exemplo de uso das funcionalidades da ferramenta.

O primeiro passo para a produção do exemplo foi criar uma estrutura de código JAVA, que fosse capaz de cobrir o máximo possível de funcionalidades da ferramenta. A estrutura do exemplo está o representada na Figura 8.

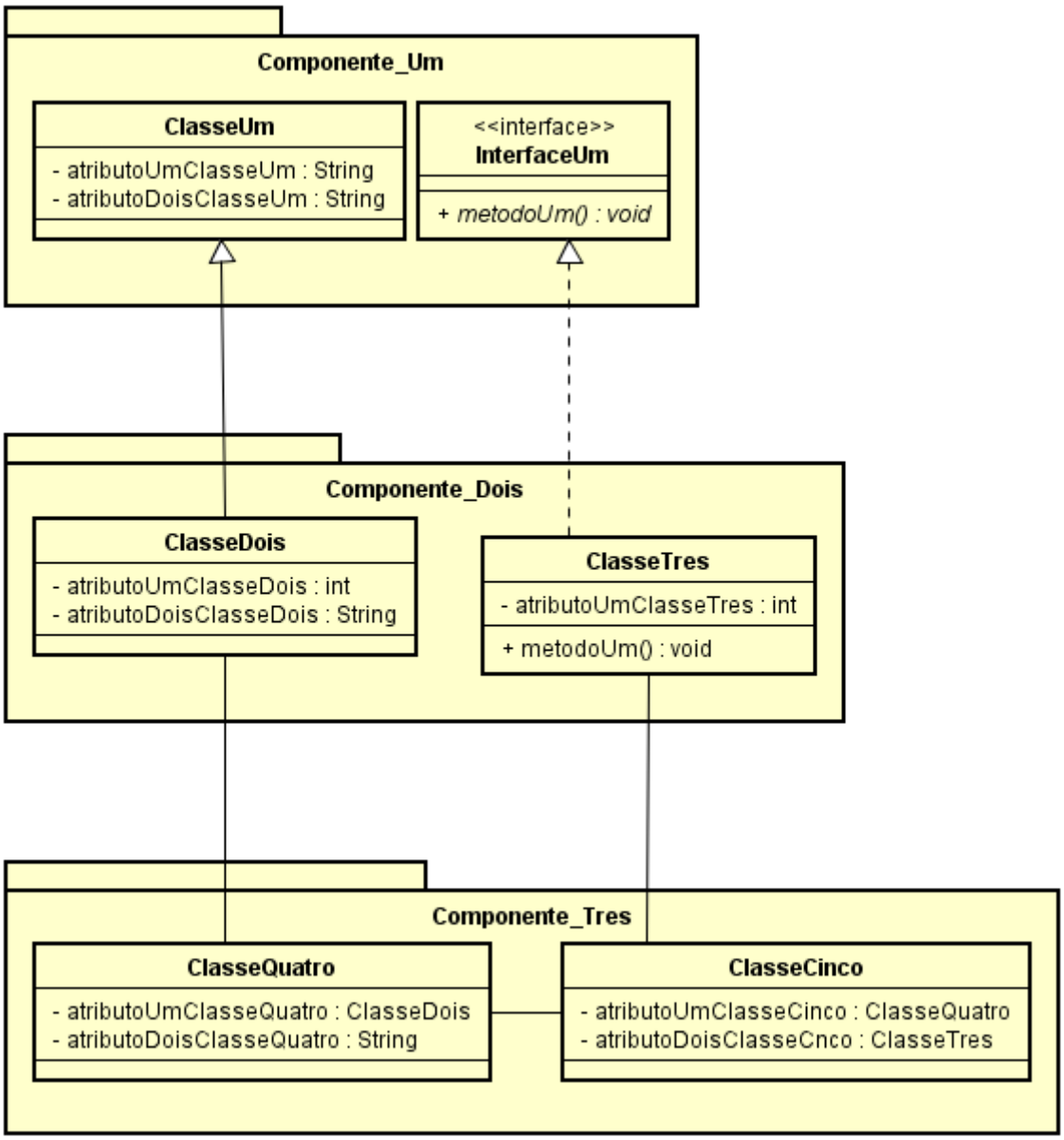

Figura 8: Diagrama do código fonte que será utilizado no exemplo
Fonte: Autoria do autor

Na ferramenta o primeiro passo é extrair do código fonte a estrutura e validar se a ferramenta está fazendo a extração de forma correta. Desse modo, a primeira etapa desse exemplo será criar um objeto do tipo StrutureGenerate e pedir para o sistema gerar a estrutura a partir do código fonte criado, como mostrado na Figura 9.

Depois de informar o caminho do código fonte à ferramenta, basta executar a mesma

```
1  package br.edu.utfpr.code.icode.reader;
2
3  import java.io.FileNotFoundException;
15
16 public class App {
17     public static void main(String[] args) {
18         StrutureGenerate sm = new StrutureGenerate();
19         try {
20             Structure s = sm.generateStructure("C:\\Users\\Eduardo\\workspace\\Exemplo\\src");
21
```

Figura 9: Informando o caminho do código fonte ao sistema
Fonte: Autoria do autor

e ela irá retornar a estrutura do código fonte, uma vez que não será feito nenhuma modificação na estrutura nesta parte do exemplo. A estrutura que foi retornada pela ferramenta pode ser visualizada na Figura 10.

ARQUITETURA GERADA

```
COMPONENTES
    ComponenteDois
        CLASSES
            ClasseDois.java
            ClasseTres.java
        REQ INT EXT
            ClasseTres_req_InterfaceUm
        REQ EXT EXT
            ClasseDois_req_ClasseUm
        PRO REF EXT
            ClasseTres_pro_ClasseCinco
            ClasseDois_pro_ClasseQuatro
    ComponenteTres
        CLASSES
            ClasseCinco.java
            ClasseQuatro.java
        REQ REF INT
            ClasseCinco_req_ClasseQuatro
        REQ REF EXT
            ClasseCinco_req_ClasseTres
            ClasseQuatro_req_ClasseDois
        PRO REF INT
            ClasseQuatro_pro_ClasseCinco
    ComponenteUm
        CLASSES
            ClasseUm.java
            InterfaceUm.java
        PRO INT EXT
            InterfaceUm_pro_ClasseTres
        PRO EXT EXT
            ClasseUm_pro_ClasseDois

CONECTORES
    Connector_0
        REQUIRED
            ClasseDois_req_ClasseUm
        PROVIDED
            ClasseUm_pro_ClasseDois
    Connector_1
        REQUIRED
            ClasseTres_req_InterfaceUm
        PROVIDED
            InterfaceUm_pro_ClasseTres
    Connector_2
        REQUIRED
            ClasseCinco_req_ClasseQuatro
        PROVIDED
            ClasseQuatro_pro_ClasseCinco
    Connector_3
        REQUIRED
            ClasseCinco_req_ClasseTres
        PROVIDED
            ClasseTres_pro_ClasseCinco
    Connector_4
        REQUIRED
            ClasseQuatro_req_ClasseDois
        PROVIDED
            ClasseDois_pro_ClasseQuatro
```

Figura 10: Arquitetura Gerada pelo `i.code.reader` para o código fonte de exemplo
Fonte: Autoria do autor

No momento da saída, foram abreviados os nomes dos tipos de conectores, por isso

aqui segue o significado de cada uma das abreviações que podem aparecer neste e nos demais exemplos implementados neste trabalho:

- REQ INT INT - Conexão do tipo Inteface Requerida Internamente no pacote;
- REQ INT EXT - Conexão do tipo Inteface Requerida Externamente ao pacote;
- REQ EXT INT - Conexão do tipo Herança Requerida Internamente no pacote;
- REQ EXT EXT - Conexão do tipo Herança Requerida Externamente ao pacote;
- REQ REF INT - Conexão do tipo Referência Requerido Internamente no pacote;
- REQ REF EXT - Conexão do tipo Referência Requerido Externamente ao pacote;
- PRO EXT INT - Conexão do tipo Herança Provida Internamente no pacote;
- PRO EXT EXT - Conexão do tipo Herança Provida Externamente ao pacote;
- PRO INT INT - Conexão do tipo Inteface Provida Internamente no pacote;
- PRO INT EXT - Conexão do tipo Inteface Provida Externamente ao pacote;
- PRO REF INT - Conexão do tipo Referência Provido Internamente no pacote;
- PRO REF EXT - Conexão do tipo Referência Provido Externamente ao pacote;

Ao analisar a saída do sistema, é possível verificar que o sistema efetuou seu trabalho de forma correta, uma vez que foi capaz de trazer todos componentes (Pacotes), Classes e suas conexões, separando cada uma dessas conexões de acordo com o que foi proposto neste trabalho.

A próxima parte da execução de nosso exemplo está em trabalhar com o gerenciador de estruturas da ferramenta. Para isso será gerado um exemplo de cada uma das funcionalidades de gerenciamento implementados.

O primeiro exemplo divide um componente em dois, para isso será utilizada a estrutura utilizada no exemplo anterior. O componente que será dividido neste exemplo será o `ComponenteDois`, onde iremos separar a `ClasseDois` e a `ClasseTres` em componentes diferentes.

Para isso, será criado um objeto do tipo `StructureManager`, e logo após irá ser executado o método `splitComponent`, passando a estrutura gerada e a classe que será separada do componente. Como o mostrado na Figura 11.

```
38              StructureManager sm = new StructureManager();
39              structure = sm.splitComponent(structure, component, clazzes);
```

Figura 11: Código para realizar uma separação de componentes
Fonte: Autoria do autor

Com o código para separar os componentes implementado basta executar o código novamente para se obter a nova estrutura gerada pelas modificações. A Figura 12 representa a estrutura que foi gerada a partir da separação do `ComponenteDois`. É possível verificar que os conectores gerais não foram modificados, somente os conectores do `ComponenteDois` que tiveram de serem realocados de acordo com a separação das classes.

ARQUITETURA GERADA APÓS A SEPARAÇÃO DE DOIS COMPONENTES

```
COMPONENTES
    ComponenteTres
        CLASSES
            ClasseCinco.java
            ClasseQuatro.java
        REQ REF INT
            ClasseCinco_req_ClasseQuatro
        REQ REF EXT
            ClasseCinco_req_ClasseTres
            ClasseQuatro_req_ClasseDois
        PRO REF INT
            ClasseQuatro_pro_ClasseCinco
    ComponenteUm
        CLASSES
            ClasseUm.java
            InterfaceUm.java
        PRO INT EXT
            InterfaceUm_pro_ClasseTres
        PRO EXT EXT
            ClasseUm_pro_ClasseDois
        PRO REF EXT
    ComponenteDois_pt1
        CLASSES
            ClasseTres.java
        REQ INT EXT
            ClasseTres_req_InterfaceUm
        PRO REF EXT
            ClasseTres_pro_ClasseCinco
    ComponenteDois_pt2
        CLASSES
            ClasseDois.java
        REQ EXT EXT
            ClasseDois_req_ClasseUm
        PRO REF EXT
            ClasseDois_pro_ClasseQuatro

CONECTORES
    Connector_0
        REQUIRED
            ClasseDois_req_ClasseUm
        PROVIDED
            ClasseUm_pro_ClasseDois
    Connector_1
        REQUIRED
            ClasseTres_req_InterfaceUm
        PROVIDED
            InterfaceUm_pro_ClasseTres
    Connector_2
        REQUIRED
            ClasseCinco_req_ClasseQuatro
        PROVIDED
            ClasseQuatro_pro_ClasseCinco
    Connector_3
        REQUIRED
            ClasseCinco_req_ClasseTres
        PROVIDED
            ClasseTres_pro_ClasseCinco
    Connector_4
        REQUIRED
            ClasseQuatro_req_ClasseDois
        PROVIDED
            ClasseDois_pro_ClasseQuatro
```

Figura 12: Arquitetura gerada após a separação dos dois componentes
Fonte: Autoria do autor

O próximo exemplo de modificação da estrutura será unir dois componentes em um. A estrutura que vai ser utilizada vai ser a estrutura que foi gerada a partir do último exemplo.

Neste exemplo será feita a união dos componentes.

Será utilizado o objeto do tipo `StructureManager` que foi criado anteriormente. Será executado o método `mergeTwoComponents`, passando por parâmetro ao método os componentes que serão unidos e a estrutura que será modificada, como mostrado na Figura 13.

```
50          structure = sm.mergeTwoComponents(structure, componentDoispt2, componentTres);
```

Figura 13: Código responsável por realizar a união de dois componentes
Fonte: Autoria do autor

Após a execução do código, o sistema irá gerar a nova estrutura removendo os componentes que foram passados ao método, e então criar um novo componente contendo os dados dos dois componentes. A Figura 14 representa a estrutura obtida após a execução do código.

O último exemplo contemplará as três últimas funcionalidades de gerenciamento da estrutura. Neste exemplo voltaremos a utilizar a estrutura original do exemplo, isso apenas para ser mais fácil de se identificar as modificações feitas na estrutura. Assim sendo, o primeiro passo deste exemplo está em criar um novo componente, que neste caso irá ser chamado de `ExemploAddComponent`. Após ter criado o novo componente será a hora de mover classes para esse novo componente, desse modo iremos mover às classes: `ClasseUm`, `InterfaceUm`, `ClasseDois` e `ClasseQuatro` ao novo componente. Logo em seguida será feita a exclusão do componente que ficou sem nenhuma classe que no caso foi o `ComponenteUm`. A Figura 15 representa o código responsável por realizar o processo citado acima.

Com a execução do código podemos ver a nova arquitetura gerada pelo sistema. A Figura 16 representa a saída obtida. Nela podemos validar que o sistema conseguiu realizar todas as operações de acordo com o solicitado pelo usuário de forma adequada.

ARQUITETURA GERADA APÓS A UNIÃO DOS COMPONENTES

```
COMPONENTES
    ComponenteUm
        CLASSES
            ClasseUm.java
            InterfaceUm.java
        PRO INT EXT
            InterfaceUm_pro_ClasseTres
        PRO EXT EXT
            ClasseUm_pro_ClasseDois
        PRO REF EXT
    ComponenteDois_pt1
        CLASSES
            ClasseTres.java
        REQ INT EXT
            ClasseTres_req_InterfaceUm
        PRO REF EXT
            ClasseTres_pro_ClasseCinco
    ComponenteDois_pt2_merge_ComponenteTres
        CLASSES
            ClasseDois.java
            ClasseCinco.java
            ClasseQuatro.java
        REQ EXT EXT
            ClasseDois_req_ClasseUm
        REQ REF INT
            ClasseCinco_req_ClasseQuatro
            ClasseQuatro_req_ClasseDois
        REQ REF EXT
            ClasseCinco req ClasseTres
        PRO REF INT
            ClasseQuatro_pro_ClasseCinco
            ClasseDois_pro_ClasseQuatro

CONECTORES
    Connector_0
        REQUIRED
            ClasseDois_req_ClasseUm
        PROVIDED
            ClasseUm_pro_ClasseDois
    Connector_1
        REQUIRED
            ClasseTres_req_InterfaceUm
        PROVIDED
            InterfaceUm_pro_ClasseTres
    Connector_2
        REQUIRED
            ClasseCinco_req_ClasseQuatro
        PROVIDED
            ClasseQuatro_pro_ClasseCinco
    Connector_3
        REQUIRED
            ClasseCinco_req_ClasseTres
        PROVIDED
            ClasseTres_pro_ClasseCinco
    Connector_4
        REQUIRED
            ClasseQuatro_req_ClasseDois
        PROVIDED
            ClasseDois_pro_ClasseQuatro
```

Figura 14: Arquitetura gerada ao unir os componentes
Fonte: Autoria do autor

```
Component newComponent = new Component();
newComponent.setName("ExemploAddComponent");
structure = sm.addComponent(structure, newComponent);//adiciona novo componente
structure = sm.moveClass(structure, newComponent, classeQuatro);//move ClasseQuatro
structure = sm.moveClass(structure, newComponent, classeDois);//move ClasseDois
structure = sm.moveClass(structure, newComponent, classeUm);//move ClasseUm
structure = sm.removeComponent(structure, componentUm);//remove ComponenteUm
```

Figura 15: Código responsável por criar um novo componente, mover classes a um outro componente e remover um componente
Fonte: Autoria do autor

ARQUITETURA GERADA APÓS AS OPERAÇÕES EXECUTADAS

```
COMPONENTES
    ComponenteDois
        CLASSES
            ClasseTres.java
        REQ INT EXT
            ClasseTres_req_InterfaceUm
        REQ EXT EXT
        REQ REF EXT
        PRO INT EXT
        PRO EXT EXT
        PRO REF EXT
            ClasseTres_pro_ClasseCinco
    ComponenteTres
        CLASSES
            ClasseCinco.java
        REQ INT EXT
        REQ EXT EXT
        REQ REF EXT
            ClasseCinco_req_ClasseQuatro
            ClasseCinco_req_ClasseTres
    ExemploAddComponent
        CLASSES
            ClasseDois.java
            ClasseQuatro.java
            ClasseUm.java
            InterfaceUm.java
        REQ INT INT
        REQ EXT INT
            ClasseDois_req_ClasseUm
        REQ REF INT
            ClasseQuatro_req_ClasseDois
        PRO INT INT
        PRO INT EXT
            InterfaceUm_pro_ClasseTres
        PRO EXT INT
            ClasseUm_pro_ClasseDois
        PRO EXT EXT
        PRO REF INT
            ClasseDois_pro_ClasseQuatro
        PRO REF EXT
            ClasseQuatro_pro_ClasseCinco

CONECTORES
    Connector_0
        REQUIRED
            ClasseTres_req_InterfaceUm
        PROVIDED
            InterfaceUm_pro_ClasseTres
    Connector_1
        REQUIRED
            ClasseCinco_req_ClasseQuatro
        PROVIDED
            ClasseQuatro_pro_ClasseCinco
    Connector_2
        REQUIRED
            ClasseCinco_req_ClasseTres
        PROVIDED
            ClasseTres_pro_ClasseCinco
    Connector_3
        REQUIRED
            ClasseDois_req_ClasseUm
        PROVIDED
            ClasseUm_pro_ClasseDois
    Connector_4
        REQUIRED
            ClasseQuatro_req_ClasseDois
        PROVIDED
            ClasseDois_pro_ClasseQuatro
```

Figura 16: Arquitetura gerada após a excução do código
Fonte: Autoria do autor

4 CONCLUSÕES GERAIS

Para um desenvolvedor ou pesquisador compreender um sistema que não possua uma documentação adequada é complicado, uma vez que nem sempre quem desenvolveu o software está disponível para ajudar a compreender o sistema. Uma possível solução está na utilização de ferramentas que sejam capazes de fazer um processo de engenharia reversa e obter a arquitetura do software, para então estudar como o software funciona. Porém, a maioria das ferramentas disponíveis no mercado não são capazes de realizar o gerenciamento da estrutura gerada, obrigando o usuário a utilizar outras ferramentas para gerenciar a estrutura.

A ferramenta apresentada neste trabalho de conclusão foi criada para possibilitar que o usuário possa obter a arquitetura de um software e ao mesmo tempo realizar as modificações que achar necessárias na estrutura para seus estudos, ou projetar implementações ao sistema sem causar dano a arquitetura do software.

A contribuição deste trabalho está no fato de que ele é capaz de obter a estrutura de um programa Java diretamente do código fonte, e ainda modificar a estrutura de acordo com sua necessidade. Permitindo que vários estudos sobre a estrutura do software possam ser realizados.

Com essas funcionalidades implementadas é possível solucionar os problemas encontrados no início do texto. Logo que a ferramenta não limita os dados extraídos já que o usuário é capaz de modificar a estrutura da forma que desejar. O problema de não realizar a representação ideal foi solucionado com a capacidade do sistema fazer a análise de todas as dependências existentes no código fonte.

Por outro lado, a ferramenta apresenta algumas limitações. A principal das limitações está no fato dela não fazer uma representação gráfica da arquitetura gerada, uma vez que ela só faz extração virtual, ou seja realiza modificações na arquitetura somente em memória. Outra limitação é que o sistema no momento é capaz de importar a estrutura somente de código fonte `Java`, assim impedindo o usuário de utilizar a ferramenta em software desenvolvidos em outras linguagens.

Como trabalhos futuros, pretende-se fazer com que a ferramenta seja capaz de tra-

balhar com outras linguagens de programação que sejam utilizadas pelo mercado atual, além de implementar novas funcionalidades na ferramenta, como por exemplo o estudo de métricas a partir da arquitetura que foi obtida da extração. Também pretende-se criar um módulo de exportação da estrutura para um novo projeto. Assim, o usuário seria capaz de criar um novo projeto com a nova estrutura.

REFERÊNCIAS

ABI-ANTOUN, M.; ALDRICH, J. A field study in static extraction of runtime architectures. In: **Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering**. New York, NY, USA: ACM, 2008. (PASTE '08), p. 22–28. ISBN 978-1-60558-382-2. Disponível em: <http://doi.acm.org/10.1145/1512475.1512481>.

ALDRICH, J.; CHAMBERS, C.; NOTKIN, D. Archjava: Connecting software architecture to implementation. In: **Proceedings of the 24th International Conference on Software Engineering**. New York, NY, USA: ACM, 2002. (ICSE '02), p. 187–197. ISBN 1-58113-472-X. Disponível em: <http://doi.acm.org/10.1145/581339.581365>.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3rd. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321815734, 9780321815736.

BRIAND, L. C.; LABICHE, Y.; LEDUC, J. Toward the reverse engineering of uml sequence diagrams for distributed java software. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 32, n. 9, p. 642–663, set. 2006. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/TSE.2006.96>.

CHARGO, J. T. Automated software architecture extraction using graph-based clustering. 2013.

EILAM, E. **Reversing: secrets of reverse engineering**. [S.l.]: John Wiley & Sons, 2011.

HOFMEISTER, C.; NORD, R. L.; SONI, D. Describing software architecture with uml. In: **Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)**. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 1999. (WICSA1), p. 145–160. ISBN 0-7923-8453-9. Disponível em: <http://dl.acm.org/citation.cfm?id=646545.696368>.

JACKSON, D.; WAINGOLD, A. Lightweight extraction of object models from bytecode. In: **Proceedings of the 21st International Conference on Software Engineering**. New York, NY, USA: ACM, 1999. (ICSE '99), p. 194–202. ISBN 1-58113-074-0. Disponível em: <http://doi.acm.org/10.1145/302405.302465>.

KNODEL, J. et al. Static evaluation of software architectures. In: **Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on**. [s.n.], 2006. v. 00, p. 10 pp.+. Disponível em: <http://dx.doi.org/10.1109/CSMR.2006.53>.

KNODEL, J.; POPESCU, D. A comparison of static architecture compliance checking approaches. In: **Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture**. Washington, DC, USA: IEEE Computer Society, 2007. (WICSA '07), p. 12–. ISBN 0-7695-2744-2. Disponível em: <http://dx.doi.org/10.1109/WICSA.2007.1>.

KOBAYASHI, K. et al. Sarf map: Visualizing software architecture from feature and layer viewpoints. **CoRR**, abs/1306.0958, 2013. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1306.html/KobayashiKYKM13>.

KRUCHTEN, P. The 4+1 view model of architecture. **IEEE Softw.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 12, n. 6, p. 42–50, nov. 1995. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/52.469759>.

MARTENS, A. et al. A hybrid approach for multi-attribute QoS optimisation in component based software systems. In: HEINEMAN, G.; KOFRON, J.; PLASIL, F. (Ed.). **Research into Practice - Reality and Gaps (Proceedings of the 6th International Conference on the Quality of Software Architectures, QoSA 2010)**. Springer-Verlag Berlin Heidelberg, 2010. (Lecture Notes in Computer Science, v. 6093), p. 84–101. Disponível em: <http://sdqweb.ipd.kit.edu/publications/pdfs/martens2010b.pdf>.

MURPHY, G. C.; NOTKIN, D.; SULLIVAN, K. Software reflexion models: Bridging the gap between source and high-level models. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 20, n. 4, p. 18–28, out. 1995. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/222132.222136>.

NELSON, M. L. A survey of reverse engineering and program comprehension. **arXiv preprint cs/0503068**, 2005.

RAMÍREZ, A.; ROMERO, J. R.; VENTURA, S. An approach for the evolutionary discovery of software architectures. **Information Sciences**, Elsevier, v. 305, p. 234–255, 2015.

RIVA, C.; RODRIGUEZ, J. V. Combining static and dynamic views for architecture reconstruction. In: **Proceedings of the Sixth European Conference on Software Maintenance and Reengineering**. Washington, DC, USA: IEEE Computer Society, 2002. (CSMR '02), p. 47–. Disponível em: <http://dl.acm.org/citation.cfm?id=872753.873546>.

SHAW, M.; GARLAN, D. **Software Architecture: Perspectives on an Emerging Discipline**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN 0-13-182957-2.

YAN, H. et al. Discotect: A system for discovering architectures from running systems. In: **Proceedings of the 26th International Conference on Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2004. (ICSE '04), p. 470–479. ISBN 0-7695-2163-0. Disponível em: <http://dl.acm.org/citation.cfm?id=998675.999450>.