FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER’S THESIS

Title:	Reverse Engineering of Legacy Software Code for Normalized Systems Exanders
Student:	Bc. Veronika Larionova
Supervisor:	Ing. Robert Pergl, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2018/19

Instructions

The goal of the thesis is to explore the possible reverse engineering approaches on legacy software code in the context of Normalized Systems Expanders (NSX) specification and formulate the conclusions.

1. Perform a review of the state-of-the-art of legacy software code reverse engineering approaches.
2. Select a suitable solution for the goal of the thesis.
3. Analyse the possible applications of the selected solution on the legacy codebase for purpose of extracting the input information into the Prime Radiant.
4. Formulate the conclusions of the analysis.
5. Discuss capabilities and limitations of the proposed solution and formulate future work.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 15, 2018

Master's thesis

Reverse Engineering of Legacy Software Code for Normalized Systems Exanders

Bc. Veronika Larionova

Department of Web and Software Engineering (specialisation Software Engineering)
Supervisor: Ing. Robert Pergl, Ph.D.

June 27, 2018

Acknowledgements

Having the opportunity, I would like to express my gratitude to the people who aided me in improvement of this thesis, namely my supervisor, Robert Pergl, Jan Verelst, the Pharo team from the University of Lille, especially Stéphane Ducasse and Nicolas Anquetil, and also Peter Uhnak and Jan Bliznišenko.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on June 27, 2018 . . . .

Czech Technical University in Prague
Faculty of Information Technology
© 2018 Veronika Larionova. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Larionova, Veronika. *Reverse Engineering of Legacy Software Code for Normalized Systems Exanders.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Daná práce se zabýva pruzkumem možností, jak lze z legacy kódu získat vstupy pro NS expandery, za účelem jeho modernizace a reimplementace do normalizované podoby. Práce zkouma existující přístupy reverzního inženýrsví pro oblast extrakce byznys pravidel z již existujicích systému. Následně se evaluují možností použití existujicích nástrojøu pro experimentální ověření, zda lze docílit generické automatizace extrakce dat z kódu. Na základě této fáze se formulují závěry a možná doporučení pro následné kroky.

Klíčová slova Reverse engineering, Normalizované softwarové systémy, extrahováníbyznys logiky, legacy kód, re-engineering.

Abstract

This thesis is meant as an exploration of possible usages of reverse engineering techniques and tools for purpose of obtaining the inputs for Normalized Systems expanders to re-engineer the legacy software system into its normalized version. During the exploration the attention is to be paid to the techniques of reverse engineering and its application in already existing case studies. Additionally, the survey of the existing solutions and tools is provided. Moreover, the thesis includes the experimental part, where the reverse engineering tool

(namely Moose) is used to gain the overview on the code and see the possibility of using in for acclimatization of the process. Finally, the work is concluded with the advises for future work.

Keywords Reverse engineering, Normalized software systems, business logic extraction, legacy code, re-engineering.

Contents

Introduction **3**
Formulation of the goals 4

I State of the Art exploration **5**

1 Normalized Software Systems **7**
1.1 The theoretical background 7
1.2 NS elements 10
1.3 Element expansion 15

2 Reverse engineering and re-engineering **17**
2.1 Re-engineering process 18
2.2 Reverse engineering and its approaches 18
2.3 Business rules mining 23
2.4 Reverse engineering solutions and tools 31

II Application of research **39**

3 Introduction of sample projects **41**
3.1 Project 1: Student Information System 1 [1] 41
3.2 Project 2: Student Informational System 2 42
3.3 Project 3: Student Management System [2] 43
3.4 Project 4: Ticket selling system 44
3.5 Evaluation of the projects 45

4 Experimental phase **47**
4.1 Exploring the projects 48
4.2 Reflections 50

5 Further directions and proposals **53**

Conclusion **55**

Bibliography **57**

A FAMIX **63**

B Projects' UML **65**

C Contents of CD **69**

List of Figures

2.1 Re-engineering process 18
2.2 System's To Be BR 24
2.3 Comparsion Modelware to Grammarware 28
2.4 Moose Workflow representation 32

3.1 Student Information System 2 GUI 42
3.2 Student management system DB schema 43
3.3 Project 5: Ticket Systems's database model 45

A.1 FAMIX simplified overview 63
A.2 FAMIX hierarchy 64

B.1 Project 1: Student information system UML 66
B.2 Project 2: Student information system 2 UML 67
B.3 Project 3: Student Management System UML 68

Introduction

These days the term *legacy system* is used relatively often. By definition it refers to “outdated computer systems, programming languages or application software that are used instead of available upgraded versions.”[3] However, outdated does not only have to be associated with age. It also may refer to it maintainable state. In many situations happen, that even the system developed by relatively recent technologies, can become unmanageable through time.

The developers, who worked on the project, may leave, the newly introduced changes does not have to be reflected in the documentation. Of course, the aging criteria is also playing its part. The older technologies does not have to show a good performance metrics. Also, through years the older legacy components are tangled in vast amount of integrations with other different systems (e.g. this can be the case for situation like exposing the application interfaces to their services through internet). The list of similar issues can go even further.

No wonder, that for more than 25 years parallel to this exists the discipline which has aimed at understanding of the software artifacts. This discipline is called Reverse engineering (see 2). This was inspired by the ideas from the area of hardware engineering and tried to tailor them for the purposes of software. Through years the approaches evolved and some of the process that previously were done manually, became automatized.

Later, the reverse engineering techniques started to being used as a source of business information for rejuvenated systems. The major migration started when the concept of Object Oriented programming was introduce. At that time there was a motion to move less comprehensible programs in COBOL or Fortran to more modern technologies as Java or C++. Even now the tendencies of software modernization take place. The great stress is put on the idea of new and more maintainable software.

The other concept, which is to be presented in this thesis, is called Normalized System Theory and its application in Normalized Software Systems (see 1).

It strives to define the criteria for the system, that can be evolvable through time without uncontrollable propagation of the changes through it.

Formulation of the goals

The thesis tries to explore the possibilities of the reverse engineering part of re-engineering process, where the forward engineering will be cover by already existing tools for NS generation. The aim is too see, to what extend it is possible automatize the extraction of the information from the legacy code base for purpose of using it as an input for NS expanders.

As of now, the process of creating the Normalized software systems is starting from the beginnin (anew)g. The qualified business analysts does the requirements collections and based on it designs the inputs to the generator. As stated in [4] the process of analysis takes time, however, the implementation part is much faster then in cases of fully manual development. The thesis strives to come up with possible improvements of this by introducing the reuse of the legacy code through reverse engineering.

Part I

State of the Art exploration

Chapter 1

Normalized Software Systems

There is no one who can disagree, that these days the big advantage of the company is that it is able to adapt to changes as fast as possible. Everything around is changing really swiftly and it is ought to be reflected. However, adapting to the innovations is not that easy, especially, when this changes concern a lot of ares. One of such an area is a software which supports business processes.

Evolvablity of software is an importing topic already for a long time. Many guidelines and patterns appeared to provide the steps for developer to create the product, that can be changed more easily. For example in recent article [5] the author mentions 5 good practices to follow in order to obtain an evolvable system, e.g. encapsulation and clear concepts, analyzability of the code (for example by using some specific tool) or maintain process of changes. However, this advises are not always adhered and the software just gains the small "holes" that can in time become bigger. What's more, there is no guaranty, that even by following the rules, in future there would be no need to make changes, that can lead to big impacts, like the change of interface of the library that is used in code.

Normalized software (NS) theory [6] provides the other way, how to tackle the issues of system changes in time. The theory gives the strict guidelines, that lead to *evolvable modularity.* It provides the theoretical proven foundations and also the practical application in software engineering.

1.1 The theoretical background

One of the building stones of NS theory is notion of system stability (stability of the model) which corresponds with the idea of evolvable system. The generic concept is defined as *BIBO*(Bounded Input Bounded Output). When applied to software systems it results into the fact, that if we have some bounded set off additional functionalities on existing system, it should result into bounded number of changes applied to the system primitives. If then taking in account

the unlimited system's evolution the impact of changes on the system should never be related to the size of the system (expected to be constant as the system gets bigger).

The possible instabilities, that breach the previously mentioned impact limitation, are called *combinatorial effects*. They are unwanted features that represent a danger for evolvability. The example of such undesirable situation is when we would like to enhance a simple record about a person with additional information like height. Assume, that we have some actions that are dependent on the item under change. These actions will also require new attribute for computation. Then the impact itself is not only on the record of the person, but also on the actions that process the person record.

As the result of analysis of the system stability principle four main theorems were stated. They provide more stricter rules to abide in time of development compared to previously mentioned best practices for software development. What's more, if atleast one of the rule is broken in any point in development, the combinatorial effect will take place. It is worth noticing, that these rules have existed already before.

1.1.1 Separation of Concerns

The first theorem concerns the the implementation of the tasks. Ideally the task should only have one purpose, that it serves for, e.g. reading a file, converting from one format to another . . . .

Theorem 1. *A processing function can only contain a single task in order to achieve stability.[6]*

If the rule is broken and possibly one function has atleast two tasks, then it can be assumed, that, when later there will be more versions of the function, the next change will have to pass through all the versions and take effect. The situation like this leads to unbounded impact.

1.1.2 Data Version Transparency

The way, how data are passed through the function, is important. As example the data structure university Course can be updated by a new parameter like credits that are based on new standard proposed by EU. Then there could be a task processing this structure. If only some required attributes of the structure are passed (so called data coupling), then it is necessary to update all versions of processing task, which will require the new parameter, too. So, the change of one argument brings the complication of updating the function. However, if passing the whole Course structure is considered, than the addition of new attribute is only done in structure (so called stamp coupling) and function interface does not have to be changed.

Theorem 2. *A data structure, that is passed through the interface of a processing function, needs to exhibit version transparency in order to achieve stability.[6]*

One of the most common manifestation is the idea of encapsulating the properties of the object and make them accessible through set/get interface.

1.1.3 Action Version Transparency

If there are function that uses (calls) the other function, the change of the interface of second function has an impact of the first one.

Theorem 3. *A processing function that is called by another processing function, needs to exhibit version transparency in order to achieve stability.[6]*

One of the solution, how to tackle the problem, is to have a wrapper that allows the encapsulation of the change-driver. In languages like Java the action version transparency can be achieved by usage of interfaces.

1.1.4 Separation of States

Here the adhering to the previous two theorems can be assumed. It is not necessary to concern with the changes of interfaces, however, the new implementation of a task, can result in new error state. Then this change impacts the transition between calls. As the authors of NS theory propose, this situation can be dealt with by introducing error keeping state. In this situation there can exist a function responsible for processing the error keeper. The caller then does not have to be concerned with error handling and can pass the responsibility to dedicated function.

Theorem 4. *Calling a processing function withing another processing function needs to exhibit state keeping in order to achieve stability.[6]*

The possible encounters of application of this principle are asynchronous systems, for ex. javascript promises. The implication of this theorem is so called *stateful workflow*. The main idea is the presence of state of the function and system overall. The state is a combination of the data that come as input and the actual result. The state is something which is used to pass through the workflow.

As authors mentions, this restriction could resemble the assembly line. However, this view is very tricky. If implemented in software, it can result to the chain of function invocation. Regretablly , they are not independent on each other if executed in synchronous manner, because the caller is ought to wait for competition of callee. However, when there is a presence of *controller*, which orchestrates whole process (calling functions and passing the state), then the coupling like in caller-callee relationship is no longer present.

1.2 NS elements

By applying the theorems to the system design the very fine-grained structure is reached. They adhere to the rules, however, it is hard to have overview over them, because their numbers can be quite high. Due to this reason, the small modules can be grouped to *elements* – an abstract group, which combines the modules that serves for specific purpose. For the purposes of software information systems (as one possible manifestation of NS theory) the five elements were introduced. The encapsulate the anthropomorphic representation withing the modules, which are responsible for crosscutting concerns, such as persistency in case of data element. These elements are:

- data element – a data version transparent software element with accessors to its attributes,
- action element – a single processing function containing one task,
- workflow element – represents a sequence of actions, that are executed in order,
- trigger element – the element, which is meant for checking the states of the system and triggering actions accordingly (clock-like control),
- connector element – construct, which serves as the connection to external systems in stateful manner.

They represent the patterns, that are further used to design a new normalized software system. As the proof of concept the whole structures were realized using Java EE technology. In next subsections the elements will be presented more closely.

1.2.1 Normalized element in general

The aim of this section is to provide a overall ideas, which are connected to elements in general. At the beginning the description of the idea of the recombination of modules will be presented, followed by ideas of more closer view on elements' representation from the point of the view of the projection of a real word part to the programmable structure.

1.2.1.1 Modular structure

Following the NS rules results into a big amount of really small modules. As authors of concept state: "This splitting into small modules is the only way to allow for large numbers of possible system variations, while (at the same time) only having limited numbers of modules to develop and maintain."[6] The simple view is that the exists a modules which implement some unit work and this modules can have some variants k. One can recombine this

modules – versions to obtain a new product. By following simple calculations it is possible by having $\sum k_i$ module versions to obtain $\prod k_i$ system variants. Which represents a big gain, however, if consider more monolithic structure, where module contains more unit of work, then if one unit undergoes a change it result into the need to construct whole new module. This results into having the same amount of module versions as system variants.

One can argue, that modularity is already a used practice. However, it is important to see the granularity. If a building block – module is for example a whole room (e.g. bathroom, bedroom, 20 x 20 meter room, ...), then by recombining them, it is possible to obtain a flat or home. Nevertheless, the emerged building can be a monster if the rooms do not pass one to each other. Apparently such modules are not evolvable. The reason for it is very simple, they are not highly structured. If the module is a piece of wall with all needed integrated (like pipes for water, electricity cables, etc.), then it will be possible to build whatever structure needed.

On the other hand, if we make a change in one small module it can cause a ripple-effects to others. This point leads to the fact that modules should be encapsulated to stop any propagation.

1.2.1.2 Element's general structure

Element itself represent some reoccurring structure which is consists of an essence (noun – data entity or verb – processing action) from the real world) and its cross-cutting concerns (e.g. persistency, presentation, remote access). For clearer explanation the simple example can be a data element Order. From real word terminology it will be projected to an actual representation of order with it properties as attributes, but additionally if also gains the set of modules like database connectors or remote accessors. Later any change that will be needed on Order will be reflected in order representation and also can affect the its cross-cutting concern modules.

Noun Transformation - NounDetails The core of the transformation of the data is the actual content without any connection to any other additional concerns. The object that resulted from the transformation is referred as *NounDetails*. Because it contains just needed data, it can be easily passed to the processing function.

Verb transformation - VerbImpl As it was in case of the noun, the verb also should have a version transparent representation which is *VerbImpl*. The representation is ought to contain the implementation of the verb which takes *NounDetails* and process it. As the result the output class is returned. To ensure the Action Version Transparency there should be a class defining the interface which is later to be implemented by all versions of the verb implementation.

Adding context Except the needed attributes for nouns or implementation code for verb the "classes" can also include the *context*.

- user context – e.g. information about the user accessing the web (name, id , automatic user);
- processing context – the information that affects the processing process (implementation of the task, parameters to use, . . .);
- error context – can include the type and/or detail of the error.

1.2.2 Data element

As mentioned before, the element for data is meant for storing and exhibiting the actual data (e.g. Order). The core consists of the transformation for *NounDetails* and it is enhanced by CRUD functionality (with also possible other additional modules for cross-cutting concerns). Except the whole detailed view on the data object it is also possible to define the *projections* which will only hold the attributes that can be relevant for specific presentation (*NounInfo* class).

The actual implementation can be divided into layers. Each of the layer is dependent on external technology (e.g DB drivers for persistance). These layers later are to use the interfaces to provide the communication between them (e.g. *Noun Remote*). The layers to be distinguished are :

- data layer – it hold the responsibility of taking care data persistency and retrieval.To have the technology agnostic *NounDetails* class the data classes are created to represent the data entity that will be connected with database and later the action classes are used for any needed CRUD manipulation (for ex. NounDataFinder).
- logic layer – is responsible for business logic and transaction.
- remote client/proxy server – provide the availability of remote calls to the data elements. This layer can for example use existing technologies such as RMI (Remote Method Invocation) or Web Services

1.2.2.1 Data element and domain entities

The data element itself is just a template, which after it is provided a parameters, reflects the real life object. While the exploring the problematic, the "domain model' is built.

The model covers the just the data part without any additional features, like helper or behavior. The whole process is not dependent on any methodology or notation, however, to make the analyzing steps easier, the whole structure of the domain entities can be divided into categories:

- **Core** – refers to the essential entities which are important part of the organization.
- **Taxonomy** – describes the entities that has the purpose of classification, e.g. Brand, Color, ...
- **Directory** – stands for the elements which symbolize the location of core entity (even hierarchical).
- **Domain** – covers the data that are typical for domain (and shared across organization). This data can be known and to some extent constant (for ex. list of countries – Country).
- **Reusable domain components and integration entities** – corresponds to the entities which can be reused between more systems. They can be gathered into separate unit (e.g. CRM system), which later can be connected via connectors.
- **Other** – this category can have the entities representing cross-cutting concerns or even entities which help to divide m:n relationship. The entities mostly do not have to be the part of Normalized system analysis.

Identification of data element in general can be seen as four-step-process. During the beginning the basic specification is defined – the element itself, its attributes and relations. After it the additional concerns are specified. This is later followed by selecting the tasks and flows for each data element. Finally, the stating of any customizations takes place.

1.2.3 Task element

The role of the element is a representation of one simple task (a Verb) without any internal workflow inside. Except the main function it also contains a needed cross-cutting concerns modules. Similar to data element the structure can also be divided into the 3 layers.

The core is represented by *VerbImpl*, which due to following Action Version Transparency rule is encapsuled into the interface. Moreover, if needed the other interface is also provided to allow remote access to the task.

1.2.3.1 Analysis of the tasks from domain

In the first two steps of the analysis of data entities (possible data elements) the elements specification and additional requirements are discovered. However, they are to some point just the data which can be processed. The processing part is covered by the task. The task itself is just one processing activity that takes the data element and provides the output. On other hand, not all data are expected to have the behavior. In majority the features of behavior are

the trait of core elements. On contrary, elements such as domain ones do not expose any changes through their lifecycle and consequently do not need to have any task attached to them.

The tasks could be divided into types:

- **Standard** – in case of such a task the system executes some action.
- **Bridge** – during its execution the new data elements are created, which then will pass through their own workflow.
- **Manual** – requires the user input which will result in state directing the wrokflow processing.
- **External** – expects the external process (e.g. from another system) to preform the action, which will lead to a change of the state of some core element.
- **Branching** – the output of the task is more than one state. Here the task is directing the actual branching.
- **Update** – refers to bridge task and holds the purpose of providing the update to the parent, which is participating in creation of "children" elements.
- **Spawning** – is meant for tasks, that are to be heavy on execution time and would require a separate thread for processing.

1.2.4 Flow element

The Flow element holds the responsibility of the orchestration/sequencing of the tasks. Its inner structure contains various elements, such as the task elements, that are sequenced, or data elements that are processed by tasks. One of the contained groups is *configuration elements*:

- Workflow – the data representation of a workflow
- StateTask – the representation of specific task which is to be executed in specific state
- FlowTaskStatus – the entry of the log table stating the task creation and execution

1.2.4.1 Analysis of the workflow

The execution of the workflow was briefly described in Separation of States theorem (1.1.4). The idea lies in the execution based on the states, where the actual state triggers the expected task. The workflows can be dependent on the each other. As in cases of bridge or update task. Here the task of

one data elements starts the workflow of the other or provides the trigger for proceeding in execution.

1.2.5 Trigger element

The Trigger element is a part of which helps to orchestrate workflow. Based on the events on application server the trigger evokes the appropriate flow element. Similarly to the workflow element it also has the configuration data elements:

- TimeEvent – describes the data element holding the actual event
- TimeWindow – holds the time period information about the period when engines are to do their work
- EngineService – represent the running engine with it control parameters

1.2.6 Connector element

The Connector element represents the input/output connection. This element can be of two types:

- user connector – for user interfaces
- service connector – for external application systems

Its functionality is provided both to the data or to the task element. The trait of layer division is again presented here. The first layer is a *control layer* where the routing of incoming request to appropriate method of application class (proxy layer) is taking place. For providing the session-aware remote access 2 types of classes are introduced: *NounAgent* and as for verb *VerbAgent*. With their help the user-session identity is later passed to the proxy classes and to the business logic server. The responsibility of presenting the element falls onto the *View layer*.

1.3 Element expansion

Each element represents some relatively general structure (pattern) that has a purpose. In real application these elements will be given more direct purpose by specifying attribute, parameters, task behavior, . . . which is described in requirements. The term to refer to this is *expansion*. During it the elements are created from pattern by the generators – *expanders*. The expanders preserve the pattern structure of the code that adheres the the rules 1.1.

For the purposes of the expansion process the specific software tools were developed.[7] These tools are responsible for generating the actual code based on the structured input. They consider three basic hierarchical structures:

- NSX elements – the smallest possible aggregation which were presented previously in 1.2.1;
- components – the aggregation of elements based on the functional purpose;
- application – the construct which wraps more components.

The parametrization itself is done with help of "descriptor files" – the XML-like structured files that contains the values to instantiate the element, e.g. name of the data element, its relationship to other elements, ...[8]

After the expansion to generated structures it is need to tailor the code fit into the requirements (e.g. extending visual presentation of the page). The enhancement is done with the help of plugn-in code, which can be represented as separated classes – extensions or just code parts which are placed between specified anchors – insertions. However, for making the code easier to evolve it is needed to have the plugin-in code separated from generated skeleton. The process of getting the plug-in code for separated storing is called *harvesting*. Afterwards, the elements are generated and harvested code is injected into the obtained skeleton. Although such a scenario is only possible in case the inserted code has the interface, which is not dependent on specific technology and preserves version transparency. Some of the customization however can be also incorporated in general structures.

Chapter 2

Reverse engineering and re-engineering

The reverse engineering is the term that already exists couples of decades, although, the use of the term originally came from the area of hardware engineering. On the other hand, the idea of getting to understand the already existing product could undoubtedly be presented even in the area of software engineering. One of the most prominent person in the beginning of formulating the idea of software reverse engineering was E.J.Chikovksy. In his work [9] he provided the formal definition of it: "the process of analyzing the subject systems to

- Identify the system's components and their interrelationships and
- Create the representations of the systems in another form or at higher level of abstraction."

However, nowadays the reverse engineering is not only connected with the idea of documentation and maintenance, but also refers to the things like [10]:

- the recovering of architecture and design patterns,
- identification of reusable assets,
- change impact computation
- migration towards new architectures and platforms, ...

In the end the reverse engineering still covers the part of understanding the system, but as the times progrees and technologies become outdated and unsupported the need of other discipline have arrisen which is called *re-engineering*. As stated in [11] : "re-engineering is the examination and alternation of a system to reconstitute it in a new form and the subsequent implementation of new form. Thus, re-engineering generally includes reverse

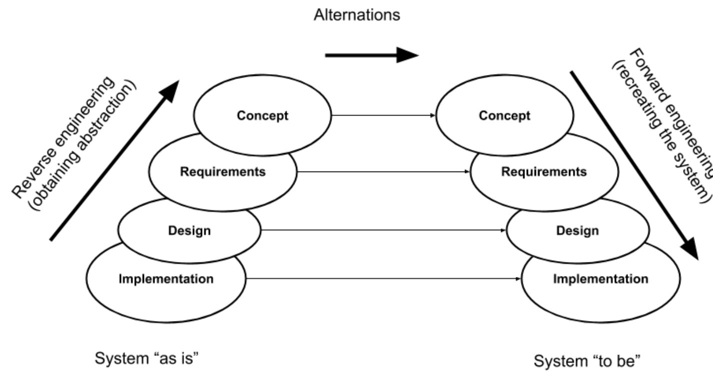

Figure 2.1: The process of re-engineering (inspired by [12])

engineering. (to achieve a more abstract description) followed by forward engineering or restructuring."

Through this chapter it will be aimed to create the overview of the usage of the reverse engineering in the area of re-engineering. Firstly, the introduction to the reverse engineering approaches will be provided. This will follow with the one of the domain where it is actually applied – the business rule and logic mining. The chapter will be then concluded with the presentation of the existing software tools or projects that support reverse engineering processes.

2.1 Re-engineering process

As stated previously, the re-engineering process is a combination at first the understanding of the system, represented by reverse engineering. Additionally, after obtaining the higher level of abstraction and overview of the system further reorganization and improvements could take place. Finally, the forward engineering takes place.

In [12] the author describes the process of the re-engineering (depicted on 2.1). The starting point of it is the legacy artifacts of the system that exists. After with the process of reverse engineering the abstraction is obtained. This level allows easier feature manipulation and changes. After what the forward engineering process can start.

2.2 Reverse engineering and its approaches

The reverse engineering itself represents quite a broad term with many ways how it can be approached. Traditionally, it was seen as the process with two

phases: the first one responsible for information extracting and the second one for it abstraction.[13]

However, this term does not have to be solely connected only with the area of software comprehension for purposes of documentation and maintenance, but also is more often know in the area of software security. In that cases the idea remains the same – the (ethical) hackers try to understand the software and its weak areas and at the same time programmers understanding the software can provide more security measures.

The process itself strongly depends on the technologies that the software uses, be it the implementation language, storage medium, software architectural style or even application of programmers patterns. Even the availability of execution traces and logs can be the considerable point in the process of software exploration.

Following from the statements before, the approches to the process could be categorized to three basic categories ([14] and [15]):

- *static* – this category contain the approaches that expects the source artifacts like source codes as an input. As the first step, the process of parsing is executed over them. After it the "queries" can be executed over it to obtain additional information.

  The methods from this category allows to have the view on the structure of the system or even to some extend obtain the information about the system's architecture. Additionally, it is possible to obtain the code metrics such as methods invocation or the lines of code per method metric. The method also allow to obtain the picture on elements' relationships (inheritance, composition, usage of records/classes, etc.)

  However, the methods have also drawbacks. They can not cover the inspection of the behavior .In this cases the static analysis can be combined with different approaches. Moreover, the method depends on parsing and this fact implies the need of either tools that support specific grammar (provided for example in Extended Backus-Naur form (EBNF)). The examples of such tools are Moose [16] or MoDisco [17]. On other hand, instead of having the fulfledge parser it is also possible to have user defined parsing based on regular like syntax (e.g. Codiscent tools [18])

- *dynamic* – opposing to the structural inspection with static analysis, dynamic analysis provides the possibility of extracting the information about the behavior of the system. During the process the outputs from execution are used to obtain the execution trace. One of the manifestation of such a reverse engineering approach is debugging.

  In case of dynamic analysis the scenarios for execution are prepared. Sometimes also the test cases could be used for such purposes. After it the output traces are analyzed and desired features observed.

However, this approach is prone to be complicated for computation and the process itself can take a lot of time. Also the execution scenarios could not cover all features or are prone to cover to wide collection of them. In these cases it will lead in excessive time spent on the analysis.

- *hybrid* – two approaches mentioned previously have their positive sides but also the negatives. In some cases their combination can lead to more efficient and precise outputs. As the result, for example the static analysis can provide the hints of the feature location. This hints the can provide the valid restriction condition for further processing of the behavior traces in dynamic analysis.

The three mentioned categories are to some extent main categories to encounter. However, there is also another classification to mention – the *textual* analysis. To some extent it can seem to be some branch of static analysis, but in this case the main goal is not the abstraction of the code. Here the idea lies in the fact, that a feature in the code will have probably the similar identification. Later the code parts and comments could be mapped to it. The techniques that can be used for its purposed incorporates the pattern matching or natural language processing. Also the combination of textual analysis with static or dynamic can result in hybrid.

The other technique of reverse engineering is also historical analysis.[10] In this case the system is inspected through its changes during history. For this purposes the version control such SVN or GIT are used.

2.2.1 The techniques of reverse engineering

In the introduction to the section the basic categories were presented. The division into them is primary determined by what from the software is analyzed. Here will be provided a closer look on the concrete techniques that are used for reverse engineering.

The discipline of reverse engineering exists already for more than 25 years and during that time the aim or approaches to reverse engineering were changing majorly.[13] One of the need for reverse engineering took place when the need for refinement of the software arose. The requirement for it was especially high when object oriented languages started to overthrow procedural one. Then mostly transformation approaches were used. However the issue of adhering to new technology principles still remained. Additionally, the new software did not has to be very comprehensible or maintainable.

The other branch of reverse engineering was the architecture recovering. This researches was made to obtain the view on the components which was to allow better understanding of the system.

One of the area for reverse engineering was also data reverse engineering. In this area the accent is made on understanding the data structures (which

can be or do not have to be in database). Examination is dome on their structure and relationships (e.g. [19]).

The other interesting area is also binary reverse engineering. This techniques are applied in case of source code absence. This option is quite often used in the area of security.

2.2.1.1 Program slicing

One of the most popular technique for reverse engineering is program slicing. The usage of it spreads through majority of the history of code analysis (introduced in 1984 [20]). Through the years it gained some modifications and more heuristic were introduced to make the results more precise.

The essence of the idea that program can be to big and it is very complicated to analyze. However, if there is a smaller pice of it available (which includes the characterisitic that are under observation), then it is much easier to go through. In [21] the slicing is defined as: “Slicing, or code isolation is a method for extracting components of a program that are concerned with a specific behavior or interest such as a business rule or transaction.”

The initial input for the slicing process ins *slicing criterion* [20]. Originally defined as a pair $< i, v >$, where $i$represents the number of statement at which observation is to be conducted and v is a variables’ set which is to be observed. Then the term *slice* is described. It represents some unit of abstraction. The slice represents the part of the code which: should be a subset of the original program obtained by deletion of some parts and should show the same behavior as original system in the frame of the slicing criterion. However, to not stumble on the problem, that after deletion of some lines of code, that the slice becomes grammatically incorrect (will not adhere to the grammar of programming language), the flow graph is defined and all slicing processes are reflected on it.

This conventional idea of program slicing was later enhance in [22]. Where the slicing was not seen as a simple mean for debugging, but also as possibility of code comprehension and maintenance. They introduce so called *interface slicing* – the extraction the behavior (interface) of the modules. In this case the slicing criterion is the pair of the module (with code and interface specification) and the list of the behavior/interfaces that are needed.

In [23] the slicing is presented in two forms: *static* and *dynamic*. The static approach relies on Program Dependence Graph (PDG) - “directed graph with vertices corresponding to statements and control predicates, and edges corresponding to data and control dependencies.” The slice criterion is then a one vertex and the slice is the set of the vertices through which it is possible to reach selected criterion. Later, by traversing the graph in forward or backward two sets are obtained: one contains all the parts of the code that possibly can be affected by the change of the criterion and the second contains statements that could affect the criterion itself.

The dynamic slicing on the other hand inspects the unexpected result on the program. On the contrary to static variant the dynamic is able to see only the current situation and react to some specific input. Tn the case of static slicing output the quite big slice can be obtained that considers all possibilities, however, dynamic one narrows them to only one that is under interest. Also, there exists the *quasi-static slicing*. Here the static slicing use the feature of dynamic one (fixating the values of some variable) to make the "search space" smaller and lessen the requirements for computation.

One other variation of slicing is *dicing*.[24] It operates with slices, where there are two possible conditions for slicing and the two final results are then combined to obtain the information for analysis. This slicing variation was used to find the unreachable code or as a mean for located the bugs in the code.

Slicing is the technique that is opulently used in the re-engineering tools and studies nowadays. The occurrences can be find for example in [25], [26], [27].

2.2.1.2 Pattern Matching

This technique can have two slightly different directions. One direction is based on the idea, that in the code, there can be similar programming structures which are worth to explore further (for code repetition, clustering, etc.). Here some template language can be used to extract the currencies. [24] In [28] the patterns were use to find the similarities in code fragments (code-to-code matching) or also finding the abstract description (concept) in the code and creating the assignment bond between them (concept-to-code matching).

In [29] the example is shown with regular based pattern matching. They claim that available tools can miss some associations between artifacts which can lead to not precise picture on the software architecture. They propose the tool, where the user is responsible for specifying the regular expression for extracting pattern and specifies resources to analyze. This allows to create different views depending on the need.

The other approach is more developed nowadays. Here the patterns are presented in two meanings: one as the design patter – commonly defined and applied proggramming structures – and patterns – the definition (can be more abstract one) that specifies what we are looking for. Because the design patterns represent some kind of standard, it allows to understand program better. Also the obtained structure allowes to obtain the relevant knowledge of programm structure. In [30] the approaches are presented using tools which takes classes, design pattern description and possible constraints and localize the occurrences of similar structure in code.

However, the static pattern analysis can give even the false positive results. To improve it the studies were carried with usage of machine learning (e.g. [31] or [32]).

2.2.1.3 Visualization

The visualization is not exactly a technique of reverse engineering, but it is still a very important part of it. The output from the analysis do not have to be formed in understandable way. Additionally, going through many pages of text can be very tiring. For this purposes many tools support the visualization of code as whole or some parts.

The term of *software visualization* can be simply defined as "the mapping from software to graphical representations".[33] There is a broad selection on how can the information be visualized in the end, e.g. sequence graphs, data flows, diagrams, ...

Closer look to vizualization in reverse engineering is given in [34]. In the work author addresses the visualization as a mean how to create the mental picture of the software or evaluating the classes. The representation goes from more coarse-grained view just on the classes and their relationship to more fine-grained one with the inner structure of the class. In the study the polymeric graphs are used – they incorporate the metrics into visualization which can for example help to detect excessively large classes.

The other manifestation of vizualization can be the Unified Modeling Language (UML). This diagram is already incorporated in tools such as Enterprise Architect [35].

2.3 Business rules mining

The first step on the way of understanding the purpose of reverse engineering is the idea of high importance of being agile in term of changes. However, it is impossible to incorporate something new unless you know what you actually have. Nowadays a lot of companies still depend on the old software which is essential for them. On the other hand, it already hard to maintain. Moreover, the knowledge that is contained in it can already be incomplete.

As in context of re-engineering it is very important to understand the implemented business logic behind, because the stake-holders do expect the re-engineered system to fulfill the roles that it did before.

While modernizing the system it is actually very important to gasp the view on the existing rules, because in majority stakeholders expect the new system to cover them. However, as schematically depict in figure 2.2 not all of the business rules are to be incorporated into the re-engineered system. Some of them could already become obsolete during the time. On contrary the new system could have additional enhancements over the existing ones (for ex. due to the changes in legislation). Additionally, the completely new rules could be also introduced to the modernized system. However, it is important to keep in mind, that the change in business rules happen only in situation, when the business itself changes.

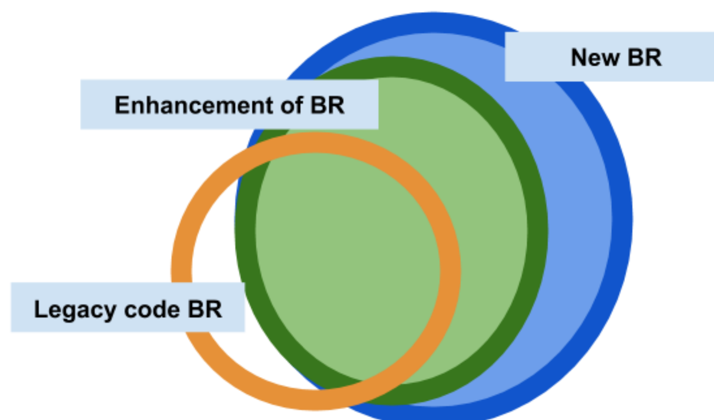

Figure 2.2: The diagram of System to be buisness rules. (inspired by [36])

2.3.1 Business Rule (BR) and business logic

As the first step it is important to define the term of *business rule*. As the matter of fact, the BR is an essential component of Information System (IS). More precisely the Information System implements the buisness process which enhalt the business rules. They later guides the flow of the processing. Consequently, the implementation of the software (the source code) stores the the business knowledge.citeBRinDB

In [37] it is stated that: "A business rule is a statement that defines or constrains some aspect of the business". It could be view as something that describes the business hierarchy or its behavior. Additionally, the rule must be atomic – there should be no possibility of braking it into smaller parts.

The greate importace is also put into its correct formulation. In the [38] the two types of BR are mentioned: top-down and bottom-up rules. In case of the first one the rule is described in more higher up abstract perspective which can be hard to understand to people who are not engaged into the domain. In the case of the second one, the provided description is more specific and understandable for other. Also in [39] the great stress is put in appropriate structure of the business rule, where the four building parts of business rule definition are: vocabulary, rules, semantic formulation and notation. In general it is composed of two parts: the condition, which has to be satisfied, and action, that is to take place on fulfilled condition.[40] In [39] it is stated, that buisness rules can be of two types which is affected weather it is connected to the structure or the behaviour:

- *structural rules* (definitions rules)– they define the structure or organization of business knowledge. Their form holds the sense of obligation or necessity. Their nature allows their automation which can be for example seen as the integrity constraints on data.

- *behavioral rules* (operative rules)– tend to describe the part of behavior. They follow the idea of obligation or prohibition. In some cases it is impossible to make each such rule automatic.

However, in [26] the categorization is different. The categories itself are divided into structural and behavioral part and they can have some way of manifestation in each of them. The four categories are:

- *term definitions* – describes the vocabulary of the business rule. Through the actual implementation it results into data tables and attributes.
- *facts* – describes the relationship between business terms.
- *constraints or structural rules* – the category provides the restriction to the behavior. The rules define the constraints that should be followed. Such rules could result into the integrity constraints in the phase of implementation.
- *derivations or behavioral rules* – describes how one knowledge can be transformed to another. Majorly, they are reflected as behavior.

This rule categorization is relatively close to the categories presented in [37]. However, if compared to the classification presented as the first, the second categories - term definitions and facts are actually thought to be not a rule but its part.

The combination of interrelated buisnes rules forms the *business logic.*[26] This logic is responsible for providing the functional requirements of the system from the perspective of user [40].

2.3.2 Business Rules Extraction (BRE)

As already mentioned, the business rules changes and this events are also reflected in the code. However, this changes are poorly documented or the documentation is even absent. Through time the software become less and less manageable. As the ruslt, programmers also tend to loose any confidence into the written documentation and due to this focus only on the implementation.[41] Such situations lead to the need to have the option to extract the business rules from the source code (software in general). The process of mining of the business rules is called Business Rules Extraction. Its purpose however does not have to be purely for documentation. The output of it can also serve as an input for modernized version of the system. Additionally, the knowledge what part of code represents some rule could be very helpful when updating the existing software.

One of the common ways how to approach the process itself depends on what is used as an input of the process. However, it is also important to be aware of the precision that the stakeholders could allow. In mostly cases they

will expect the 100% of BRE to be identified. On the other hand, to achive such a precision is a very challenging task. The posible approaches to tackle the BRE are:[36]

- *Traditional Business Analysis* – uses the communication with the domain experts and the reviewing of artifacts. The success of such an approach can be up to 70%.
- *Static Business Rule Extraction Tools* – the tolls does the source code parsing which later enables tracing the relationships of the sources and helps the code understanding. With this approach it is possible to reach 90% accuracy.
- *Dynamic Business Rule Extraction* – aims at following the execution of the rules with the "white-box" approach. This enhances the knowledge about the business rules much more, than the possible assumptions of analysts based on the communication (it is possible to the the implementation in practice).

However, it is also added, that to have the most precise view and outcome, all approaches should be used in combination. This way the static analysis and manual business analysis can lessen time requirements on dynamic analysis and, on other hand the dynamic analysis provides the precise view on the implemented flow.

The article [42] the authors tries to provide some overview on the tendencies in the field of Business Rules Extraction (from year 2013). They state, that even in reverse engineering topic already exists for a long time, the field of extraction BR from code artifacts is quite immature (the oldest study from year 1996). Additionally, it is mentioned that the process of BRE provides two representation of attained knowledge: intermediate and output. The output formed is aimed more for business people and it should be comprehensible for them. On the other hand the intermediate one could have the for of dependency graphs or table, data flows or meta-models which are used during the extraction process.

It is also valuable to know, that even though the research in this field have lasted already for two decades, there exists no automatic solution, that can achieve the perfect execution of Business Rules Extraction. In the essay [36] the authors state: "Unfortunately, there are no magic tools that will do what people want, and there never will be because software tools do not handle ambiguities very well. People are much better than software in this regard."

2.3.3 Researches and Case Studies

The tackling of the problem of Business Rules Extraction was addressed many times during a long period. The approaches were looking at some parts of

the systems, e.g. databases. Or tried to create the methodology. Also the purposes were differing. One tried to create the documentation view, the others to provide the inputs for regenerating of the software. This section is meant for presenting some of the ideas of the approaches.

2.3.3.1 Using static analysis for business logic extraction [40]

The work addresses the extraction of the business logic into the representation of BPMN (Business Process Model and Notation). The author argues in favor of static methods for reverse engineering. As such the do not require any need for input or execution of the software.

The whole process is consistent of three steps which take the source code as an input and produces the refined BPMN as the output.

1. The first step serves as the filter. The input classes are distributed into three groups (author refers to the clustering defined by Jacobson). The groups are: boundary (interfaces), control and entity. From the perspective of business logic analysis control group in of the most interest. However, including the classes from other groups is not restricted. The choice of it is left on software engineer. For this purpose author is speculating that the basic program slicing techniques can be used.

2. The second step is the the one, where actual logic extraction happens. It takes the control classes mentioned in step one (with potential added other classes) and as final step produces the BPMN representation.

   The process itself is contained from four steps. During the first step the actual parsing of the code of the selected classes takes place. Based on the parsing process the Abstract Syntax Tree (AST) is created.

   During the second step this tree is traversed to obtain the abstraction, that will contain more pure business logic. During the traversal process the more program specific parts are omitted. Such parts can be the import statements, setters and getters, variable declaration or error handling.

   The third step relies on existing technologies for UML activity generation. It will take the filtered AST representation and provide the diagram based on it. The activity diagram after will represent the input for BPMN generation which is the last fourth step.

3. The last third activity is to make the BPMN model obtained through automatic/semi-automatic way more understandable for the business users.

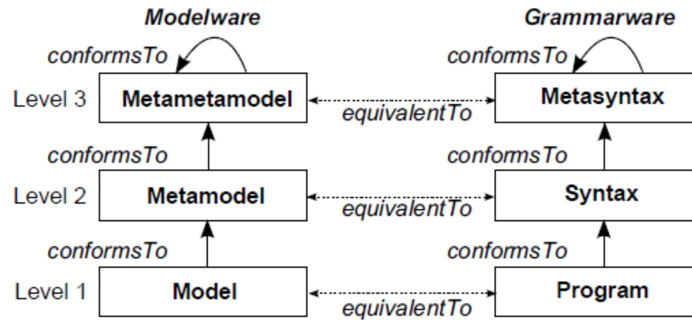

Figure 2.3: Comparsion Modelware to Grammarware (used from [26] p.44)

2.3.3.2 Extracting structural Business rules from database [39]

In this approach the authors stress the need for strict structure of the business rules. In work the use the SBVR standard. The stress is on creating declarative and non procedural BR.

As the input for further processing the tables, constraints and comments are used. At first the *business concept is derived*. In case of the tables the concept can be given by the table itself, however, some tables are just helpers (for example for decoupling m:n relationships). Also the columns can contain the concept. The foreign keys represent the verbs (inclusion, belonging). For creating fact types the constraints are used.

In the case study described in the paper it is stressed that a big advantage a good comments. They allow to deduct the relevant names for concepts and assign the appropriate verbs to facts. In case of their absence there will be a need for intervention in the automatic process to make the name selections manually. The found information was tracked in the database. Also the link between the legacy DB entity and the extracted business rule was preserved.

The result of the study was relatively successful. The automatic process provided 85% result. However, there were still issues presented in naming or false rule identification.

2.3.3.3 Model-driven buisness rules extraction [26]

The other work has a small similarity with 2.3.3.1. Author bases the idea on Model Driven Engineering (MDE). Here the model is the main focus of the process. Additionally, the similarities are provided between program grammar and model (more on Fig. 2.3). The pattern of architecture, when model conforming the meta-model, which defines the notation, and meta-model later conforming to meta-meta-model, is to some extend equivalent to program, which follow the language grammar, and the grammar that adheres to the meta-syntax (e.g. EBNF).

In general process is divided into the four bigger steps. The process in backed up with static analysis methods accompanied with some basic heuristics (the more sophisticated heuristics could be added later for specific cases). It is claimed, that the behavioral part is to be found in the statements from execution path for specific business variable. The structural parts are, on other hand, in the declarative constraints in database or in the conditions for triggers. The four described steps are:

1. *Model discovery* – this steps allows to move the heterogeneous program representation uniform model representation. Also it is possible to have more view that later can be combine into one. The model representation is to be one-to-one mapped to the program structure (one of such model is AST).

2. *Business Term Identification* – in this step the identification of business terms. The terms could be found in different places. One are to be found as variables in behavioral part or as tables, columns and view in structural part. In case of behavioral part the heuristics are implemented such as selecting the variables with mathematical operations, conditional statements or input and output statements. The structural part is claimed to be in majority corresponding one-to-one. After the extraction process the discovered terms are still traceable back to their code representation. This is the result of the idea of MDE.

3. *Business Rule Identification* – after attaining the business term the next one is identification of the rules itself. Here it is again different for behavioral and structural rules. In case of structural rules the database constraints are analyzed. For behavioral part the control flow and data flow analysis is applied, additionally, the slicing is executed.

   The control flow analysis allows (using static analysis) to create the picture on possible execution paths and also finding the unreachable parts. The output after traversal is the input tree with respective annotations to show the flow.

   The data flow analysis take the model from the previous. Here, the context of the variable takes place. The paths are traversed in attempt to connect the statements that affect the same variable. The statements are then to remember their predecessor or successor to keep the trace.

   As the last step slicing methods are applied. It allows to obtain the parts of the code that are relevant to specific variable. The output then contains the business rules and context.

4. *Business Rule Representation* – the final step is meant for representing the finding from previous analysis. It is consists of two operations:

Vocabulary Extraction and *Visualization*. Also, at the end of it the traceability information is included to the output to preserve the connection to its source representation.

The Vocabulary extraction step is meant to provide the verbal representation of program constructs found by analysis. The default generated output for behavioral rules is a tuple with the name of the variable/function and its definition. In cases of insufficient information the human intervention can be needed to provide the right naming. In case of structural rules the database schema is considered. Out of it the business terms and their relationships are extracted. The output is then presented in a form of UML defined conceptual model.

The next operation is Visualization. The purpose of it can be in clarifying the view on the relationship of the obtained rules. Also the textual representation can allow to focus just on one specific rule.

2.3.3.4 Business rules extraction for purpose of application modernization [36]

Compared to previously provided studies, the company claims that to obtain the best output from BRE the combination of static and dynamic analysis is needed. What's more, the human input is also expected. Even though the costs for applying the static and dynamic analysis can be quite high, it is told to be compensated by lesser need for testing or the repair.

The divide their process in three stages:

- Stage 1 – encompasses the business analysis via interviews and review of artifacts. This stage can get up to 70% of needed information. Although, tho some extend business analysis are able to collect significant amount of information, their knowledge is affected by the people they interview and documentation resources provided. As the result, some information maybe missing or is ambiguously presented. Due to this, it is worth being able to correctly specify the point, where there is no new information to obtain the the process should move to stage 2.

- Stage 2 – static business rule analysis. Through applying it after the first stage it is possible to reach more precise output. It uses the BRE tools which are expected to provide the programmers' business rules (the one that reflect the implementation). One can provide fully automatic output which is as authors state a good reference, but not a good business rules description. The other can provide the semi-automatic approach with dependency on user queering. The quality of queering sadly can be different and solution itself is not robust. The main factor can be the business analyst and his experience. In the end the output is taken and

translated into the business analyst business rules and business requirements.

- Stage 3 – dynamic business rule analysis. This stage is seen to be the critical in obtaining the remaining precision, however, the success of this stage is conditioned by the correctness of its execution. The tools that acts like debuggers are expected, so that it is possible to follow the execution along. This fact is especially pointed out, because the "white-box" manner of it provides more advantages for creating tests and checking the correctness of the results until now (there is no pure dependency on the requirements).

  The stage starts with preparing the test-cases which are later to be used for dynamic analysis. They are done keeping the previous steps' results in mind. It also allows to find possible errors in defined rules. While execution of the tests the information about code coverage is collected. It allows to detect possible missing rules.

2.3.3.5 Reflections on Approaches

The first three provided studies on business rules extraction are more from the academic area. Here it is visible, that the tendencies for applying the static analysis take place. The arguments against usage of dynamic one are usually due to the need of software execution or dependency on the inputs. The aim was at mainly automatic process with possibilities of human intervention. Even though the results after application of the studies (2.3.3.2 and 2.3.3.3) were considerably good, there still parts that were not successful left.

On other hand, the last essay describe the view from practice, where the it is also important to achieve the maximum precision. They combine more means for analysis with significant user input. Here, the dynamic analysis seen as the mean how to achieve the best precision. Also the feature of it following only one execution path depending on input is seen as benefit. Here the usage of each approach is driven by its payoff.

On other hand, the academic studies actually provides the refinement of one of the stages of the process 2.3.3.4. They provide the valuable improvements, that later can aid to the BRE in general. Even though there is some striving into automation of the process, the user input still remains needed if the higher precision is to take place.

2.4 Reverse engineering solutions and tools

The previous sections served as overview of existing principles and studies in the area of reverse engineering and re-engineering. But to execute reverse engineering process more efficiently and faster, the supporting tools are in general needed. On the market some spectrum of full solutions is available.

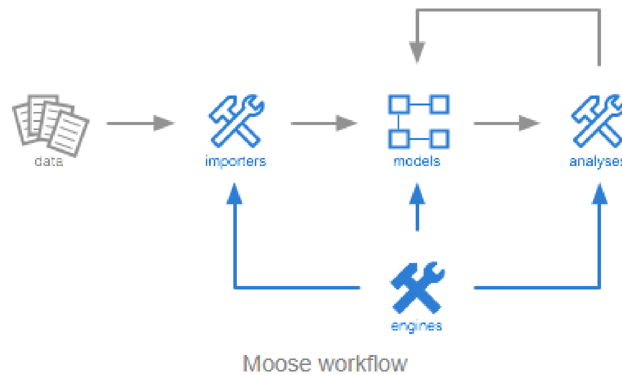

Figure 2.4: The general representation of the Moose Workflow (used from [16])

These projects in majority provide the tools (automatic or semi-automatic) that serves the purpose of code understanding or even applying Business Rules Extraction. Moreover, some of the projects provide also the tools for software modernization and re-engineering.

On the other hand, there are also the smaller projects/tools, that maybe do not provide the vast possibility of full stack toolkit, but they still cover the part of providing the insight into the system.

This section is meant to provide in generally brief description of some of the available possibilities as for the area of commercial full-stack toolkit or projects, and also smaller tools.

2.4.1 Moose [16]

The Moose is platform with the idea of providing the help in data analyzing in mind. The users of this platform can be divided primary in tree categories: the researchers in area of software analysis and reverse engineering, the architects and engineers that aims at understanding their code or the tool builders.

The tools are written in Smalltalk dialect called Pharo. The syntax of it is relatively simple (the postcard syntax), however, it is also powerful. Moose helps to analyze the data, however, the process of analysis is upto user to program. The platform is not meant for simple clicking.

The general idea how the Moose platform works is represented by 2.4. The initial input to the tools (platform in general) are data. It can be different types of objects, properties and their relationship. This all can be found in one project, where there are source codes to represent entities and relationships and configuration files to provide additional information. The data itself are loaded to system with help of the *importers*. These importer can be of two

kinds: internal one (e.g. for Smalltalk code, XML, JSON, MSE) or external which can be defined for other non available technologies.

After data are imported they get their model representation. From this point on the all the operations adhere to iterative nature of the Moose logic: the models come to the operation as an input and this operation provides the model as an output. This also allows chaining the operations to obtain the result. The other idea behind is that each of this operation is customizeable and is to be tailored however is needed. For this purpose the dedicated engines are provided to create new importer, models or analyses' processes.

Of course Moose contain the basic options for analysis. It is possible to obtain some metric of the data, query the model or even visualize it. Moreover, combination of what available options can provide the measures how to tackle more complex solutions.

As mentioned previously Moose provide the specific engines which can be used to create customized tools:

- *Petite Parser* – this engine helps with the first stage of Moose workflow – importing of the data. It allows to create a custom parser that will help to parse needed grammar. Except the basic ability to define the abstract description of the structure to parse, the engine provide also an option for debugging. The browser allows to try the defined grammar correctness by providing the textual input. Additionally, it is also possible to visualize the created grammar through graph.

- *Fame* – this is one of the two engines, that provides the meta-modeling infrastructure for Moose. Also one of the features it provides is the compact serialization format (MSE) which can be used for not only importing the data structures into Moose, but also their exporting (for example after enriching it by analysis execution).

  The Fame is used for majority of meta-models in Moose. One of the most vivid from them is *FAMIX*. This models enables the technology independent model representation of the code. In contains the elements that are able to model things like attributes, methods, accesses. The models provides the meta-description for the Fame engine which allows to separate its actual implementation from the abstraction. The FAMIX is also providing the API (Application programming interface) to execute queries upon the model.

- *Roassal and Glamour* – Roassal represents the engine that is used for purpose of the visualization. Even though it uses the fine grained objects to describe the view, it also provides the built in builders, that allows to create more generic types of graphs faster. The obtained visualization has also benefit of being interactive, so it is possible to inspect each its elements separately. Glamour, on the other hand, provides the

possibility of creating the interface, that user can use for further model browsing.

All the described engines could be seen combined in *Finder*. It is an interface which allows to browse the model structure alliteratively. Provides the view depending on the model (if it is the structure of more entities or just a leaf node containing the entity). It is also possible to use queering on the obtained structures (FAMIX API).

2.4.1.1 FAMIX and MSE

As was mentioned previously, FAMIX model is based on Fame meta-model engine.The idea behind its introduction is relatively simple. In reverse engeneering exists a lot of tools that process the information (sourcecode) and it would be more combortable to have one platform independat representation of the system view (model).[43]

The MSE format allows to describe any generic data without the dependence on any meta-model. It provides relatively compact, simple and extensible format. It is less verbose then XML and take use of parenthesis, which allows easily describe inclusive structures. The main purpose of it is having the good interchangeable format between the systems.

The FAMIX "is a family of meta-models for representing models related to various facetsof software systems".[43] Even though the core elements can cover some spectrum of technologies (see hierarchy model at A.2), it may be needed to keep in mind that there can be a need for its extension later for another, not supported, languages.

The whole structure is quite complicated, however, from practical point of view only some basic elements are needed to start the process of analysis (simplified visualization in A.1):

- Namespace – it represents the wrapper for all variables names. From the point of technologies like Java it can be confusing at first, because this element actually stands for what is know as package

- Package – the package has more characteristic of what package is in SmallTalk

- Class, Method, Attribute – the elements have the same meaning as the programming abstractions

These elements are related to each other. This relationships can be described by four types:

- Inheritance – this relationship denotes what is known as inheritance in object-oriented languages (relation between subclass and superclass).

- Access – this relation has in general two participants: the method that is accessing and the attribute that is being accessed (behavioral-structural entity relation).

- Invocation – describes the connection between source and possible behavioral entities (Methods) that are invoked by it (participating entities are both behavioral – methods or functions). The plurality of methods is due to the feature of dynamic languages, when it is not possible to easily determine the invoked method by name, because the type of instance of an object could not be determined by static analysis.

- Reference – represents the method-to-type relationship. The *type* construct, however, is not very typical in all technologies. The purpose of it is to allow having the "multiple inheritance" in purely object oriented wolds.

2.4.2 BlueAge [44]

On contrary to Moose, that provided platform to enhance, BlueAge represents the full commercial solution. The provide the tools to help analyzing the code or their services to modernize the applications (re-engineer them). The technology that are used comply with the idea of Model Driven Architecture. In initial steps the source is translated to the model which is later operated with. The primary concern for the company are legacy COBOL programs. However, it also supports analysis of different technologies (e.g. JEE, PL/SQL).

The initial phase of reverse engineering produces the Blue Age model and parametric mock-ups. Later, for modernization the manual mapping is dome between obtain model and the model for forward engineering. The analysis is available in sense of obtaining the data flows, getting the predefined patters or trans-modeling.

2.4.3 MoDisco [17], [45]

The another open-source solution is MoDisco. It is in general Model Driven Reverse Engineering (MDRE) Framework. The goal of it is to provide a means for development of model-driven/model-based solution for specific usage in reverse engineering process. It supports two basic stages defined by MDRE standard: Model discovery and Model Understanding.

The MoDisco is itself a set of Eclipse features and related plug-ins available to obtain into the Eclipse tool. The plug-ins are parts of generic components that create the MoDisco's infrastructure layer. The example of such components are: Discover Manager, Workflow, Model Browser or Querying. Each of this components remains not dependent on artifacts under the interest or the reverse engineering purpose. Additionally, the mechanism for dynamic

met-model extension is provided. This allows to extend the model with additional information at a run-time. Such extensions are called *facets*. Last but not least, MoDisco provides the means how to presents the metrics result and visualize them. On technological layer MoDisco provides a set of useful deployable components. Natively, MoDisco provides the support for technologies like JEE with JSP and XML.

The main idea which is supported by this framework is very similar to what Moose has to offer. It bases the assumption that the analysis process is to be consisted of a lot of transformation steps, that will lead to result. Each step of this workflow receives the model as an input and provides its modified version to the output. The layer that is in the end responsible for orchestration of the workflow and application of components is Use Case layer.

MoDisco is, however, fully extensible and it is possible to apply changes to each of its layers. Although, creating the modifications on the Use Case layer can be easier than, for example, adding the support for another legacy technology on the Technologies layer.

2.4.4 Overview of other projects

The previous etries were looking more specifically of separate solution, however, there are more to be found. One such solution is from DeepAlgo [46]. Their target is understanding of code by extraction the information from the algorithms and visualizing it.

The SoftwareMining[47] company provides the services for modernizing the COBOL applications. They apply the artificial intelligence algorithms to obtain the better heuristics for pattern matching in legacy systems' code.

The other following tool set is from company Imagix [48]. It provides the possibility of reverse engineering of the code written in C, C++ or Java. The main purpose of the tool is to provide the aid to developers and help the understand the code better (their own code or obtained from the third party). The tool enables the code high-level abstraction with the option of analyzing data flows. Additionally, there is possibility to check metrics of the code to determine the fitness of it. Last but not least, the tool gives the chance to generate documentation based on the code information.

On contrary to the comercial, it is also possible to find the opensource projects centring on BRE[49]. The project supports only rule extraction from Java code and uses MoDisco plugin for source code abstraction. The output is very schematic and is primary textual.

2.4.5 Reflections

The provided overview showed some of the examples of existing technologies for reverse engineering. Majorly the technologies are commercial, however, in

these cases they usually also offer the services for modernization of the legacy code-base.

The presented open-source solution have a common idea in mind. In general they represent the framework/platform, that provides some basic capabilities that can be used for further development and application to artifacts' analysis. Both of them work with model representation of the code, that abstract from underlying technology, and counts with iterative querying/transformation approach.

For further practical experiments the Moose platform was selected. This selection was partly base on the fact, that the Pharo technology is easier to gasp and provides a abundant output.

Part II

Application of research

Chapter 3

Introduction of sample projects

Due to nonavailability of the real legacy systems, the four small projects were taken form online sources or provided by other students. The initial idea is to try to see the possibility of getting the results on something small and traceable. This to some extent allows variability of input but also the possibility of manual reevaluation.

All code-bases tries to reflect the informational system. This step is made because the possibilities of the possibilities of the Prime Radiant and existing application of NS theory.

As the legacy technology Java language was selected. The main reason for such choice is that the technology that NS expanders support is also Java. This also allows easier transition between thinking than trying to re-engineer procedural or functional language.

In the following section the brief introduction will be provided for each project. The introduction will include primary the source code structure description. The primary accent will be put on data structures – possible data elements.

3.1 Project 1: Student Information System 1 [1]

The first project for the exploration is representing a very simple student information system (UML available B.1). It uses the Swing framework for providing the graphical interface for user with simple operations of creating, updating, deleting and searching for specific student by id.

The data are persisted into the relational database. All information is saved only to one table called *Student*. The table has columns: ID, First (first name), MI (middle name), Last (last name), Age, Gender, Status, City, Diploma, Form137, TOR, Birth. The last four items represent the requirements that can either be labeled as “checked” or “null”. Except for persistence operation (CRUD) the system does not provide any additional expressions of the behavior.

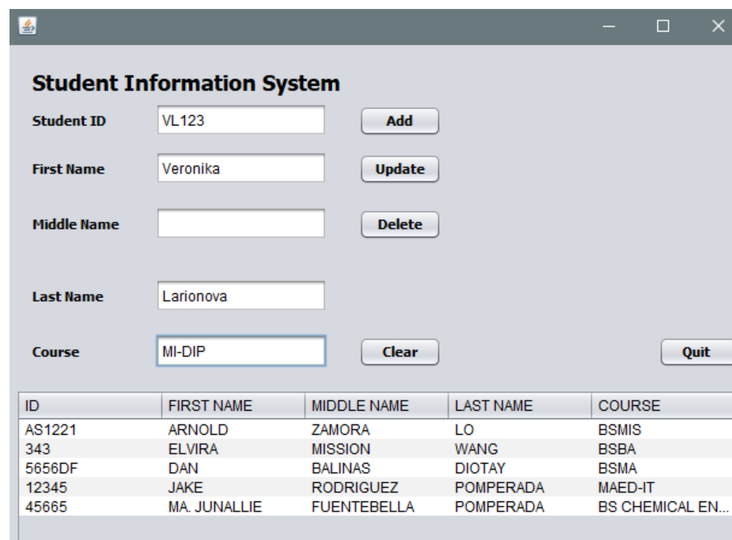

Figure 3.1: Student Information System 2 – Interface

Except the main class, the project also includes so called *Anonymous classes*. This classes responsible for handling the events. The example of such class is for example:

```
CBDiploma.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent event) {
    JCheckBox CBDiploma = (JCheckBox)event.getSource();
      if (CBDiploma.isSelected()) {
         txtRDiplomaHolder.setText("check");
      } else {
        txtRDiplomaHolder.setText("");

      }
 }
});
```

3.2 Project 2: Student Informational System 2

The second project also represents the student informational system. Similarly it is also mainly written in one class, that fully takes the responsibility for everything happening. The project has the graphical interface specified in separate file. The saved data are shown in table(see 3.1. The table is populated from the text file, where each row represents one record. The record contains the information about the student and it connection to some course.

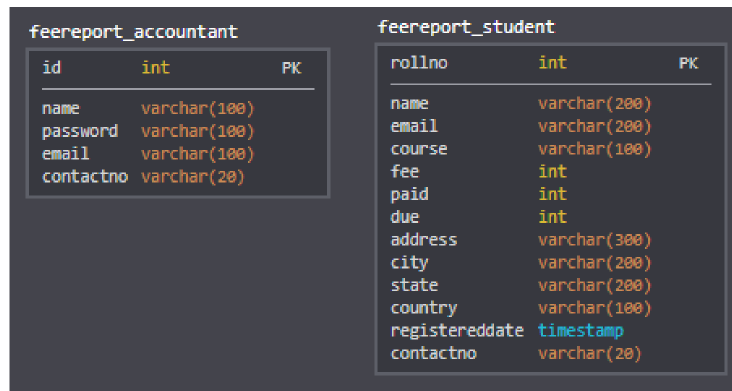

Figure 3.2: The database schema for Student management system.

Persisted information is: ID, FIRST NAME, MIDDLE NAME, LAST NAME, COURSE. Also similar to the first case this project also contains anonymous classes, which are responsible for handling the events. Except handling the new insertions, updates and deletions the system does not perform any other business valuable activity.

3.3 Project 3: Student Management System [2]

The third projects describes the student management system. The purpose of the system is to be a fee report software. For this system there are two roles specified:

- Admin – should be able to add/view/delete accountant
- Accountant – should be able to add/view/edit/delete student check due fee and logout

From requirements follows that, there are two entities to be saved. One entity is accountant and the other is a student, that has to pay a fee. The schema 3.2 shows the both tables and it columns. In case of an accountant the attributes like name, password, email and contact number are persisted. The student record on the other hand contains the information that is used for describing the student: name, email, address, city, country, contact number, and the fee that is assigned to the student: fee,paid,due,course,registered date.

Form the generated UML diagram B.3(the diagram is missing the relations between classes) it is visible, that the classes can be divided base on the purpose (or the entity they are responsible for). The grouping is marked by dark rectangles. The classes Student and Accountant are purely data holders with only accessors available. They are later used by the classed which are responsible for graphical interpretation and database manipulation.

3.4 Project 4: Ticket selling system

The last project represents the ticket selling system. This sour-code was provided by Jakub Jirsa, who created it as the prototype for his bachelor degree [50]. The application was meant to be as desktop with connection to the shared database. The project was created to provide the better solution for the city Volyň.

The code is primary structured to the packages that are describe the responsibility of the classes that are contained in it.

- ZakladniCast (Core Part) – the package contains the classes that are responsible for database connection, base GUI creation and the User class.
- PomocneTridy (Helper Classes) – here are the classes for helping with every database manipulation, class for establishing and managing FTP communication and, finally, the class that is responsible for common data validation.
- Clanek (Article) – the package groups the classes that are responsible for handling the manipulation with the articles in the system. There should be two operation that is possible to do with articles: create and edit them.
- Prodej (Selling) – the classes contained in this package are responsible for correct selling process. It is stressed out, that the situation, when the already sold or reserved ticket is sold to someone again, should not happen.
- Rezervace (Reservation) – reservation are meant in context of the tickets. The package has similar properties as the event package, where there are classes responsible for the creation of the GUI and user interaction. Additionally to it there is the main class that is responsible for handling the reservation data.
- Akce (Event) – the event package is meant to contain the classes that are responsible for handling the initializing, creation, update and deletion of some cultural event, that is to happen in the city. The two contained classes are primary responsible for drawing the content of the form and all needed graphical elements. The main Akce (Event) class is responsible, on the other hand, for the data.

The schema 3.3 represents the Database tables for the project. The model contains already mentioned entities such as Event, Article, Selling, Reservation. The Employee table reflects the User class for determining the access rights to the system. However, there is also the tables for Ticket, Entrance

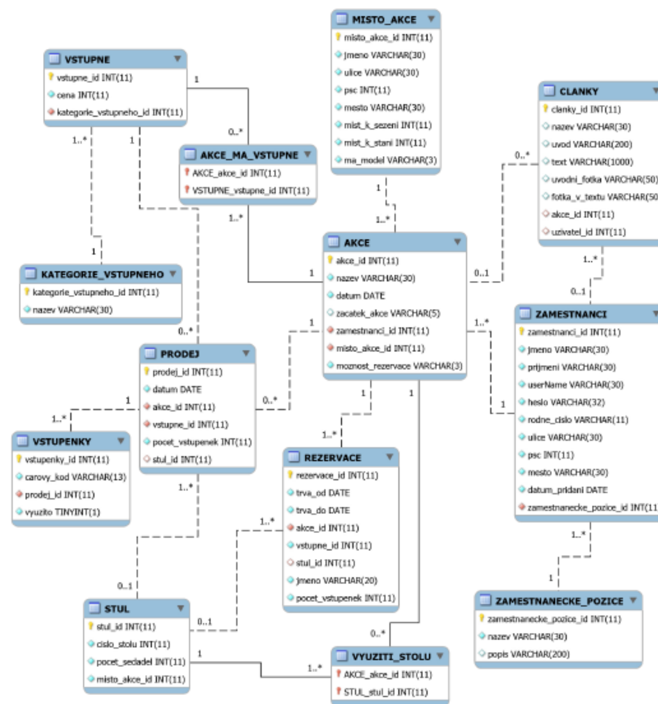

Figure 3.3: Database model for the Ticket system (used from [50])

Fee or the Counter, which comes logically from the nature of the application, although they do not have their dedicated classes.

3.5 Evaluation of the projects

As mentioned before, there were no legacy projects available. Due to this reason the smaller projects were selected. Additionally, their size allows to make the manual exploration and understand code better. Moreover, the selection of the code-base was restricted by the requirement narrowing the system types only to information systems.

The first two project represent a very simple, one class projects (if the anonymous classes are omitted). On one hand, having only one major class seems to be easier to gasp. However, it can be harder to obtain the data elements from it. Even though they may seem to similar, the project number 2 provides the possibility of exploring the usage of different persistence medium.

The third project is already more sophisticated. It contains the data classes, which reflect the almost one-to-one mapping with database tables, and the classes responsible for graphics and logic. The process also has more “business logic” inside. The first two projects had only the CRUD require-

ments, however, here there had to be implemented some validation. And access logic.

The last projects provides more real life version of legacy system. The structure of the classes is vaster and also the project contains test classes. Additionally the business rules in the code-base show more complexity and have real-life feeling.

Selected projects, however, are not using more sophisticated implementations in enterprise systems. They are purely desktop with no usage of web technologies. Also the applications have more academic-practice shape. The selected approach in case of implementation is usually not neat. The databases' schema are also lacking in correctness. However, this can also help to simulate different programming styles and thinking.

Chapter **4**

Experimental phase

As was stated at the end of the tools review. For the purpose of experiments the Moose platform was selected. In majority, for the experiments the provided query API will be used to traverse the relationships of the elements.

To obtain the traversable FAMIX model the jdt2famix [51] application was used. It takes the source code as an input, processes it and provides the serialized model in MSE format. This MSE format contains the information about the classes, interfaces, methods and attributes. The MSE format then can be loaded into Moose Panel. The panel (Finder) has some basic queries already implemented, so it is possible to to the traversal by inspecting the elements through clicking.

The primary aim of the experimental phase is finding the correct candidates for the data elements in the code. The features that the steps will be looking for will be:

- name of the data entity
- the attributes of the data entity
- the relationship between possible candidates

In previous section the data elements were located based on manual inspection of the code. This section will at first try to find the path to the candidate separately for each project. As the next step, the attempt will be made to try to generalized the ideas from findings.

The section will try to aim at answering the questions, where it is possible to automatized that inputs from data elements. If yes, how high is the failure rate compared to success. In the end of the chapter the conclusions will be made based on the observations.

4.1 Exploring the projects

This section is meant for describing the steps made in each projects to locate the data elements features. The steps will be attempted in two ways: firstly, through the moose panel by clicking through and queering in finder. secondly, by reproducing the process in Moose playground.

Additional comment could be made on the fact, that the Moose differs between the term of "model" classes and all classes. The model classes represents some subset of all classes. In fact, the main difference between them is that all classes includes even so called "*stub*" classes which are the model reflection of the class without implementation in the project. Example of such class can be the elements from the Swing library: JButton, JLabel, etc..

4.1.1 Student Information System 1

The system has only one class representing the whole system. This system representation contains the only one specified data element stored in database table. The element consists of all inputs provided through the graphical interface. These inputs are translated to the values of the columns.

From the previous evaluation there are two major indicators to follow for finding the desired element: the dependence of it on database and on the graphical elements.

4.1.1.1 Exploration through Moose Panel and Playground

For the panel go-through the following tactic was designed: because the aim is to obtain the names of the attributes and the name of the element, the idea of using the components of the GUI was used first. The labels in this case should include more correct naming compared to variables or even column names.

1. Obtaining all the attributes of the model.

2. Selecting the attributes which are of the type JLabel and getting their names. – With this step the possible names of the attributes are found.

3. Select the package for database access (sql). This decision tries to see if the results obtained through finding the labels will be the same, or they will only have common intersection.

4. Obtain all model incoming dependencies and find accessed attributes for each of them. After select the ones with JLabel type. – At first all elements that depend on the package are found (they are using the database connection. After it of each element the appropriate attributes are found and the JLabel ones are selected.

5. Make intersection between results from step 2 and 4.

After applying all the steps described the obtained intersection contained the same results and the intermediate values in step 2. The obtained labels matched the expectation majorly. Although, the additional label for background appeared. The other issue was that the separate column names for requirements were not found. Instead of it the label "Requirements" was found. The name for entity could not be found directly, because the table name occurs only once in code like the string constant passed to the function. FAMIX model, however, does not evaluate it as a part of model node, so this information has to be filtered out manually. Because this application contains only one entity, the relations were not in scope of investigation.

4.1.2 Student Information System 2

In case of the second project the situation is very similar to preceding example. Again, the sought data element is only one and its attributes are scattered across the system. The only difference is the persistent medium is now a file. The steps previously defined will be applied again with only one modification. The library or wring to file will be used in step 3.

The execution of same steps, however, lead to unsuccessful results. The first output provided the names of the label which had a very generic names, e.g jLable1. The results from the 4. step utterly differed compared to the first one. No label attribute was available here anymore. However, through iterative queering the right function for saving the information to file was found, obtaining the column names is difficult. The names could be deducted from the attributes of type String or JTextField.

In this case the steps that goes from the direction of persistence of the data should also contain the manual intervention to check the source-code for string constants with possible name candidates. The name for the entity is again missing. It can be to some extend deducted from the only class of the project or obtained manually from the constant string with the name of the file with records.

4.1.3 Student Management System

The student management system contains two data entities in database, however, there is also a possibility seeing the role of admin as other entity. The code itself is more structured and divided to several classes. On contrary to the previous project, for this project the steps will start from looking at the links leading from database.

In this case the first steps were following: the correct package was selected. For this package all model classes were found. However, the result contained two classes that were actually executing the database operations and validations. So the second querying for all outgoing dependencies of this classes was

executed. Only then the classes for Student and Accountant were found, but at the same time they were mixed with the database helper classes.

After obtaining the desired classes it is not that hard to obtain its all attributes. Later the relation between the classes can be queried through the FAMIX query language. The names of the entities are obtained from the class names.

4.1.4 Ticket System

The ticket system provides more sophisticated database structure. Because the example is more closer to the management system case, similar steps were also applied here. The exploration started at the jdbc package and continued further. However, with each iteration more of additional surplus classes appeared. In the end with this method it was not possible to reach sensible results.

The other option contained the idea of starting from the list of classes and reject them by specific criteria, like for example filtering out the test case classes, anonymous classes. Regrettably, even after filtering the other excessive classes remains. Also, because the project also had the code-base from library for accessing the database, this code was also mixed as not a stub.

4.2 Reflections

Due to the fact, that through through applying the logic to each project it always resulted with generally opposite outcomes, the generalization step was skipped. However, one attempt for general algorithm was made. It was based on the idea, that all important data should be persisted, so it traced the dependencies on the packages which enables the persistence. But because there are the case like project number 3 and 4, where the data class is not directly dependent on package under the observation, the need for transitive definition of dependency arouse. If we have already some classes and they are not our data entities, then the data entity will be used by one of the classes (outgoing dependency). The algorithm then could be as follows:

1. get all incoming dependencies, that are not a stub, for package the is observed

2. while list of candidate is still updating, get all outgoing dependencies for each candidate as a set

This solution, however, led to a big amount of candidates, where majority of them ware not connected to data elements. To get more precise results in these cases, the guided iteration with options of selection the better candidate would produce more accurate results.

As the result of conducted experiments two possible directions for proceeding further were formulated:

- The first branch came from the fact, that maybe by trying to improve the heuristics in traversal of the elements, the possible candidate could be found to some extend. However, the general algorithm would lead to big failures in locating of the entities. This is caused by usage of different programming styles and the variable architecture of the program.
- The second branch lead to the idea, that all iterative steps are to be done manually, because the person is able to guess the direction of exploration. This can also narrow the final outputs and lead to correct solutions.

Although, if compared to business logic extraction process presented in 2, the extraction process seemed to be more keen for automatic approach. The possible reasons of unsuccess in obtained results, in this case, can be due to the "granularity" of FAMIX model. FAMIX model works on high level of abstraction and does not consider the behavioral part of the system. On the other hand, in the BRE in 2.3.3.3 the accent was put on the fact, that the model should reflect the code, which, for example, includes having the branching statement in it. This information is later used to determine the variables and behavior.

Chapter 5

Further directions and proposals

The experiments tried to see, whether it is possible to achieve the automation, or at least semi-automation, of the extraction of inputs for Normalized software systems. However, for this case it was shown, that on seaming small projects it was not possible to find some general step sequence that could lead to successful results. The selected tool allowed to view the code and navigate through it, but for whole understanding of the traversal it was needed to also refer to implementation itself. For reaching the desired information it was necessary to constantly make queries and checking the source statements for possible name string containment.

However, the reverse engineering can be still useful in the phase of analysis of the legacy system for expanders. As written in [36], the static analysis of legacy software can reveal the business rules which the interview just could not find. There can be more reasons for it happening, but the simplest one is that people just do not remember. On other hand, trying to apply just "black-box" dynamic analysis does not have to lead to desired information, because just the can always be on case of the input which did not happened. In this case it is worth to rethink the usage of code analyzing tools.

Currently, Moose is relatively good for visualizing the rough structure of the program. However, more desirable features of the tool will be closer to what MoDisco tool can offer. It provides more fine-grained view on source-code structure (enabling to analyze control flow or data flow). This allows to react to branching conditions or mathematical operators to deduct the behavior of the variables in system.

However, using Moose can still have an advantage. The modeler, which supports the modeling process of Normalized Software, uses Pharo as development platform. This can allow to transfer the information, obtained through querying which is executed on the code, to the modeler for visual assessment and possible correction. In this case, there will be a need for improvements of the model that is obtained after parsing of the code. Also additional helper tools could be provided to lessen the need of writing the queries as a program

for analysts.

On the other hand, the already existing FAMIX model provides the API for querying the relation of the elements. If the correct views are created, this information can also help for reverse engineering of legacy systems. For example, with the already obtained knowledge of domain or having some knowledgeable expert, it is possible to gain a better picture of the relationship like this: method A from class A calls/invokes the method B of Class B. With a bit of knowing this information can make more sense and it is possible to deduct more significant outputs.

In conclusion, the summary of the propositions could be stated in points following:

- see the use of reverse engineering not for automatic extraction of inputs, but as some helpers that can accompany the process of business analysis in context of NSS.
- either try to explore the new tools for reverse engineering or refine the Moose to enable querying not only the general structure, but also more fine-grained model which will contain the information concerning the inner behavior of the methods.
- last but not least, considering the usage of moose visualization as guiding the analysis. The set of specific view can be predefined. In combination with already existing expertise this will enable the revalidation of already obtained information.

Conclusion

In conclusion, the thesis provided two sections. During the first part the introduction to the concept of Normalized Systems theory was conducted. This was accompanied by the overview on the concept of reverse engineering and its application in case studies. The most prominent area, the reverse engineering is used for, is represented by re-engineering and software modernization. As a subset of this process, reverse engineering can be used to gain the information about the business rules, which are hiding in legacy artifacts. This knowledge is then used in forward phase to obtain the re-engineered software.

Thesis took the manner of initial exploration to see, if the reversed engineering can be used for the means of making the process of analysis of legacy systems more efficient in context of Normalized System expanders. The initial idea was too see, whether it can be automatic to some extent (this assumption came from the available toolkits that promise fully automatic tools for rules extraction).

The experimentation started with just one goal in mind – extracting the simple information: name, attributes, relations, for data element template. Few small projects in Java were selected to simulate the legacy code-base. For code analysis the Moose tool was selected based on previous evaluation of possibilities.

For each project the querying process was executed to see, if by applying some steps, it would be possible to reach the desired information. However, even for the projects, that were seemingly similar, it was impossible to get the desired output by applying same steps. The difference in implementation an code structure could not be tackled by the means of Moose querying.

However, based on previous research in chapter 2 it is speculated, that if the model for traversing can mimic the behavior implemented in methods, it will be possible to find more clues for attaining the desired information and identifying the data elements.

Additionally, few other advices for further exploration and implementation were formulated. However, they do not stress the need o automation of the

process. On the contrary, the process of analysis should be done in combination with classic business analysis. This will help to gain useful output and possible validation criteria.

Bibliography

[1] Bermoy, L. Studen Information System. 2018. Available from: https://www.sourcecodester.com/java/8805/student-information-system-java.html

[2] Point, J. Fee Report — Studen Management System. 2018. Available from: https://www.javatpoint.com/fee-report-project-in-java

[3] Technopedia. Legacy System. 2018. Available from: https://www.techopedia.com/definition/635/legacy-system

[4] Oorts, G.; Huysmans, P.; et al. Building Evolvable Software Using Normalized Systems Theory: A Case Study. In *2014 47th Hawaii International Conference on System Sciences*, Jan 2014, ISSN 1530-1605, pp. 4760–4769, doi:10.1109/HICSS.2014.585.

[5] aclav, R. V. Five recommendations for software evolvability. *Journal of Software: Evolution and Process*, volume 0, no. 0: p. e1949, doi:10.1002/smr.1949, e1949 JSME-18-0004, https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1949. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1949

[6] Mannaert, H.; Verelst, J.; et al. *Normalized systems theory : from foundations for evolvable software toward a general theory for evolvable design*. Koppa, 2016, ISBN 978-90-77160-09-1, 507 p. pp. Available from: http://hdl.handle.net/10067/1367590151162165141

[7] Coelho, R. M. F. Normalized Systems: An Assessment of Evolvability Based on Metrics. 2016. Available from: https://fenix.tecnico.ulisboa.pt/downloadFile/844820067125175/MEIC-73265-Ricardo-Coelho-Thesis.pdf

[8] Oorts, G.; Ahmadpour, K.; et al. Easily evolving software using normalized system theory-a case study. *Proceedings of ICSEA*, 2014: pp. 322–327.

[9] Chikofsky, E. J.; Cross, J. H. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, volume 7, no. 1, Jan 1990: pp. 13–17, ISSN 0740-7459, doi:10.1109/52.43044.

[10] Morgado, I. C.; Paiva, A. C.; et al. Dynamic reverse engineering of graphical user interfaces. *International Journal On Advances in So ware*, 2012.

[11] Harsu, M. *Re-engineering Legacy Software through Language Conversion*. Dissertation thesis, University of Tampere, 2000.

[12] Nguyen, P. V. The Study and Approach of Software Re-Engineering. *arXiv preprint arXiv:1112.4016*, 2011.

[13] CanforaHarman, G.; Penta, M. D. New Frontiers of Reverse Engineering. In *Future of Software Engineering, 2007. FOSE '07*, May 2007, pp. 326–341, doi:10.1109/FOSE.2007.15.

[14] Dit, B.; Revelle, M.; et al. Feature Location in Source Code: A Taxonomy and Survey. In *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

[15] Lau, K.-K.; Arshad, R. *A Concise Classification of Reverse Engineering Approaches for Software Product Lines*. 4 2016.

[16] The Moose Book. http://themoosebook.org/book/index.html, accessed: 27.02.2018.

[17] Eclipse Foundation, I. Eclipse MoDisco — The Eclipse Foundation. https://www.eclipse.org/MoDisco/, accessed: 21.02.2018.

[18] Codiscent - Generative software engineering. http://codiscent.com, accessed: 10.03.2018.

[19] Hainaut, J.-L. Database Reverse Engineering: Models, Techniques, and Strategies. In *ER*, 1991, pp. 729–741.

[20] Weiser, M. Program Slicing. *IEEE Transactions on Software Engineering*, volume SE-10, no. 4, July 1984: pp. 352–357, ISSN 0098-5589, doi:10.1109/TSE.1984.5010248.

[21] Postema, M.; Schmith, H. W. Reverse Engineering and Abstraction of Legacy Systems. volume 22, no. 33, 1998.

[22] Beck, J.; Eichmann, D. Program and interface slicing for reverse engineering. In *Proceedings of 1993 15th International Conference on Software Engineering*, May 1993, ISSN 0270-5257, pp. 509–518, doi:10.1109/ICSE.1993.346015.

[23] Sasirekha, N.; Robert, A. E.; et al. Program slicing techniques and its applications. *arXiv preprint arXiv:1108.1352*, 2011.

[24] Crispin, K. *Software Maintenance and the 3 R's: Reverse Engineering, Reengineering and Reuse*. 1994. Available from: `https://books.google.cz/books?id=JEdTNwAACAAJ`

[25] Jain, A.; Soner, S.; et al. An approach for extracting business rules from legacy C++ code. In *2011 3rd International Conference on Electronics Computer Technology*, volume 5, April 2011, pp. 90–93, doi:10.1109/ICECTECH.2011.5941963.

[26] Cosentino, V. *A model-based approach for extracting business rules out of legacy information systems*. Dissertation thesis, Nantes, Ecole des Mines, 2013.

[27] Sneed, H. M. Migrating from COBOL to Java. In *2010 IEEE International Conference on Software Maintenance*, Sept 2010, ISSN 1063-6773, pp. 1–7, doi:10.1109/ICSM.2010.5609583.

[28] Kontogiannis, K.; DeMori, R.; et al. Pattern matching for design concept localization. In *Proceedings of 2nd Working Conference on Reverse Engineering*, Jul 1995, pp. 96–103, doi:10.1109/WCRE.1995.514698.

[29] Rasool, G.; Philippow, I. Recovering Artifacts from Legacy Systems using Pattern Matching. volume 2, no. 10, 2008. Available from: `https://waset.org/publications/1409/recovering-artifacts-from-legacy-systems-using-pattern-matching`

[30] VanCourt, T. Reverse Engineering Design Patterns. 2001. Available from: `https://www.bu.edu/caadlab/VanCourtMS.pdf`

[31] Ferenc, R.; Beszédes, A.; et al. Design pattern mining enhanced by machine learning. In *In Proceedings International Conference on Software Maintenance (ICSM 2005). IEEE Computer Society, Los Alamitos*, IEEE Computer Society, 2005, pp. 295–304.

[32] Alhusain, S.; Coupland, S.; et al. Towards machine learning based design pattern recognition. In *2013 13th UK Workshop on Computational Intelligence (UKCI)*, Sept 2013, ISSN 2162-7657, pp. 244–251, doi:10.1109/UKCI.2013.6651312.

[33] Koschke, R. Software Visualization for Reverse Engineering. In *Revised Lectures on Software Visualization, International Seminar*, London, UK, UK: Springer-Verlag, 2002, ISBN 3-540-43323-6, pp. 138–150. Available from: http://dl.acm.org/citation.cfm?id=647382.724659

[34] Lanza, M. Object-Oriented Reverse Engineering. May 2003.

[35] Systems, S. Enterprise Architect — Software Engineering — Import Code. 2015. Available from: http://www.sparxsystems.com/enterprise_architect_user_guide/12/software_engineering/reverseengineersourcecode.html

[36] Don Estes Consulting, I. Business Rule Extraction in Application Modernization Projects. 2018. Available from: https://www.donestes.com/business-rules/

[37] Hay, D.; Healy, K.; et al. Defining business rules. What are they really? The business rules group. 01 2000: pp. 4–5.

[38] LLC, U. Business Rules Extracted from Code. September 2017. Available from: https://www.uniquesoft.com/pdfs/UniqueSoft-Extracting-Business-Rules-From-Code-v11.pdf

[39] Chaparro, O.; Aponte, J.; et al. Towards the Automatic Extraction of Structural Business Rules from Legacy Databases. In *2012 19th Working Conference on Reverse Engineering*, Oct 2012, ISSN 1095-1350, pp. 479–488, doi:10.1109/WCRE.2012.57.

[40] Alawairdhi, M. Static analysis based business logic modelling from legacy system code: Business process model notation (BPMN) extraction using abstract syntax tree (AST). In *2015 International Symposium on Networks, Computers and Communications (ISNCC)*, May 2015, pp. 1–6, doi:10.1109/ISNCC.2015.7238572.

[41] Huang, H.; Tsai, W.-T.; et al. Business Rule Extraction from Legacy Code. In *Computer Software and Applications Conference, Annual International*, volume 0, 01 1996, pp. 162–167.

[42] Normantas, K.; Vasilecas, O. A Systematic Review of Methods for Business Knowledge Extraction from Existing Software Systems. *Baltic Journal of Modern Computing*, volume 1, no. 1-2, 2013: pp. 29–51.

[43] Ducasse, S.; Anquetil, N.; et al. MSE and FAMIX 3.0: an Interexchange Format and Source Code Model Family. Research report, Nov. 2011. Available from: https://hal.inria.fr/hal-00646884

[44] BlueAge. User's guide - application modernization. 2017. Available from: https://wiki.bluage.com/mediawiki/index.php/En_ba_rev_modeling

[45] Brunelière, H.; Cabot, J.; et al. MoDisco: a Model Driven Reverse Engineering Framework. *Information and Software Technology*, volume 56, no. 8, Aug. 2014: pp. 1012–1032, doi:10.1016/j.infsof.2014.04.007. Available from: https://hal.inria.fr/hal-00972632

[46] Algo, D. Deep Algo: Solution. 2018. Available from: https://www.deepalgo.com/en/home

[47] SoftwareMining. SoftwareMining. 2018. Available from: https://www.softwaremining.com/

[48] Imagix. Reverse Engineering Tools - C, C++, Java - Imagix. 2018. Available from: https://www.imagix.com/index.html

[49] valerio. JBREX. 2016. Available from: https://github.com/valeriocos/jbrex

[50] Jirsa, J. Systém pro podporu prodeje vstupenek. 2104.

[51] feenk. jdt2famix. 2018. Available from: https://github.com/feenkcom/jdt2famix

Appendix A

FAMIX

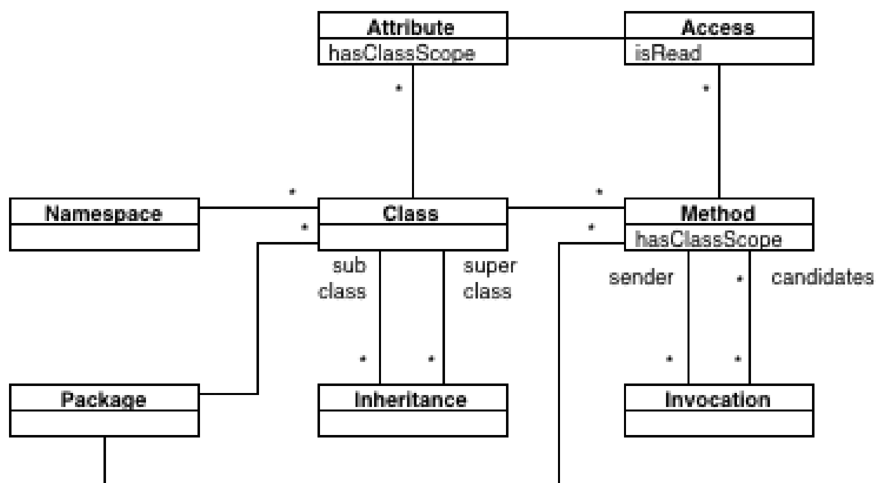

Figure A.1: The FAMIX simplified overview on model elements (used from [16]))

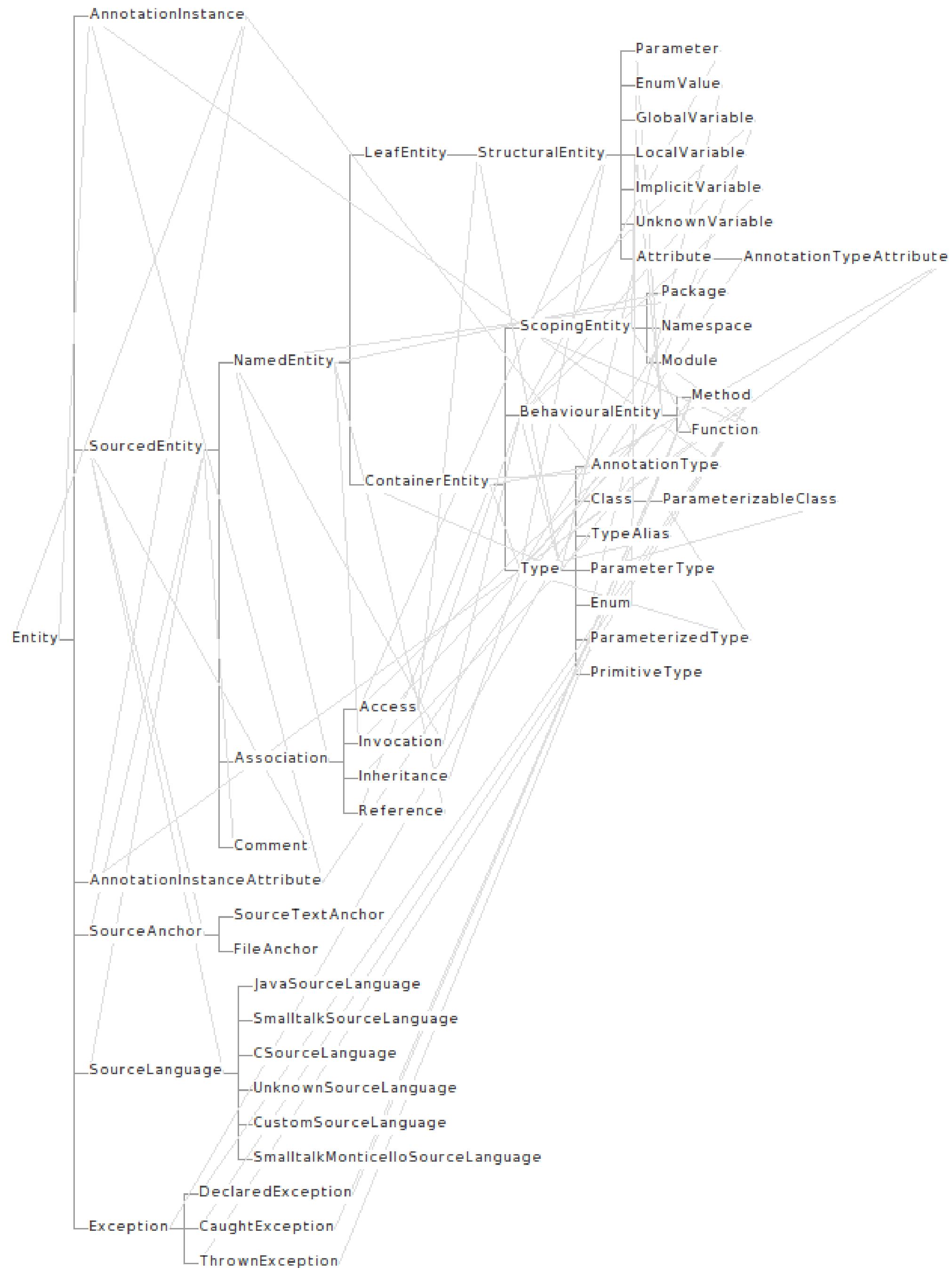

Figure A.2: The FAMIX hierarchy with the symbolised relationships between elements (used from [16]))

Appendix B

Projects' UML

Figure B.1: The Student Information System 1 UML

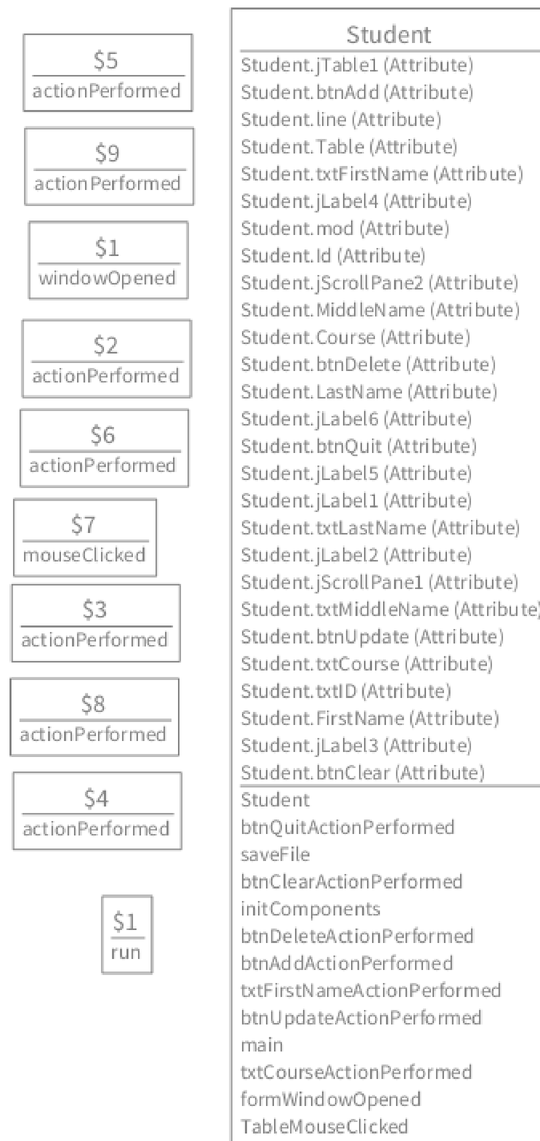

Figure B.2: The Student Information System 2 UML

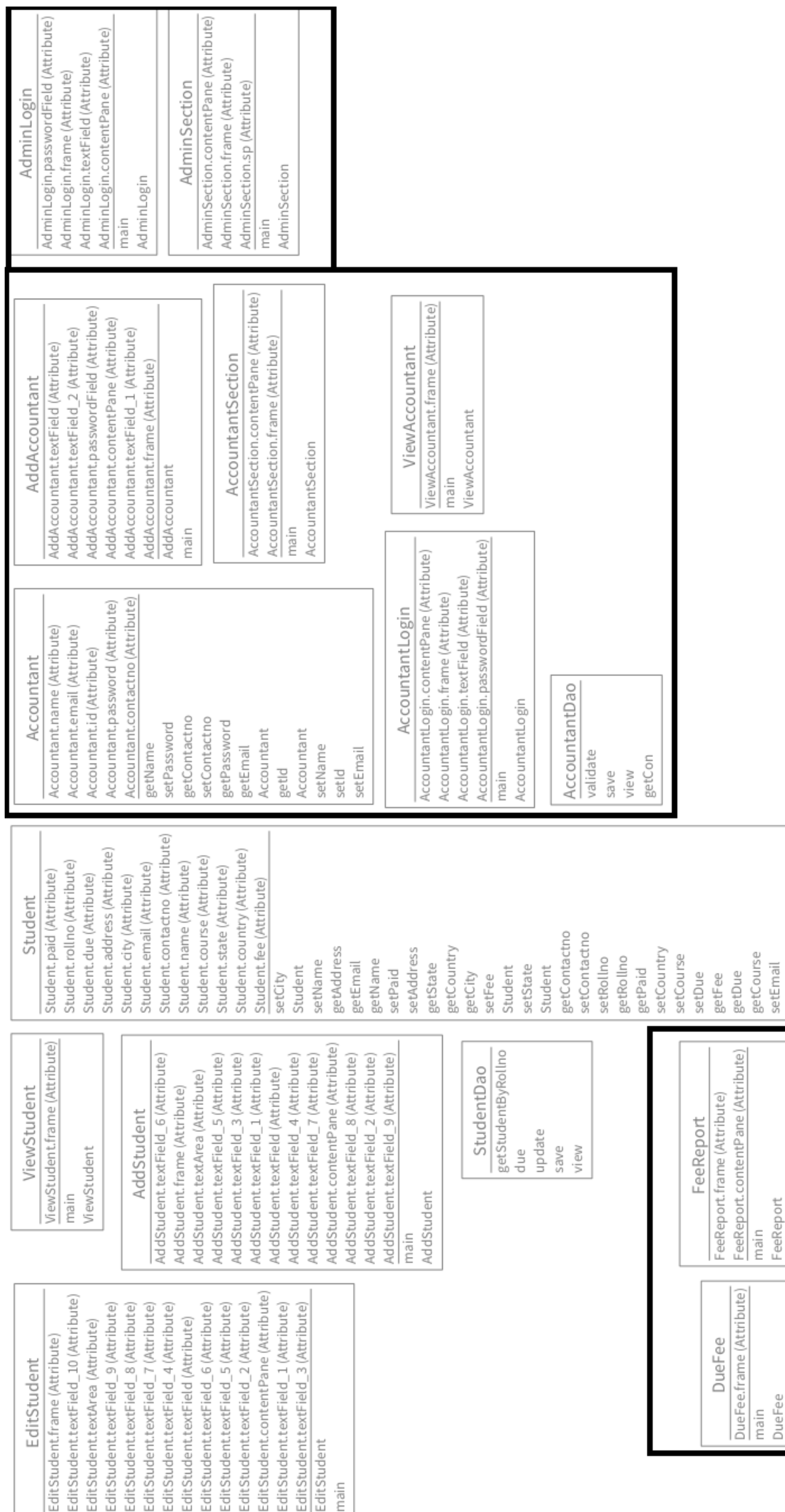

Figure B.3: The Student Management System UML

Appendix C

Contents of CD

- `src` .... the directory of source codes
  - `experimentalImages` . the directory containing the images with moose
  - `sourceCodes` .... The directory with experimental programs
    - `StudentInfoSys` the source codes for experimental project: Student Information System 1
    - `StudentIfoSystem` ..... the source codes for experimental project: Student Information System 2
    - `feereport` ...... the source codes for experimental project:Student Management System
    - `BP-klient` the source codes for experimental project: Ticket system
  - `thesis` .... the directory of LaTeX source codes of the thesis
    - `pics` .... the thesis figures directory
    - `*.tex` .... the LaTeX source code files of the thesis
- `text` .... the thesis text directory
  - `thesis.pdf` .... the Diploma thesis in PDF format
  - `thesis.ps` .... the Diploma thesis in PS format