



**Universiteit
Leiden**
The Netherlands

Opleiding Informatica

Measuring functional volume of a general programming
language based on framework code

Yvo Hu

Thesis Supervisors:

Prof.dr.ir. J.M.W. Visser

Dr. W. Heijstek

Internship Supervisor:

Jeroen Meetsma

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

www.liacs.leidenuniv.nl

23/08/2023

Abstract

Background: Function Point Analysis (FPA) is a decades-old technique of software measurement to measure the functional size of an application. Using these metrics, we are able to gain powerful insights into estimating project effort and costs over a period of time. This technique has traditionally always been done manually by software measurement experts, this introduces subjectiveness into the measurements between individuals. We can limit the amount of subjectiveness in these measurements by inventing algorithms that follow a strict set of rules to apply the FPA techniques. This will be applied to code that utilizes REST API and ORM design patterns. The algorithm itself will be developed using the development framework of BonCode.

Objective: This thesis tries to establish whether or not a low-code model analysis can be adapted to and applied to general programming-language code that makes use of JPA annotations and the Spring framework

Methods: Our approach to this problem is by utilizing a proprietary development framework in Java made specifically for code analysis. This framework allows us to analyze the parse tree of a program written within a particular programming language, and gather metrics on the source code based on various algorithms. Using this framework, we will be able to implement an algorithm based on the official FPA specifications to count function points.

Results: The reports generated by our tool show results that are consistent with the specifications of the IFPUG FPA ISO standard[ISO09]. Furthermore, the benchmark experiments show us that the upper bound of the time complexity is at most linear.

Conclusion: We have shown that automated FPA is indeed possible and feasible, but requires us to recognize implementation-specific patterns within various development frameworks, and invent algorithms to analyze each of these for every project to accurately apply the FPA concepts to the source code.

Contents

1	Introduction	6
1.1	Background and context	6
1.2	Research problem and questions	6
2	Methodology	8
2.1	Problem identification and motivation	8
2.2	Objectives of a solution	8
2.3	Design and development	9
2.4	Demonstration	9
2.5	Evaluation	9
2.6	Communication	9
3	Literature Review	11
3.1	Benefits and challenges of FPA	11
3.2	Existing research on automated FPA	12
3.3	Gaps in the literature	12
4	Definitions (FPA)	13
4.1	Record Element Type (RET)	13
4.2	File Type Reference (FTR)	13
4.3	Data Element Type (DET)	13
4.4	Transaction DETs	13
4.5	Data Functions	14
4.6	Internal Logical File (ILF)	14
4.7	External Interface File (EIF)	15
4.8	Transactional Functions	16
4.9	External Input (EI)	16
4.10	External Output (EO)	17
4.11	External Inquiry (EQ)	18
4.12	Complexity Adjustment Factor	19
5	Definitions (other)	19
5.1	Model View Controller (MVC)	19
5.2	Representational State Transfer (REST) API	20
5.3	Object Relational Mapping (ORM)	20
6	Overview of the BonCode development framework	21
6.1	Token generation	21
6.2	Syntax analysis and grammar construction	21
6.3	Low-code implementation	22

7	Design	23
7.1	Purpose	23
7.2	Scope	23
7.3	Internal Logical File Analyzer	24
7.4	External Interface File Analyzer	24
7.5	Transactional Functions Analyzer	24
8	Implementation	26
8.1	Internal Logical File Analyzer	26
8.2	External Interface File Analyzer	28
8.3	Transaction Functions Analyzer	28
8.4	Reporting	30
8.5	Testing	32
9	Experiments	35
9.1	Results	35
9.2	Time Complexity	37
10	Interpretation of results	42
10.1	Contributions to knowledge	42
10.2	Limitations of the study	42
10.3	Conclusion	43
10.4	Final remarks and suggestions for future work	43
11	Appendices	44
	References	47

Acknowledgement

First and foremost, I want to thank my first thesis supervisor Prof.dr.ir. J.M.W. Visser for bringing me in contact with the people at BonCode, and guiding me meticulously through the research process. He has helped me shape the layout of the thesis and given me feedback on its contents.

Secondly, I want to thank the team at BonCode for their very thorough assistance during the span of the internship. They guided me on a weekly basis through the whole problem statement and the potential solutions whilst making use of their in-house development framework. I was able to learn a lot about FPA, and their specific methods of software architecture.

1 Introduction

The purpose of this thesis is to provide a function point analysis (FPA) method based on framework code for a general-purpose programming language. FPA is a frequently used method for software estimation which counts the number of user inputs, outputs, inquiries, and files to estimate the functional complexity of the software. However, because most FPA techniques are based on functional requirements, they are difficult to utilize with computer languages that don't have a distinct user interface or interaction.

The framework code, which is code that follows a predefined template and serves as a foundation for creating applications, will be leveraged by the suggested FPA approach. The suggested approach will identify the functional point size of the code by gauging its complexity based on the inputs, outputs, and processing logic through the analysis of the framework code. The approach will be evaluated on a well-known programming language and one of its popular frameworks, and contrasted with contemporary manual FPA approaches. This thesis will focus on the analysis of framework code that readily allows for the development of design patterns such as Representational state transfer API (REST API) and data persistence using Object Relational Mapping (ORM).

The findings of this study will advance the field of software estimation by offering a more automated, standardized and effective method to measure software functionality. The FPA approach based on framework code can be used to streamline collaboration between developers and determine the time needed to create software applications, and in particular, calculate the functional volume of each independent section of code.

1.1 Background and context

Function point analysis (FPA) is a software sizing and estimation technique that was developed in the 1970s by Allan J. Albrecht of IBM [Alb79]. It is a widely recognized method for measuring the functional size of software applications, which is important for estimating development effort, project duration, and resource requirements.

FPA quantifies the functionality provided by a software application based on the number of inputs, outputs, inquiries, and files that it processes. The technique has gained popularity over the years due to its ability to provide a standardized and objective measure of software functionality. Additionally, it can help to identify areas of software applications that are complex, error-prone, or require improvement.

The company BonCode has developed an implementation of automated function point analysis of the low-code platform of OutSystems. This enables them to efficiently measure various metrics regarding FPA of various OutSystems projects.

1.2 Research problem and questions

The main goal of this thesis is to answer the following research question:

Can we design an implementation for measuring the functional volume of software written in a general-purpose programming language using a development framework, in line with the low-code implementation of the software company BonCode?

The aforementioned general-purpose programming language will be Java. The frameworks that we will be focusing on in this thesis are Hibernate and the Spring framework. In addition to this, we have also formulated three supporting sub-questions

- Is the method generalizable to several languages and frameworks?
- How accurate is this automated FPA tool compared to manual FPA techniques, and what factors can impact the accuracy of the results?
- How feasible is the tool with respect to the time complexity of the algorithm?

2 Methodology

This section will outline the methodology being used to tackle the research question. It roughly follows the Design Science research process which is a research approach that focuses on creating and evaluating innovative solutions to complex problems. It aims to develop and validate new design artifacts, such as models, methods, processes, and tools, that can be used to address practical challenges [PTG⁺20].

2.1 Problem identification and motivation

While FPA has become a widely accepted technique for measuring software size and effort estimation, it has its limitations. Most FPA techniques are based on functional requirements, which can make it difficult to apply to software applications that lack a distinct user interface or interaction. Additionally, FPA can be time-consuming and error-prone when performed manually, as it requires a thorough understanding of the software application and its functionality.

To address these limitations, automated function point analysis (AFPA) tools have been developed [sco] [Bon]. These tools automate the process of function point counting, providing a faster, more accurate, and consistent way of measuring the functional size of the software. AFPA tools can also help to identify areas of software applications that are complex, error-prone, or require improvement.

Overall, FPA and AFPA techniques are critical for software development teams to estimate the effort required to develop or maintain software systems. By accurately estimating software size and effort, teams can better plan and allocate resources, improve project outcomes, and increase the overall quality of software development.

2.2 Objectives of a solution

The main objective of this research is to design and implement software that is able to determine the functional volume of other software that utilizes the specific framework(s) that we are trying to analyze and have designed an implementation for. In particular, our research aims to categorize specific sections of code and attach a function point volume statistic to each of these. Additionally, we want to be able to see where and what kinds of code components within the code section are used to calculate this statistic and display these appropriately.

FPA provides a common language and framework for discussing the functional requirements and complexity of software applications. This can help to facilitate communication and collaboration between developers, users, and other stakeholders involved in the software development process. In particular, this program will be able to determine the functional size and complexity of the software application, which can be used to estimate the cost, effort, and resources needed to develop and maintain the software. By keeping track of this statistic, it can be used to plan and manage software development projects more effectively by providing a quantitative estimate of the software's functional size and complexity. This can help to identify potential risks and ensure that the project stays on track. By measuring this statistic over multiple time points within an application's development period, we are able to measure the productivity of software development teams, as well as assess the quality of the software produced. This can help to identify areas for

improvement and ensure that the software meets the needs of its users. Overall, FPA is needed to ensure that software development projects are successful, efficient, effective, and within the prospected budget.

2.3 Design and development

To solve this problem, we will need to design and develop software to analyze relevant source code and attach metrics to the aforementioned pieces of code.

Function Point Analysis is traditionally used as a manual way of counting the functional volume of a piece of software based on the amount of business functionality an information system (as a product) provides to a user. We will therefore need to translate this concept via an algorithmic set of procedures to quantify the functionality of the software. The software may be written in many different ways, i.e. there are equivalent solutions to achieve the same end result, but the code always follows the same grammatical structure. Thus, we are able to recognize similar components within the code. This is especially true in the case of framework code. Via this framework code, and in particular, frameworks that allow for the development of REST API and data persistence using ORM, we are able to relate similar concepts that are used within FPA.

We will make use of the proprietary software development framework of BonCode to help us accomplish this goal. This framework is specifically designed for developers to analyze source code and attach relevant metrics to the code based on the structure of the code.

2.4 Demonstration

The end result will be an extension of the code analysis tool developed by BonCode. It will run alongside several other code analysis extensions to form a part of the complete code analysis tool. Code segments will be visually displayed within the tool and properly annotated with the corresponding function points and concepts within FPA to demonstrate the efficacy of the tool.

2.5 Evaluation

To evaluate the tool, we need to observe and measure how well the tool supports the aforementioned solution to the problem. We will compare the FPA metrics of the analyzed code with the specifications in the relevant FPA literature.

Important metrics include the following:

- Inputs, Outputs
- Date element types, Record element types, File type references
- Functional points

2.6 Communication

The findings of this thesis should be of relevance to professionals active in the field of software measurement. FPA is an integral part of it, and it would allow for a way to do FPA in a more

structured, consistent, and efficient manner. Consequentially, companies who utilize the services of software measurement companies may benefit from improvements in FPA to more effectively gauge the various metrics of FPA within their codebase.

3 Literature Review

In this section, we will discuss some definitions of function point analysis, background information, and relevant literature.

3.1 Benefits and challenges of FPA

Benefits of Function Point Analysis [ISO09] [HK91] [Sym88]:

- Improved project estimation: Function Point Analysis helps to accurately estimate the size and complexity of a software system, which is essential for project planning and budgeting. This results in a more realistic and accurate project plan, which can help to avoid delays and budget overruns.
- Improved communication: Function Point Analysis provides a common language for communicating the functional requirements of a software system between different stakeholders, including developers, project managers, and business analysts. This improves communication and ensures that all stakeholders have a clear understanding of the requirements.
- Improved productivity: Function Point Analysis helps to identify areas of the software system that can be streamlined or automated, leading to improved productivity and efficiency. By focusing on the functional requirements of the system, developers can optimize the system to meet business needs.
- Improved quality: Function Point Analysis can help to identify functional requirements that are not necessary or redundant, leading to a higher quality software system. By removing unnecessary requirements, developers can focus on delivering a system that meets business needs, while avoiding unnecessary complexity and costs.

Challenges of Function Point Analysis:

- Requires expert knowledge: Function Point Analysis requires expert knowledge and experience to accurately identify and categorize the functional requirements of a software system. It can be challenging for less experienced developers to accurately estimate functional points, leading to inaccuracies in project estimation.
- Can be time-consuming: Function Point Analysis can be a time-consuming process, particularly for larger and more complex software systems. This can lead to delays in project planning and execution.
- May not be suitable for all software systems: Function Point Analysis may not be suitable for all types of software systems, particularly those that have unique or custom requirements. It may also be less effective for systems that are heavily reliant on technology or implementation details.

In summary, Function Point Analysis is a valuable tool for measuring and managing the functional requirements of software systems. It provides several benefits, including improved project estimation, communication, productivity, and quality. However, it also has some challenges, including the need for expert knowledge, time consumption, suitability for certain types of systems, and potential customization requirements.

3.2 Existing research on automated FPA

There has been extensive research on Function Point Analysis (FPA) over the past several decades. Much of this research has focused on the effectiveness of FPA as a software measurement technique and its ability to accurately estimate software project effort, schedule, and cost [Ver02]. Some key areas of research related to FPA include:

- Empirical studies: Several empirical studies have evaluated the effectiveness of FPA in estimating software project effort, schedule, and cost. These studies have generally found that FPA is a reliable and effective technique for software measurement [Fur97] [AR96].
- Comparison studies: Numerous studies [ZN12] [GTS⁺16] have compared FPA with other software measurement techniques, such as Lines of Code (LOC), to determine their relative effectiveness in estimating software project effort, schedule, and cost.
- Framework development: Research has been conducted to develop frameworks for using FPA within specific domains, such as web development, embedded systems, and agile development. These frameworks aim to provide guidance on how to apply FPA in these domains to improve its effectiveness [FTB06] [MM93] [SC04] .
- FPA automation: Several studies have investigated the automation of FPA to improve its efficiency and accuracy. These studies have developed automated tools for FPA and evaluated their effectiveness in improving FPA accuracy and efficiency [FTB06].

3.3 Gaps in the literature

As FPA was designed very early on in the history of software estimation, little about future developments have been taken into consideration in the design of FPA. This opens up various gaps within the overall scope of FPA.

- Limited research on FPA in the context of automation. FPA has traditionally always been done manually by software estimation experts. This practice has become more unfeasible the bigger a project becomes. Therefore, various implementations have been made to try and automate this process. These are based on UML diagrams or low-code implementations such as OutSystems. There are however, a range of other different domains that are still yet unexplored.
- Various standards: There are differences in the standardization of the application of FPA, which can lead to variations in the results obtained by different practitioners using different implementations of the different standards. There is a need for more centralized authority and more research on how to standardize the application of FPA to improve its accuracy and consistency. There are existing standardization bodies such as IFPUG, NESMA, and COSMIC, though these all have their own standards (however small the differences may be). Ideally, there would only need to be a single central authority on the standardization of FPA.

Addressing these gaps in the literature helps to improve the effectiveness and applicability of FPA as a software measurement technique as a whole. In this thesis, we will try to address a gap in the application of automated function point analysis, namely the analysis of software that is written in a general-purpose programming language that utilizes REST API and ORM design patterns.

4 Definitions (FPA)

In the following section, we will define a list of terms that are commonly used within FPA. These definitions are taken from the function point modeler manual [Fun], but these, in turn, follow directly from the FPA standards as defined by the IFPUG [ISO09]

4.1 Record Element Type (RET)

A Record Element Type (RET) is a distinguishable subset of data elements found within either an ILF or an EIF (these will be elaborated on in the upcoming sections). The most effective approach is to examine logical clusters of data to aid in their identification. The concept of RET will receive comprehensive coverage in the chapters dedicated to internal logical files and external interface files.

Examples of a RET include other tables within a database that are being referenced within a table via a foreign key.

4.2 File Type Reference (FTR)

An FTR is a file type referenced by a transaction. An FTR must also be an internal logical file or an external interface file [ISO09].

Examples of an FTR are tables within a database that are being accessed by a transaction function (discussed in a later section).

4.3 Data Element Type (DET)

A Data Element Type (DET) is a distinct, identifiable field that is not repeated in a recursive manner. DETs encompass dynamic information rather than static content. Dynamic fields are either extracted from a file or generated from other DETs found within a File Type Reference (FTR). Moreover, a DET can trigger transactions or offer supplementary details concerning transactions. In cases where a DET is recursive, only the initial instance of the DET is taken into account, not every occurrence.

Examples of DETS are columns within a table of a database or input fields on a form.

	DET	RET	FTR
ILF & EIF	X	X	
TF	X		X

4.4 Transaction DETs

The following is a list of what things may be classified as a DET from a functional requirements perspective as a user [ISO09].

- External Inputs: Data Input Fields, Error
- External Outputs: Data Fields on a Report, Calculated Values, Error Messages, and Column Headings that are read from an ILF. An EQ and EO can have input and output sides.
- External Inquiries: Input Side - field used to search by, the click of the mouse. Output side - displayed fields on a screen.

4.5 Data Functions

Data functions represent the functionality provided to the user to meet internal and external data requirements. Data functions are either internal logical files or external interface files [ISO09].

4.6 Internal Logical File (ILF)

An Internal Logical File (ILF) is a user-identifiable collection of logically connected data or control details that are confined within the application's scope. The core objective of an ILF is to house data that is managed through one or more elementary processes within the application being evaluated. Moreover, for data or control information to qualify as an ILF, both of the subsequent IFPUG counting rules must also be applicable [ISO09]:

- The group of data or control information is logical and user identifiable.
- The group of data is maintained through an elementary process within the application boundary being counted.

Examples of ILFs Samples of things that **can** be ILFs include:

- Tables in a relational database.
- Flat files.
- Application control information, perhaps things like user preferences that are stored by the application.
- Lightweight Directory Access Protocol data stores.

DETs			
RETS	1-19	20-50	51+
1	L	L	A
2-5	L	A	H
6+	A	H	H

Complexity	No. of Function Points
Low (L)	7
Average (A)	10
High (H)	15

4.7 External Interface File (EIF)

An external interface file (EIF) designates a collection of logically linked data or control information that can be identified by users. This collection is referenced by the application, yet it remains under the jurisdiction of another application's boundary. The core purpose of an EIF is to retain data referred to by one or more elementary processes within the application's boundary that's being assessed. Therefore, an EIF is counted for an application only if it exists within an Internal Logical File (ILF) of another application.

DETs			
RETS	1-19	20-50	51+
1	L	L	A
2-5	L	A	H
6+	A	H	H

Complexity	No. of Function Points
Low (L)	5
Average (A)	7
High (H)	10

4.8 Transactional Functions

Transactional functions pertain to the interactions between users and the application, leading to the execution of specific data processing or transactions. These functions are classified according to the primary user's intention or objective while interacting with the software. Within FPA, transactional functions fall into three distinct categories.

4.9 External Input (EI)

An external input (EI) denotes a basic procedure that handles data or control details originating from beyond the application's confines. The primary goal of an EI involves the management of one or more Internal Logical Files (ILFs) and/or the potential modification of the system's behavior.

Examples of EIs include [\[ISO09\]](#):

- Data entry by users.
- Data or file feeds by external applications.

DETs			
FTRs	1-4	5-15	16+
0-1	L	L	A
2	L	A	H
3+	A	H	H

Complexity	No. of Function Points
Low (L)	3
Average (A)	4
High (H)	6

4.10 External Output (EO)

An external output (EO) signifies a fundamental procedure that dispatches data or control details beyond the confines of the application's scope. The principal purpose of an external output is to provide users with information through processing logic that extends beyond the mere retrieval of data or control particulars. This processing logic must encompass at least one mathematical formula or calculation or lead to the creation of derived data. Additionally, an external output might oversee one or more Internal Logical Files (ILFs) and potentially impact the system's behavior.

EO examples include [\[ISO09\]](#):

- Reports created by the application being counted, where the reports include derived information.

DETs			
FTRs	1-5	6-19	20+
0-1	L	L	A
2-3	L	A	H
4+	A	H	H

Complexity	No. of Function Points
Low (L)	4
Average (A)	5
High (H)	7

4.11 External Inquiry (EQ)

An external inquiry (EQ) refers to a basic operation that transmits data or control details beyond the confines of an application. The core purpose of an external inquiry is to furnish users with information by fetching data or control specifics. These operations lack mathematical formulas or computations in their processing logic, and they don't generate any derived data. They don't maintain Internal Logical Files (ILFs) during processing, nor do they modify the system's behavior.

Examples of EQs include [\[ISO09\]](#):

- Reports created by the application being counted, where the report does not include any derived data.

DETs			
FTRs	1-5	6-19	20+
0-1	L	L	A
2-3	L	A	H
4+	A	H	H

Complexity	No. of Function Points
Low (L)	3
Average (A)	4
High (H)	6

4.12 Complexity Adjustment Factor

There also exists a concept called the "Complexity Adjustment Factor" (CAF). It is a numerical value that reflects the additional complexity and effort required to complete a software project due to certain factors. The CAF is typically applied to adjust the estimated effort, time, or cost of a project.

The Complexity Adjustment Factor (CAF) consists of 14 "General System Characteristics", or GSCs. These GSCs represent the characteristics of the application under consideration. Each is weighted on a scale from 0 to 5. [Fun].

The GCSCs

- Data Communication
- Distributed data processing
- Performance
- Heavily used configuration
- Transaction rate
- Online data entry
- End-user efficiency
- Online update
- Complex processing
- Reusability
- Installation ease
- Operational ease
- Multiple sites
- Facilitate change

We will not be able to include this factor into our calculations because it is not purely dependent on the written code, but a lot of external factors as well.

5 Definitions (other)

5.1 Model View Controller (MVC)

The MVC model is a design pattern frequently used in application development. Its use is to separate the concerns of an application's user interface, data management, and control logic.

5.2 Representational State Transfer (REST) API

A REST API is a set of conventions and guidelines for building and interacting with web services. It is an architectural style for designing networked applications, particularly web services, that follows a set of principles to create scalable and efficient communication between different software systems.

REST APIs are based on the concept of resources, which are identified by Uniform Resource Locators (URLs). Each resource represents a piece of data or functionality that can be accessed and manipulated using standard HTTP methods such as GET, POST, PUT, and DELETE. These methods correspond to actions like retrieving data, creating new data, updating data, and deleting data, respectively.

5.3 Object Relational Mapping (ORM)

ORM stands as a collection of principles and methodologies designed to simplify the interaction between object-oriented programming languages and relational database management systems. The core objective of ORM is to close the divide between the object-oriented model utilized in programming languages and the relational model that databases employ, characterized by tables, rows, and columns. Within this framework, ORM tools offer a mechanism to depict database tables as classes or objects, seamlessly linking their attributes to corresponding columns within the database structure.

6 Overview of the BonCode development framework

The proprietary development framework of BonCode is a piece of software that has been maintained and refined over many years. It is primarily written in Java and is a crucial part of numerous other code analysis tools that they have developed. We will be using this development framework to design and implement a version of automated function point analysis in Java with a contemporary Java framework that persists data in some recognizable format and allows us to easily implement a REST API.

The BonCode development framework closely resembles the formal definition of grammar in the context of programming languages. It has a functionally similar working to some early stages within a compiler but has a technically different design altogether. Because this development framework is the intellectual property of BonCode, we will not be able to thoroughly dissect the implementation details of the framework because of an NDA clause. We will however try to explain the general workings of software in the upcoming sections.

6.1 Token generation

Similarly to a lexical analyzer in a conventional compiler, the development framework of BonCode utilizes a token generation program to categorize each individual code component (i.e. a character, or group of characters) and generates a corresponding token. Each token may correspond to multiple code snippets that are each at least a single character long. An example of this are the operators such as `+`, `-`, `*`, `/`, `&&`. All of these are recognized as the same type of token, namely the operator token. The original information of the code snippet is retained whilst generating the token in case it is needed to differentiate between these tokens in future analysis and to display for general reporting.

6.2 Syntax analysis and grammar construction

And again, the development framework utilizes a syntax analyzer similar in functionality to a syntax analyzer in a conventional compiler. However, we have to construct our own grammar for the language beforehand. A more specific grammar for each implementation allows us to analyze the syntax, but a singular grammar for the entire language has greater generalizability. This part of the framework allows us to group an arbitrary number of tokens or constructs together in a specified order, and simultaneously categorize them. These categories are also formally known as constructs and may serve as building blocks for even higher-level constructs. The framework allows us to fine-tune the exact construction of these constructs to a degree in which we are able to analyze the code with a high if not total level of control. The constraints that we are able to put on a construct include but are not limited to: The sequence of tokens/constructs, The number of tokens/constructs, and the optional or mandatory occurrence of tokens/constructs.

Whenever a section of code is analyzed, it will try to match the highest-order construct, and then try to match potential lower constructs. For example, there exists a class declaration [Wik23], which is in its entirety a construct of its own, but it simultaneously contains constructs within itself like its data members or method declarations. This information will be retained in the analysis so

that we know when and where to perform calculations based on this data.

6.3 Low-code implementation

The current low-code implementation focuses on OutSystems. OutSystems is a low-code application development platform that enables organizations to build, deploy, and manage complex web and mobile applications with minimal traditional hand-coding. It is designed to help businesses accelerate their software development processes by providing visual development tools, pre-built components, and a platform for managing the entire application lifecycle. All aspects of an OutSystems project are defined in files within a structured format. BonCode has developed an AFPA implementation to analyze these files and generate informative reports based on this analysis. The analysis of these files also makes use of the same development framework that is used in this thesis.

7 Design

It is necessary to create a general outline for the automated function point analysis software. We, therefore, need to design a blueprint to identify the key requirements and objectives of the software, as well as to provide a clear and detailed process for achieving these goals. Function point analysis is more commonly performed based on functional requirements, the product features, or the desired operations of a program, rather than the technical specifications i.e. the underlying code. Regardless, we will attempt to design an automated function point analysis program that utilizes the structure of frameworks, with clear and concise code layouts, to develop a method to count functional points within the code.

The main objective of this function point analysis tool will primarily focus on the functionalities within a web development framework, and in particular, the functionalities regarding database persistence. The function point analysis concepts will therefore be applied in the context of server-side web development, using the commonly used ORM and REST API architectural styles.

7.1 Purpose

The purpose of implementing AFPA is to streamline the function point analysis process and enhance its reliability. By automating the counting and calculation of function points, AFPA reduces the manual effort required and ensures consistent and accurate results. It enables organizations to conduct function point analysis in a more efficient and cost-effective manner.

7.2 Scope

The scope of AFPA includes automating the various steps involved in function point analysis, including function identification, counting, and reporting.

Ideally, we would like to incorporate every element of function point analysis into our implementation, but it is evident that not all frameworks are the same and may differ in the functionalities that they are able to provide. Consequently, there may not be a relevant code structure to do such an analysis on. Additionally, the design should be generalizable to multiple languages/frameworks, so that it has a wider practical application, than just the implementation that this thesis is focused on.

For a complete function point analysis, our design should incorporate the following elements:

- Internal Logical File Analyzer
- External Interface File Analyzer
- Transactional Functions Analyzer

7.3 Internal Logical File Analyzer

This component should be able to find the source code that manages all the data stores within an application. A model type commonly represents a table in the application's database.

Every internal logical file should ideally be separated into its own file. A common concrete example of this in code would be a class declaration that acts as an entity, whilst its attributes act as its properties. Consequently, there should be a clearly defined structure to demarcate the layout of the entity and its properties. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic, and rules of the application.

The properties of the entity should each be counted separately as a data element type, as each of these signifies a column within a table. The exact way in which these columns are represented within the code should be clearly demarcated, as there may also be some properties within the code that are not directly used as columns.

Record element types may be represented as another attribute or a list of attributes within the code. These must map to the source code of another entity in the same repository. The data element types of these other entities are added to the overall sum of the original encompassing entity

In the context of function point analysis, the model may therefore be seen as an internal logical file, as it directly serves as a data store for the application.

7.4 External Interface File Analyzer

As was detailed earlier, an external interface file is essentially an internal logical file that is maintained by another application. This means that there is no data persistence being done in the current application, and no relevant source code for the data persistence in the application's software. The only way to obtain such data is via requests made to another application. This may be represented by requests made to other servers. A common method to retrieve data from another server is by making an HTTP request.

An example of this could be to recognize what kind of data the server receives. If the data is neatly structured via common data formats such as JSON or XML, then we would be able to gauge the different data element types and record element types of the data.

7.5 Transactional Functions Analyzer

This component should be able to find the source code that manages all the transactional functions within an application. For this thesis, we try to apply FPA concepts in the context of a REST API controller. The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it, and then passes the input to the model.

These transactional functions could either be separated in their own file, or they should be

grouped together.

Conventionally, each model would have an associated controller; for example, if the application had a Client model, it would typically have an associated controller as well.

The controller is typically structured as a class, with various similarly structured methods. Each of these methods serves as a separate REST API endpoint, which is uniquely accessed using specific Hypertext Transfer Protocol (HTTP) headers. The controller receives input from the user in the form of HTTP requests via these endpoints, and may correspondingly create, read, update, or delete something from the model.

As was explained earlier in the thesis, there are three types of transactional functions (EI, EO, EQ). Depending on how the REST endpoints handle each of these HTTP requests, these may be categorized differently. These are outlined as the following:

- External Input (EI):
The request updates or inserts data into the model without returning any other from the model back to the user.
- External Output (EO):
The request retrieves data from the model, performs calculations and creates a derived result, and returns it to the user.
- External Inquiry (EQ):
The request retrieves data from the model and returns it to the user.

If we try to apply function point analysis to these, then the REST API endpoints may be seen as transactional functions.

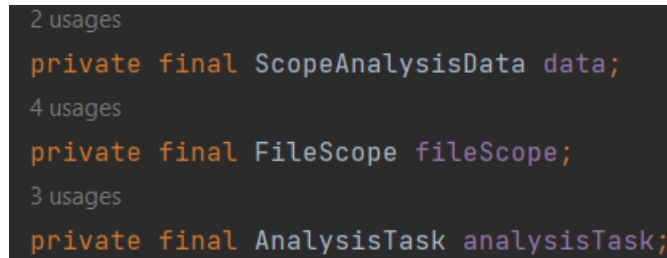
8 Implementation

This section will detail the implementation of our automated function point analysis tool. The total amount of code that has been written for the abstraction layer. The tool itself is written in Java, and this specific implementation equals approximately 1200 lines.

As has been detailed in the previous chapter, this tool will concern itself with the back-end of web applications utilizing the ORM and REST API design patterns. This drastically narrows down the number of suitable software that we aim to analyze, and as such we will focus on software that utilizes the following frameworks:

- Java Spring Framework
- Java Hibernate (JPA)

As is good practice with most software, it is a good idea to be generalizable. In addition to this, there are also a few functionalities that are dependent on the development framework that we are using. These are used to analyze the code structure in general and are not specific to function point analysis in general, and are needed in a lot of different components.



```
2 usages
private final ScopeAnalysisData data;
4 usages
private final FileScope fileScope;
3 usages
private final AnalysisTask analysisTask;
```

Figure 1: BonCode specific implementation details of their development framework. This is needed to analyze the relevant source files. This is both needed in the ILF analyzer, and the Transaction Function analyzer

Therefore, we need to implement an abstraction layer like abstract classes to minimize code duplication. This is to ensure that the code remains clean and more maintainable.

8.1 Internal Logical File Analyzer

There are a couple of things that should be common in all implementations of an internal logical file analyzer. To count the function points of an internal logical file, we need to keep track of a couple of metrics. These are the total record element types (totalRET) and the total data element types (totalDET). Based on these values, we will be able to calculate the total function points (totalFP) of an ILF.

This abstraction layer also shares similar methods between all implementations. It uses components from the development framework to perform the necessary metrics of an internal logical file.

```

private Construct internalLogicalFileNode;
4 usages
private int totalRET;
5 usages
private int totalDET;
9 usages
private int totalFP;

```

Figure 2: FPA relevant metrics to keep track of within the ILF analyzer class.

```

1 usage 2 implementations yvoh
public abstract Construct findInternalLogicalFile(FileScope fileScope);
4 usages 2 implementations yvoh
public abstract int calculateTotalDET_ILF(Construct internalLogicalFileDeclaration);
2 usages 2 implementations yvoh
public abstract int calculateTotalRET_ILF(Construct internalLogicalFileDeclaration);

```

Figure 3: Methods of the ILF analyzer class.

- **findInternalLogicalFile:**

This method takes a parameter containing information about every source file, and depending on its contents, decides whether this file is used for persistence, and may therefore be classified as an internal logical file. If it is indeed classified as an internal logical file, it returns a corresponding object which may be used in other methods.

We can find these files by searching for class declarations annotated with the entity annotation.

```

@Entity
@Table(name = "owners")
public class Owner extends Person {

```

Figure 4: A code snippet that utilizes the Hibernate framework that our ILF analyzer can analyze. The @Entity annotation marks a file as an ILF.

- **calculateTotalDET_ILF:**

This method takes a valid object returned from the method findInternalLogicalFile and performs calculations to count the total amount of data element types.

Once we've located the internal logical file, we can find its data element types by looking for data members which have been annotated with the column annotation.

```
@Column(name = "address")
@NotEmpty
private String address;
```

Figure 5: A code snippet that utilizes the Hibernate framework that our ILF analyzer can analyze. It corresponds to a DET of an ILF.

- **calculateTotalRET_ILF:**

This method takes a valid object returned from the method `findInternalLogicalFile` and performs calculations to count the total amount of record element types.

Once we've located the internal logical file, we can find its record types by looking for data members which have been annotated with the "one to many annotation" or other similar annotations.

```
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
@JoinColumn(name = "owner_id")
@OrderBy("name")
private List<Pet> pets = new ArrayList<>();
```

Figure 6: A code snippet that utilizes the Hibernate framework that our ILF analyzer can analyze. It corresponds to a RET.

8.2 External Interface File Analyzer

Though an EIF is a core concept used within FPA, we have not been able to relate to such a concept within the utilized frameworks of this thesis. We have therefore not included it in our FPA extension.

8.3 Transaction Functions Analyzer

Like internal logical files and external interface files, transaction functions are also one of the measurement components used to assess the functional size of a software system. To quantify the number of function points of a transaction function, we need to determine the amount of data element types and file type references used within the transaction function.

Similarly to the ILF, we are able to utilize common functionalities and data members within an abstraction layer to make the code more manageable. The most important ones have been listed down below

```

4 usages
private StandardCodeAnnotationType transactionType;
8 usages
private Construct action;
4 usages
private int totalDET;
5 usages
private Set<String> uniqueTablesReferenced;
8 usages
private int totalFP;

```

Figure 7: Data members of the Transaction Function Analyzer (TFA) It keeps track of the transaction type, the analyzed code snippet, and the relevant FPA metrics

```

1 usage 1 implementation new *
public abstract Set<String> gatherUniqueTablesReferenced(AnalysisTask analysisTask);
1 usage 1 implementation new *
public abstract List<FPAInput> retrieveInputs(Construct transaction, AnalysisTask analysisTask);
1 usage 1 implementation new *
public abstract List<Integer> gatherNrOfInputsFromIncomingReferences(List<FPAInput> inputs,

```

Figure 8: The main methods of the TFA

- **gatherUniqueTablesReferenced:**

This method analyzes the transaction and uses a reference resolving technique to locate the definition of all function calls. If a certain function call corresponds to a class that handles transaction management, then the contents of this function may be analyzed to locate all file-type references used within this method.

In this example, the function `this.owners.findById(ownerId)` may be classified as such a function, and we are able to add the string "owner" as a unique file type reference.

```

@GetMapping("/owners/{ownerId}/edit")
public String initUpdateOwnerForm(@PathVariable("ownerId") int ownerId, Model model) {
    Owner owner = this.owners.findById(ownerId);
    model.addAttribute(owner);
    return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
}

```

Figure 9: A code snippet that utilizes the Spring framework that our TF analyzer can analyze. It corresponds to a transaction function

- **retrieveInputs**

```
@Query("SELECT owner FROM Owner owner left join fetch owner.pets WHERE owner.id =:id")
@Transactional(readOnly = true)
Owner findById(@Param("id") Integer id);
```

Figure 10: A code snippet that utilizes the Spring framework that our TF analyzer can analyze. This part is very implementation specific and may not occur in other frameworks. It is a SQL query that allows us to extract referenced tables

This method analyzes the transaction and determines the inputs to the transaction function based on the function parameters, which may then be counted as data element types. The return value is a list of integers, each corresponding to a parameter in the transaction function.

In the following example, the transaction function contains several input parameters. We are able to specifically handle input parameters that contain the Valid annotation and determine the number of total data elements of this input by using reference resolving to find the class definition of Owner and its corresponding data members which may be associated as data element types based on the specific FPA implementation.

```
@PostMapping("/owners/new")
public String processCreationForm(@Valid Owner owner, BindingResult result) {
```

Figure 11: A code snippet that utilizes the Spring framework that our TF analyzer can analyze. It contains an @Valid annotation. We know from the Spring documentation, that it validates whether or not the resource contained within an HTTP request corresponds to the ORM design pattern

- **gatherNrOfInputsFromIncomingReferences** This function aggregates the total data element types from each of the inputs returned by the function retrieveInputs and gives a total of the whole transaction function.

8.4 Reporting

Thanks to the development framework, we are able to easily integrate our FPA extension with the code analysis tool of BonCode. This allows us to easily annotate sections of code with relevant metrics, and display them on a page.

- **Trend** The BonCode Analysis tool has provided us with a method to allow us to analyze multiple snapshots of a code repository at different times. This allows us to track the progression of the code repository with respect to the FPA metrics (and other metrics too) for each unique folder and its subfolders.
- **Internal Logical File** The following picture shows the annotations as displayed within the BonCode code analysis editor. According to the FPA specifications and specific to the implementation using the spring framework, this piece of code should contain 7 function points in correspondence with five DETs and two RETs. Three of the DETs and all two RETs come from the Owner class. The DETs are data members which have been annotated

with the Column annotation. The one RET is a data member which has been annotated with the OneToMany annotation in addition to the ILF itself which is also a RET.

```
spring-petclinic-main/src/main/java/org/springframework/samples/petclinic/owner/Owner.java
...
36  /**
37   * Simple JavaBean domain object representing an owner.
38   *
39   * @author Ken Krebs
40   * @author Juergen Hoeller
41   * @author Sam Brannen
42   * @author Michael Isvy
43   * @author Oliver Drotbohm
44   */
45  @Entity
46  @Table ( name = "owners" )
47  public class Owner extends Person {
48
49      @Modifier, FPA - Internal logical file - FP: 7 - DET: 5 - RET: 2, ClassDeclaration
50      @NotEmpty
51      private String address ;
52
53      @Column ( name = "city" )
54      @NotEmpty
55      private String city ;
56
57      @Column ( name = "telephone" )
58      @NotEmpty
59      @Digits ( fraction = 0, integer = 10 )
60      private String telephone ;
61
62      @OneToMany ( cascade = CascadeType . ALL , fetch = FetchType . EAGER )
63      @JoinColumn ( name = "owner_id" )
64      @OrderBy ( "name" )
65      private List < Pet > pets = new ArrayList < > ( ) ;
66
67      public String getAddress ( ) {
68          return this . address ;
69      }
70
71      public void setAddress ( String address ) {
```

Figure 12: The visual editor of BonCode that allows for dissection of the source code. It shows the Owner Class

The other two DETs come from the derived base class Person, which also has Column annotated data members.

- **Transactional Functions** Five DETs originate from the Owner parameter, that corresponds to the Column annotated data members in the ILF. An additional DET is present because of the RequestParam argument which is necessary for the endpoint.

spring-petclinic-main/src/main/java/org/springframework/samples/petclinic/model/Person.java

```
18     import jakarta . persistence . Column ;
19     import jakarta . persistence . MappedSuperclass ;
20     import jakarta . validation . constraints . NotEmpty ;

22     /**
23      * Simple JavaBean domain object representing an person.
24      *
25      * @author Ken Krebs
26      */
27     @MappedSuperclass
28     public class Person extends BaseEntity {

30         @Column ( name = "first_name" )
31         @NotEmpty
32         private String firstName ;

34         @Column ( name = "last_name" )
35         @NotEmpty
36         private String lastName ;
```

Figure 13: The visual editor of BonCode that allows for dissection of the source code. It shows the Person class upon which the Owner class is derived

```
@GetMapping ( "/owners" )
public String processFindForm ( @RequestParam ( defaultValue = "1" ) int page , Owner owner , BindingResult result ,
    Identifier, unresolved reference, FPA - External output - FP: 8.00 - DET: 6 - FTR: [owner], Annotation
```

Figure 14: The transactional functions are similarly annotated with these relevant metrics

```
@GetMapping ( "/owners" )
public String processFindForm ( @RequestParam ( defaultValue = "1" ) int page , Owner owner , BindingResult result ,
    Model model ) {
```

8.5 Testing

During the development of this tool, many different components have been written to help conceptualize the tool. Some of these components are dependent on code that existed before the start of this thesis within the BonCode development framework, and some have been newly developed to aid specifically in the FPA. We will limit the testing to a select few which have been specifically developed for this tool.

To ensure the correct behaviour during and after the software development process, we need to apply some form of unit testing. We have applied this in some limited capacity.

All the tests will follow a similar layout to the one shown down below. Due to the intercon-

nected components with the development framework, we need to find the correct data and pass complex objects as arguments to finally test the newly developed components.

```
@Test
public void testSpringILFAnalyzer_totalDET() throws Exception {
    CodeBase codeBase = new CodeBase(tmpDir.getAbsolutePath());
    codeBase.createTestScope(AnalysisConfiguration.Language.JAVA_FP_RESEARCH);

    AnalysisRunner analyzer = new AnalysisRunner();
    ScopeAnalysisData data = analyzer.analyze(codeBase).values().iterator().next();
    List<FileScope> filescopees = data.getFileScopes();
    for(FileScope fileScope : filescopees) {
        if(fileScope.getFileName().equals("SpringILFAnalyzer.javatest")){
            InternalLogicalFileAnalyzer ILF = new SpringInternalLogicalFileAnalyzer
            ILF.analyze();

            int result = ILF.getTotalDET();
            int test = 3;
            assertEquals(test, result);
        }
    }
}
```

Figure 15: In this case, we add some source code to the file SpringILFAnalyzer.javatest, instantiate our analysis class, and test the result with an expected value

Similar unit tests have been made for the following functions:

```
public void testSQLHelper() throws Exception {
```

```
public void testSpringILFAnalyzer_totalFP() throws Exception {
```

```
public void testSpringILFAnalyzer_totalRET() throws Exception {
```

```
public void testTFA_totalFP() throws Exception {
```

9 Experiments

In this section, we will discuss the results obtained from the experiments.

9.1 Results

We have earlier shown fragments of how the tool functions when it is working as intended. However, there are situations where the analysis will break down.

- **Different approaches** Like software in general, there are multiple possible ways to write the same functionality. The Spring framework is no different, and that may pose a problem in FPA. Between software repositories that use the Spring framework or Hibernate, there may be a need to use different implementations for each one.

```
@Entity
@Table(name = "employees")
public class Employee {

    private long id;
    private String firstName;
    private String lastName;
    private String emailId;
```

Figure 16: A code snippet of valid Hibernate code, but it is not analyzable for our current implementation because we also require column annotations

```
@Column(name = "first_name", nullable = false)
public String getFirstName() {
    return firstName;
}
```

Figure 17: A code snippet of valid Hibernate code, but it is not analyzable for our current implementation because our column annotations lack arguments

- **Lack of annotations/Usage of JavaBean, DTO:** There is also a different possible approach to using JPA annotations to aid in data persistence within Java. This approach utilizes Java beans, which are used as a way to encapsulate data and provide getter and setter methods for accessing and modifying that data. Similarly to the JPA annotated classes, JavaBeans themselves do not directly handle data persistence and are often used to represent data entities that need to be persisted in a storage medium like a database. These can ultimately be mapped to database tables using ORM frameworks such as Hibernate.

Nonetheless, there may be a way to indeed analyze JavaBeans by filtering on specific class names such as "JavaBean", and tracking its data members. Similarly, DTOs face the same

spring-mvc-showcase-master/src/main/java/org/springframework/samples/mvc/views/JavaBean.java

```
● 1    package org . springframework . samples . mvc . views ;

      3    import javax . validation . constraints . NotNull ;

      5    public class JavaBean {

      7        @NotNull
      8        private String foo ;

      10       @NotNull
● 11       private String fruit ;

      13       public String getFoo ( ) {
      14           return foo ;
      15       }
```

Figure 18: A code snippet of valid Java code, but it is not analyzable for our current implementation because we require the usage of the Hibernate format

kind of problem where there is a lack of relevant annotations, even though they may be used within REST endpoints to encapsulate the input data.

```
@Column(name = "first_name", nullable = false)
public String getFirstName() {
    ...
    return firstName;
}
```

Figure 19: A code snippet of valid Spring code, but it is not analyzable for our current implementation because it lacks annotations for the method arguments

- **Multiple return values:** A REST endpoint may have multiple possible branches within the method to return a value. Each of these separate branches should likewise be counted as separate transactions. This procedure doesn't really work within the development framework, and may actually be excessive function point counting. An alternative to this option would be to rewrite the code so that each branch has its own endpoint. This is a common clean code practice known as "the single exit point law". However, this would be the responsibility of the developers themselves.
- **Incomplete reference resolving 1:** A common way to implement API endpoints within the spring framework is using the ResponseEntity class. This class helps to manipulate return headers and serves as a return value for the REST endpoint. Due to incomplete reference resolving of the BonCode analysis tool, we are not able to access the type name within the class to count the function points of this transaction.
- **Incomplete reference resolving 2:** Similarly to the previous example, due to incomplete

```

@GetMapping("/owners")
public String processFindForm(@RequestParam(defaultValue = "1") int page, Owner owner, B
    Model model) {
    // allow parameterless GET request for /owners to return all records
    if (owner.getLastName() == null) {
        owner.setLastName(""); // empty string signifies broadest possible search
    }

    // find owners by last name
    Page<Owner> ownersResults = findPaginatedForOwnersLastName(page, owner.getLastName());
    if (ownersResults.isEmpty()) {
        // no owners found
        result.rejectValue("lastName", "notFound", "not found");
        return "owners/findOwners";
    }

    if (ownersResults.getTotalElements() == 1) {
        // 1 owner found
        owner = ownersResults.iterator().next();
        return "redirect:/owners/" + owner.getId();
    }

    // multiple owners found
    return addPaginationModel(page, model, ownersResults);
}

```

Figure 20: A code snippet of valid Spring code, but it is not analyzable for our current implementation because our current implementation doesn't support multiple return statements

referencing resolving, we are not able to access the type name within the List class to count the DETs of the record element type "Pet".

9.2 Time Complexity

In Java, there is no direct library for measuring clock cycles, as it is a high-level language and its primary focus is not on low-level hardware operations. We can, however, use the `System.nanoTime()` method to gauge the time complexity of the FPA extension by differing the amount of code analyzed. In all of the experiments, it seems that the time complexity of the tool is linear with regard to its input size. There is, however, a constant component that varies depending on the system that runs the tool.

Varying the size of a file containing the internal logical file: We will start our benchmark by duplicating a piece of code that is determined to be analyzed by the FPA extension, in particular, the ILF section.

```

@PutMapping("/employees/{id}")
public ResponseEntity<Employee> updateEmployee(@PathVariable(value = "id") Long employeeId,
    @Valid @RequestBody Employee employeeDetails) throws ResourceNotFoundException {
    Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id"));

    employee.setEmailId(employeeDetails.getEmailId());
    employee.setLastName(employeeDetails.getLastName());
    employee.setFirstName(employeeDetails.getFirstName());
    final Employee updatedEmployee = employeeRepository.save(employee);
    return ResponseEntity.ok(updatedEmployee);
}

```

Figure 21: A code snippet of valid Spring code, but it is not analyzable for our current implementation because the reference resolving extension fails to locate the source of the class between the angle brackets

```

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
@JoinColumn(name = "owner_id")
@OrderBy("name")
private List<Pet> pets = new ArrayList<>();

```

Figure 22: A code snippet of valid Spring code, but it is not analyzable for our current implementation because the reference resolving extension fails to locate the source of the class between the angle brackets

```

@Column(name = "address")
@NotEmpty
private String address;

@Column(name = "city")
@NotEmpty
private String city;

@Column(name = "telephone")
@NotEmpty
@Digits(fraction = 0, integer = 10)
private String telephone;

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
@JoinColumn(name = "owner_id")
@OrderBy("name")
private List<Pet> pets = new ArrayList<>();

```

Figure 23: The piece of code to be duplicated, and is (without empty lines) 13 lines long.

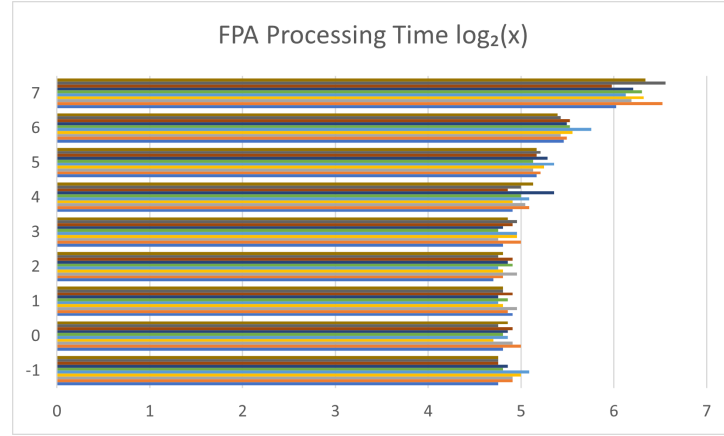
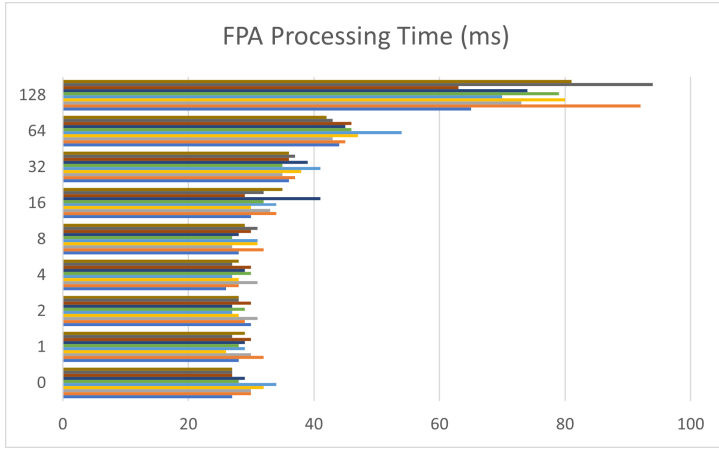


Figure 24: Isolated processing time of the FPA extension of 10 runs, with the amount of code duplication on the y-axis, and processing time on the x-axis. Left: No transformation Right: Log2 transformation

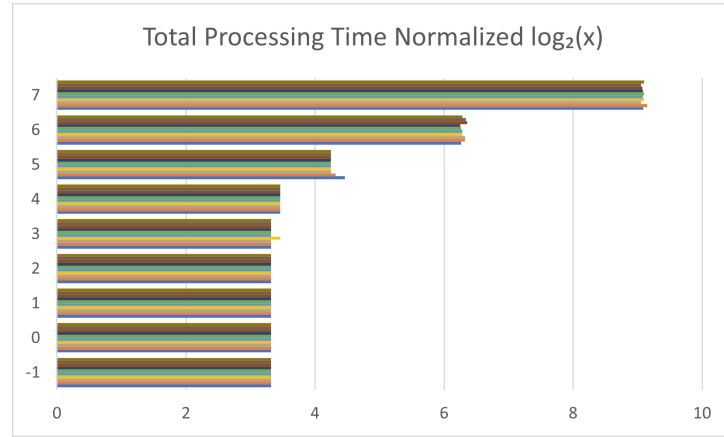
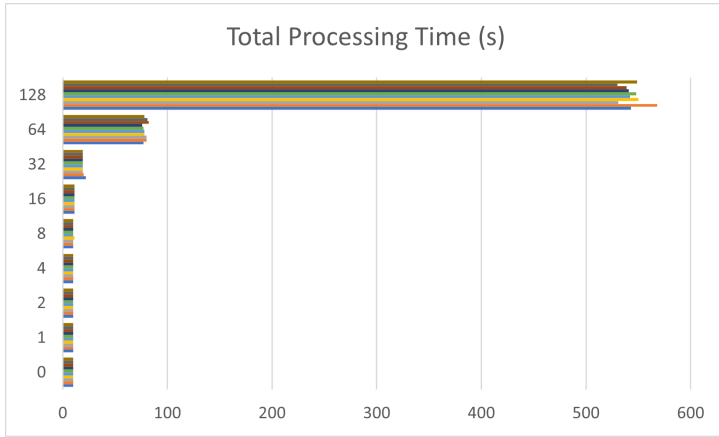


Figure 25: Processing time of the complete analysis tool of 10 runs, with the amount of code duplication on the y-axis, and processing time on the x-axis. Left: No transformation Right: Log2 transformation

Varying the size of a file containing the transaction functions. The next benchmark is similarly about duplicating a piece of code that is determined to be analyzed by the FPA extension, but this time it concerns the transaction functions.

```
@GetMapping("/owners/new")
public String initCreationForm(Map<String, Object> model) {
    Owner owner = new Owner();
    model.put("owner", owner);
    return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
}

@PostMapping("/owners/new")
public String processCreationForm(@Valid Owner owner, BindingResult result) {
    if (result.hasErrors()) {
        return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
    }

    this.owners.save(owner);
    return "redirect:/owners/" + owner.getId();
}
```

Figure 26: The piece of code to be duplicated, and is (without empty lines) 72 lines long.

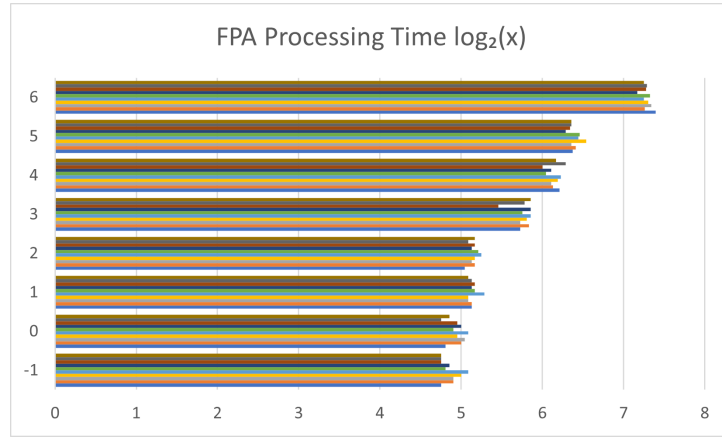
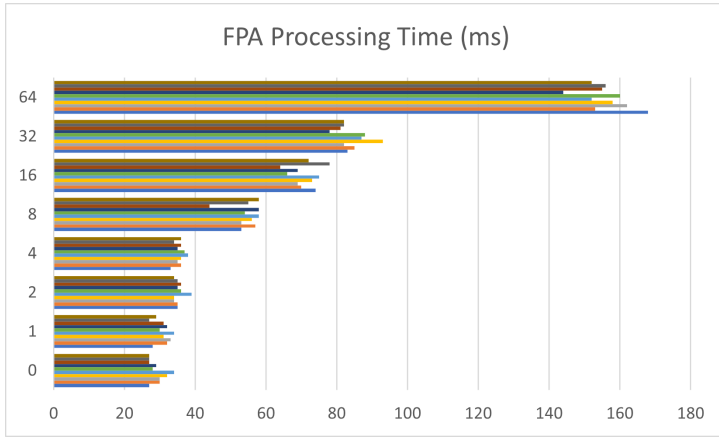


Figure 27: Isolated processing time of the FPA extension of 10 runs, with the amount of code duplication on the y-axis, and processing time on the x-axis. Left: No transformation Right: Log2 transformation

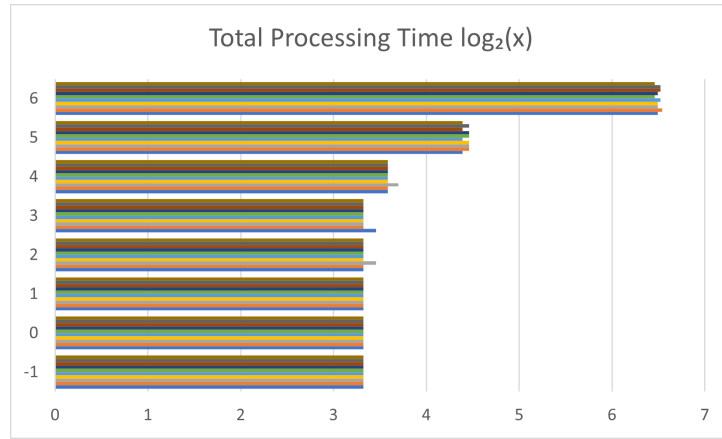
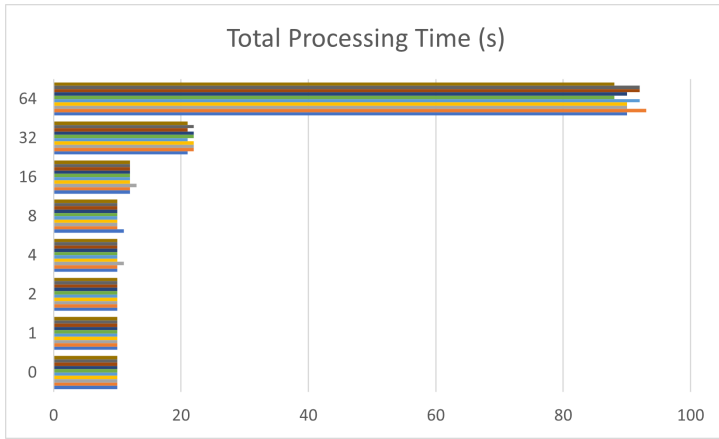


Figure 28: Processing time of the complete analysis tool of 10 runs, with the amount of code duplication on the y-axis, and processing time on the x-axis. Left: No transformation Right: Log2 transformation

10 Interpretation of results

As was shown earlier in the previous section, this tool is sufficient in analyzing FPA-related code structures within the spring framework if the implementation of the tool matches the implementation of the framework within the source code. This may further be improved with the improvement of the reference resolving algorithm of the BonCode tool. Additionally, the accuracy of the analysis would be improved when developers adhere to the single exit point law. This would better reflect the premise that there are multiple branches within the control flow of a method, and therefore that they could be split into multiple transactions. Though, the accuracy would be dependent on the comparison between the results that are generated by the tool, and the results that would be calculated by software estimation experts. Would it be the case that these results differ greatly from each other, then we can conclude that the tool is not very accurate. In this paper, we have only eyeballed the results and are therefore limited in our conclusion.

There may also be multiple different approaches to implement the functionalities within the framework. There are general structures within a framework, but there may also be variances in which ones to use for each purpose. These variances should be kept in mind when developing the analysis tool.

Moreover, the experiments show that the time complexity of the FPA processing tool is not greater than the time complexity of the overall BonCode tool (or less than the greatest time complexity of one of its other extensions). This makes it an overall feasible solution to do automated function point analysis.

10.1 Contributions to knowledge

Function point analysis has been partially applied in the context of modern web development frameworks, specifically those that use an ORM and REST API. development frameworks. This function point analysis tool can feasibly automate the task of manual function point counting ensuring a degree of standardization. Function points are generally counted over multiple points in a period of time. This may especially save a great amount of time and effort in the function point analysis of large software repositories,

10.2 Limitations of the study

The main limitations of this tool lie in the interpretation of function point analysis within web development frameworks, and the functionality of the BonCode development framework.

Function point analysis was mainly developed from a functional requirements perspective, and doesn't constrain in what way the code must be written. This means that there are endless possibilities in how code can be written, and also endless possibilities in how it may be analyzed. Additionally, not all concepts used within FPA may be applicable to a given framework. There may therefore be slight differences in interpretations of where and how to assign these function points to the source code.

Analyzing the source code in its entirety requires us to be able to find all the relevant relations within the source code. The BonCode development framework is quite extensive but is not perfect. There are some issues with reference resolving. This may happen in the case of incomplete reference resolving that has not been implemented as of yet like where we are not able to find the corresponding source code of a class declaration for a given class instance. This means that we are limited in our ability to analyze the contents of the source code, and consequently our ability to assign a correct amount of function points.

As has been mentioned earlier in this thesis, function point analysis on framework code is very implementation dependent. There are inevitably multiple ways to write the same piece of code within a framework, so you will have to take into account all edge cases but might not be able to do so.

Also, because we are using a development framework, some components will inevitably be intertwined with one another. The code analysis tools of BonCode all share a common grammar. It is sometimes necessary to change this grammar to make it easier to develop the function point analysis tool, but it may also break the analysis tools which are currently dependent on it. It is therefore not feasible to alter this grammar too much unless you are able to update the other now-broken tools appropriately, but this lies outside the scope of this thesis.

And finally, there is a limited number of publicly available software repositories that we are able to find that utilize the relevant frameworks. The software repositories that we are able to find are often very limited in functionality and size, and may not accurately represent real-world applications.

10.3 Conclusion

We have shown in this thesis that it is indeed possible to analyze the code structure of source code that utilizes ORM and REST API design patterns and apply function point analysis to it. It is, however, very implementation specific and varies greatly depending on which type of framework the software use.

10.4 Final remarks and suggestions for future work

Future frameworks could be designed to keep in mind function point analysis principles. This would enable software measurement companies to more easily implement software measurements including function point analysis.

The current function point analysis tool may be refined be more abstract and generalizable. The current version allows for multiple languages and their various respective frameworks to perform a rudimentary function point analysis (depending on the completeness of the implementation). It currently doesn't account for the complexity adjustment factor, so future improvements to this tool should take this into account.

It currently allows for the function point analysis of software that utilizes ORM and REST API design patterns, but it may be expanded upon to allow for the analysis of software that uses other design patterns like Simple Object Access Protocol(SOAP) or data mapper.

11 Appendices

- Source code for the example git repository on which the implementation is based in section 8:
<https://github.com/spring-projects/spring-petclinic>

Raw data of the graphs shown in section 9.2

fpa 1	0	1	2	4	8	16	32	64	128
	27	28	30	26	28	30	36	44	65
	30	32	29	28	32	34	37	45	92
	30	30	31	31	27	33	35	43	73
	32	26	28	28	31	30	38	47	80
	34	29	27	27	31	34	41	54	70
	28	28	29	30	27	32	35	46	79
	29	29	27	29	28	41	39	45	74
	27	30	30	30	30	29	36	46	63
	27	27	28	27	31	32	37	43	94
	27	29	28	28	29	35	36	42	81
total 1	0	1	2	4	8	16	32	64	128
	10	10	10	10	10	11	22	77	543
	10	10	10	10	10	11	20	80	568
	10	10	10	10	10	11	19	80	531
	10	10	10	10	11	11	19	78	550
	10	10	10	10	10	11	19	78	542
	10	10	10	10	10	11	19	77	548
	10	10	10	10	10	11	19	76	541
	10	10	10	10	10	11	19	82	539
	10	10	10	10	10	11	19	81	530
	10	10	10	10	10	11	19	78	549

fpa 2	0	1	2	4	8	16	32	64
	27	28	35	33	53	74	83	168
	30	32	35	36	57	70	85	153
	30	33	34	35	53	69	82	162
	32	31	34	36	56	73	93	158
	34	34	39	38	58	75	87	152
	28	30	36	37	54	66	88	160
	29	32	35	35	58	69	78	144
	27	31	36	36	44	64	81	155
	27	27	35	34	55	78	82	156
	27	29	34	36	58	72	82	152
total 2	0	1	2	4	8	16	32	64
	10	10	10	10	11	12	21	90
	10	10	10	10	10	12	22	93
	10	10	10	11	10	13	22	90
	10	10	10	10	10	12	22	90
	10	10	10	10	10	12	21	92
	10	10	10	10	10	12	22	88
	10	10	10	10	10	12	22	90
	10	10	10	10	10	12	21	92
	10	10	10	10	10	12	22	92
	10	10	10	10	10	12	21	88

References

- [Alb79] Allan J. Albrecht. Measuring application development productivity. In *Proceedings of IBM Applications Development Symposium*, page 83, Monterey, October 1979.
- [AR96] A. Abran and P.N. Robillard. Function points analysis: an empirical study of its measurement processes. *IEEE Transactions on Software Engineering*, 22(12):895–910, 1996.
- [Bon] BonCode. Outsystems. Accessed: August 21, 2023.
- [FTB06] Piero Fraternali, Massimo Tisi, and Aldo Bongio. Automating function point analysis with model driven development. In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '06*, page 18–es, USA, 2006. IBM Corp.
- [Fun] Function Point Modeler. <http://www.functionpointmodeler.com/fpm-infocenter/index.jsp?topic=%2Fcom.functionpointmodeler.fpm.help%2Fditafiles%2FgettingStarted%2Fgs-04.html>. [Online; accessed 7-August-2023].
- [Fur97] S. Furey. Why we should use function points [software metrics]. *IEEE Software*, 14(2):28–, 1997.
- [GTS⁺16] Sanjali Gupta, Sarthak Tiwari, H. Singh, Ayush Shukla, and H. S. Raghuvanshi. A comparison between various software cost estimation models. *International journal of emerging trends in science and technology*, 03:4771–4776, 2016.
- [HK91] F.J. Heemstra and R.J. Kusters. Function point analysis : evaluation of a software cost estimation model. *European Journal of Information Systems*, 1(4):229–237, 1991.
- [ISO09] Software and systems engineering — Software measurement — IFPUG functional size measurement method 2009. Standard, International Organization for Standardization, Geneva, CH, December 2009.
- [MM93] Jack E. Matson and Joseph M. Mellichamp. An object-oriented tool for function point analysis. *Expert Systems*, 10(1):3–14, 1993.
- [PTG⁺20] Ken Peffers, Tuure Tuunanen, Charles E. Gengler, Matti Rossi, Wendy Hui, Ville Virtanen, and Johanna Bragge. Design science research process: A model for producing and presenting information systems research. *CoRR*, abs/2006.02763, 2020.
- [SC04] R. Sanches and E. D. Candido. Estimating the size of web applications by using a simplified function point method. In *Web Congress, Joint Conference Brazilian Symposium on Multimedia and the Web amp; Latin America*, pages 98–105, Los Alamitos, CA, USA, oct 2004. IEEE Computer Society.
- [sco] ScopeMaster. <https://www.scopemaster.com/>. Accessed: August 21, 2023.
- [Sym88] C.R. Symons. Function point analysis: difficulties and improvements. *IEEE Transactions on Software Engineering*, 14(1):2–11, 1988.

- [Ver02] June Verner. *Function Point Analysis*. John Wiley Sons, Ltd, 2002.
- [Wik23] Wikipedia contributors. Class (computer programming) — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Class_\(computer_programming\)&oldid=1167836020](https://en.wikipedia.org/w/index.php?title=Class_(computer_programming)&oldid=1167836020), 2023. [Online; accessed 6-August-2023].
- [ZN12] Syeda Binish Zahra and Mohsin Nazir. A review of comparison among software estimation techniques. 2012.