Arus Sahakyan

Software change impact analysis with respect to data protection

Master's Thesis in Information Technology

June 10, 2019

University of Jyväskylä

Faculty of Information Technology

Author: Arus Sahakyan

Contact information: arus.sahakyan@gmail.com

Supervisor: Oleksiy Khriyenko, Vagan Terziyan

Title: Software change impact analysis with respect to data protection

Project: Master's Thesis

Study line: Web Intelligence and Service Engineering, Faculty of Information Technology (Department of Mathematical Information Technology

Page count: 37+19

Abstract: Software evolves and becomes more complex faster every year. Changes and alterations occur all the time, and new additions often can be contradictory to their previous implementations or send a ripple effect through the system, creating many disturbances in other areas. Software Change Impact Analysis studies the impact that a change can bring to a system, and in this paper, we will discuss such changes with regard to the EU's General Data Protection Regulation (GDPR).

Keywords: software change, impact analysis, gdpr

Glossary

IA	Software Change Impact Analysis
SDLC	Software Development Life Cycle
GDPR	General Data Protection Regulation of the EU

List of Figures

List of Tables

Contents

1 Introduction

Software evolves and becomes more complex faster every year. Changes and alterations occur all the time, and new additions often can be contradictory to their previous implementations or send a ripple effect through the system, creating many disturbances in other areas.

Over the course of decades, many solutions have been presented for the change preparation process. Studies have been conducted to analyse the impact the change can bring to a product, as well as several suggestions for classifications, have been made by numerous scholars.

Software Change Impact Analysis (IA) is a subject that is still being studied. It continues to evolve along with the advancement of other technological industries and domains but remains largely non-categorised and non-classified.

In this thesis, we will try to explore the subject of IA and the existing approaches that have been proposed to conduct software change impact analysis.

This thesis provides a new tool through which it would be possible for an individual or a company to easily choose software change impact analysis methods that would fit best to their interests of applying changes to their systems.

This thesis also makes an attempt to discuss how the software change impact analysis could help when attempting to make one's systems compliant with the European Union's recent General Data Protection Regulation (GDPR).

GDPR has affected many companies and their systems, and it is thought-provoking to see what those companies have done or could have done to deal with the challenges of becoming GDPR compliant. Particularly if the employment of change impact analysis approaches would have helped companies to reduce the risks of applied changes.

A study will be conducted based on an interview with a particular company. In the study, it will be discussed how the company became GDPR compliant, what the process was like,

and if the application of change impact analysis approaches would have aided to the process.

The thesis plans to:

- Give definitions and historical context of software change impact analysis. Explain why change impact analysis is necessary, as well as provide information on existing classifications and approaches that have been proposed before.
- Provide a new tool through which it would be possible based on various criteria to choose a software change impact analysis method of implementation.
- Give definitions and historical context of the General Data Protection Regulation of the European Union (GDPR)
- Use the introduced tool for the selection of software change impact analysis approach for the utilisation of GDPR onto existing systems

The question that this thesis aims to answer is:

Is it possible to apply software change impact analysis approaches when making companies GDPR compliant?

Motivation and the idea of the subject have come from the experience of the author working as a software developer in the field of IT and by witnessing the challenges that GDPR compliancy had brought.

2 Software Change Impact Analysis (IA)

In this chapter, we will present the software change impact analysis. The chapter is divided into five sections, with each section focusing on various aspects of the IA. The topics the chapter aims to discuss are:

- the theoretical background of software change impact analysis (IA) with its relation to the software engineering discipline and its role within the software development life cycle
- IA definitions and descriptions
- IA existing taxonomies and classifications
- IA review of the existing approaches

2.1 Historical Setting

In software engineering, software change impact analysis is a process within the maintenance phase of the software development life cycle.

2.1.1 Software Engineering

IEEE Standard Glossary of Software Engineering Terminology defines software engineering as:

> The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software

Laplante identifies it as "a systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of software, that is the application of engineering to software." (Laplante 2001) Whereas Sommerville quite nicely concludes it as "an engineering discipline that is concerned with all aspects of software production". (Sommerville 2016)

History of software engineering dates back to 1960s. From the 1950s and 1960s, due to

the lack of systematic software development processes, it became increasingly difficult to develop large and complex systems, since the individual approaches of program development were not sufficient enough. Projects were running over time and budget, the software was inefficient, difficult to maintain, of low quality, and there were even cases of software causing loss of life. (Leveson and Turner 1993) This period later became labelled as the "software crisis" and in 1968 NATO organised a conference to discuss said "software crisis" and the nature and future of software engineering. NATO conference on Software Engineering is also from where the term "software engineering" has initially originated.

Later, in the 1970s and 1980s, a variety of software engineering techniques were introduced and developed, and tools and standard notations were developed which became the basis of modern software engineering. During this time, it was also identified that the programming is not the only thing that there is in software engineering and other issues like architecture, building and evolution are also important. (Estublier 2000)

2.1.2 Software Development Life Cycle

Software development life cycle (SDLC) is the term that defines the process of developing and maintaining a software application. Goals of SDLC are to produce high-quality software, provide strong management controls and maximise the productivity of the software.

SDLC acronym can be used to either describe software or systems development life cycles. The concept between these two is the same. However, one refers to the lifecycle of software, while the other refers to a system that encompasses a software (Ruparelia and B. 2010). Sometimes SDLC is also called Software Development Process.

Because software creation became more complex, more structure was needed for the development effort and to form a basis for project management and support, and when looking back on the main trends of SDLC, history can be divided into different sections.

In the early years, the engineers mainly tried to understand what needs to be done to make the development process more structured, so that to avoid future problems. The 1970s and 1980s saw the "improved understanding of basic development steps", which led to new development methods and more structured and controlled development processes. However, counter-

movement grew later in the 1990s and 2000s, which focused more on self-organisation, which in itself led to the rise of agile methodologies. Nowadays, "scaling agile" is more fashionable, which is the understanding that both cultures of previous implementations have their positive and negative sides. (Kneuper 2017)

SDLC consists of models and methodologies that are being used to develop the software. Great many of such frameworks have evolved over the years, each having their recognised strengths and weaknesses, advantages and disadvantages. (Maheshwari and Dinesh 2012) These models describe SDLC phases and the order by which the phases are executed.

SDLC phases help the SDLC to achieve its goals. The phases can be defined and redefined in various ways, but to give a basic understanding, we present a list of the more significant ones.

1. Initiation and planning: This is when the project ideas emerge, the concept of the system is developed, and the requirements are defined and analysed.
2. Design: Software design is prepared from the requirements specification.
3. Development: Actual coding and implementation are done based on software design.
4. Testing: After the development is done, the system is tested again against the requirements.
5. Deployment: Once the system is tested, it is ready to be delivered to the customer for their use.
6. Maintenance: After the system delivery, the customer might want to alter the system for various reasons.

SDLC helps us to keep the system under control and thus contributes to satisfying quality and delaying constraints.

2.1.3 Software Evolution and Maintenance

Software maintenance is the discipline that is concerned with changes related to the software system after the delivery is done. (Grubb and Takang 2003)

According to the definition by IEEE93 standard 1219 software maintenance is the

> "...modification of a software product after the delivery to correct faults, to improve the performance or other attributes or to adapt the product to a modified environment."

Software maintenance can be the longest process of SDLC. Products evolve, and markets change, forcing the systems to match their competitors and the requirements of the market, and, alternatively, also overlooked software nuances, as well as software environment changes, might force for bug fixes, even after the delivery of a software product.

The term “maintenance” was introduced by Swanson in 1976 when he grouped the maintenance activities into three basic categories: corrective, adaptive and perfective (Swanson 1976), which were later incorporated into the standard software life cycle processes (ISO 14764) when also the fourth category was introduced, the preventive one. This classification of software maintenance is known as intention-based one. Other two classifications of software maintenance activities introduced by Kitchenham et al (Kitchenham et al. 1999) and Chapin et al (Chapin et al. 2001) are known as activity-based and evidence-based respectively.

To describe the growth characteristics of software, in 1965 Mark Halpern introduced the term and the concept of software evolution, which was later widely used by researchers to describe software change. (Naik and Tripathy 2014) In late 1970s Belady and Lehman published a set of principles that determined the evolution of a software system, known as Lehman’s laws of evolution. (Lehman 1980)

Bennet and Rajlich discussed that the term software evolution lacks a standard definition and that researchers use software evolution as well as software maintenance terms interchangeably. They propose that there are key semantic differences between the two terms:

- maintenance: means preventing the system from failing, fixing bugs
- evolution: means a continuous change from a simpler or worse state of a system to a better state

At the same time, maintenance and evolution have also been distinguished by Bennet and Xu:

- maintenance: all post-delivery activity
- evolution: perfective modifications, triggered by changes in requirements

Since there are not yet agreed differences in the scientific community regarding the difference between maintenance and evolution, in this thesis the term maintenance would be used to describe all of the post-delivery activity, including, bug fixes, perfective modifications as well as other issues raised while the system is still running.

It may seem that maintenance is the continuation of new development, but there are fundamental differences between these two activities. The new development is done from an empty page, while maintenance is done by the parameters and constraints of the existing system. (Grubb and Takang 2003)

Schneidewind (F. Schneidewind 1987) even argued that traditional view of SDLC has done a disservice to maintenance by depicting it solely as a single step at the end of the cycle, which is why the software maintenance should have its own life cycle (SMLC) and a number of SMLC models have been introduced over the years with certain variations. (Chapin 1988) (Sharpley 1977) (Parikh 1986) (Martín and McClure 2019) (Chen et al. 1990) (Yau et al. 1988)

Before undertaking any system development or maintenance work, an impact analysis should be carried out to determine the ramifications of the new or modified system upon the environment into which it is to be introduced. (Grubb and Takang 2003) The impact of introducing a specific feature into a system will be very different if it is done as a maintenance activity, as opposed to development activity. It is the constraints that the existing system imposes on maintenance that give rise to this difference. This difference is explained by Jones:

> "The architect and the builders must take care not to weaken the existing structure when additions are made. Although the costs of the new room usually will be lower than the costs of constructing an entirely new building, the costs per square foot may be much higher because of the need to remove existing walls, reroute plumbing and electrical circuits and take special care to avoid disrupting the current site. (Jones 1986)

Software change is a fundamental ingredient of software maintenance and impact analysis is the key in investigating the prospective change.

2.2 Technical Context

Today's software development environment is highly complex and sophisticated, which makes it harder and more challenging for software products to stay afloat in the modern market. A product has the need to meet the new requirements of its users, shareholders and industry. It constantly has the demand to match with competitors, adjust itself to the system environment changes as well as satisfy the latest and newest security requirements of the industry. To face these challenges and to continue to evolve, the products must undergo constant changes and modifications to their systems, because, for continuous and successful maintenance of software, change is inevitable.

Each system has its own architectural, environmental, software and business requirements and every time a change is proposed, a thorough evaluation and consideration for the rest of the systems needs to be done. If a change is applied "blindly", even smallest one can easily lead to a complete disregard of previous implementations, resulting in loss of time and money, due to the unexpected rework that needs to be performed.(Laplante and Neill 2003) Maintenance is considered to be the most expensive phase in the life-cycle of software development, with more than 50% of the costs coming from applying changes to the software.(Bennett and Rajlich 2000)

Over the course of the past decades, hundreds of approaches have been proposed on the ways one can handle and analyse probable modifications of the systems, and software change impact analysis is something that is required for systems that constantly evolve and continuously become more complex.

Even though challenges of impact analysis can trace back to 1970s, when maintainability of a software system, as well as the creation of systematic models for software development life cycles were being introduced and developed, the term of software change impact analysis has been initially introduced by Arnold and Bohner only in 1996. It defines as

> Identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change (Arnold and Bohner 1996)

The overall goal of IA is to identify entities which are directly or not directly affected by a change (Arnold and Bohner 1993). IA is required for constantly growing systems so as to assist with understanding, implementing and evaluating the changes (Lehnert 2011b)

Currently, there are several hundreds of studies concerned with impact analysis, but only a few proposals have been made by the scholastic community to classify the approaches. The purpose is not to propose any new approaches or classifications, but rather to find an appropriate way of separating approaches that could be more suitable for application of GDPR. For this purpose we are presenting the classifications created and given to the approaches analysed in Lehnert's taxonomy of software change impact analysis. (Lehnert 2011b)

2.3 Classifications

Lehnert argues that a clear taxonomy of software change impact analysis is required due to the "vast amount of approaches, techniques and publications" that have been published over the course of the past several decades. He also argues that ".. a solid comparison and classification of approaches could reduce development costs, since less time must be spent on finding a suitable technique for the current problem." and thus it would be easier to dismiss approaches that would not be of help for the current state of problem. (Lehnert 2011b)

The need for a clear taxonomy exists, since application of an IA approach to existing systems, should be very well understood and assessed beforehand.

In 1993 Arnold introduced a framework to characterize the approaches of software change impact analysis. The framework discusses and introduces three definitions:

- IA Application, describes how the approach is used to execute impact analysis
- IA Parts, describes the functional part of the approach, what it does and how it is done
- IA Effectiveness, examines how effective was the approach for the achievement of the goals

This framework was one of the pioneers of the classification of software change impact analysis, however it seems to lack on the classification of what models or programming languages are used ..

Later, taxonomy of Kilpinen (Kilpinen 2008) separates three groups of impact analysis approaches:

- Traceability IA
- Dependency IA
- Experimental IA

As well as the work of Mens and Buckley describes charactarisation of software change which itself can be applied to impact analysis. (Mens et al. 2003) The characterization included:

- System properties: what is being changed
- Object of change: where it is being changed
- Temporal properties: when a change should be made,
- Change support: how it is changed

However, with the introduction of these frameworks no proper inclusion of a review of the existing approaches is done to justify the applicability of the classification. They lack the proof that this or that taxonomy can be applied to the whole domain and therefore are presented in this section for the sake of the example and not the argument.

2.4 Lehnert's Taxonomy and Review

More detailed and precise proposal of a taxonomy was done by Lehnert in 2011 when the author introduced several approaches by which classification of impact analysis techniques can be done. Furthermore, Lehnert continued his study and later has made comprehensive review to analyse the approaches that were published before. The review is an extensive assessment of approximately 150 studies, where the motivation and methodology behind those approaches is discussed, as well as they are classified under the taxonomy proposed by the writers in their previous publication.

In this paper we are taking into consideration the work done by Lehnert, in consideration of his contribution to the domain.

Firstly, nine requirements are classified in Lehnert's paper to create the possibility to compare IA approaches to identify either the most suitable one or to investigate the difference between said approaches.

- R1: Analysed artefacts
- R2: Provided results
- R3: Supported change types
- R4: Utilized techniques
- R5: Availability of tool support
- R6: Scalability
- R7: Quality of results
- R8: Supported languages / frameworks
- R9: Interactivity of the analysis process

These requirements used to later describe the criterions by which approaches can be grouped. Such grouping is presented in Figure 1.

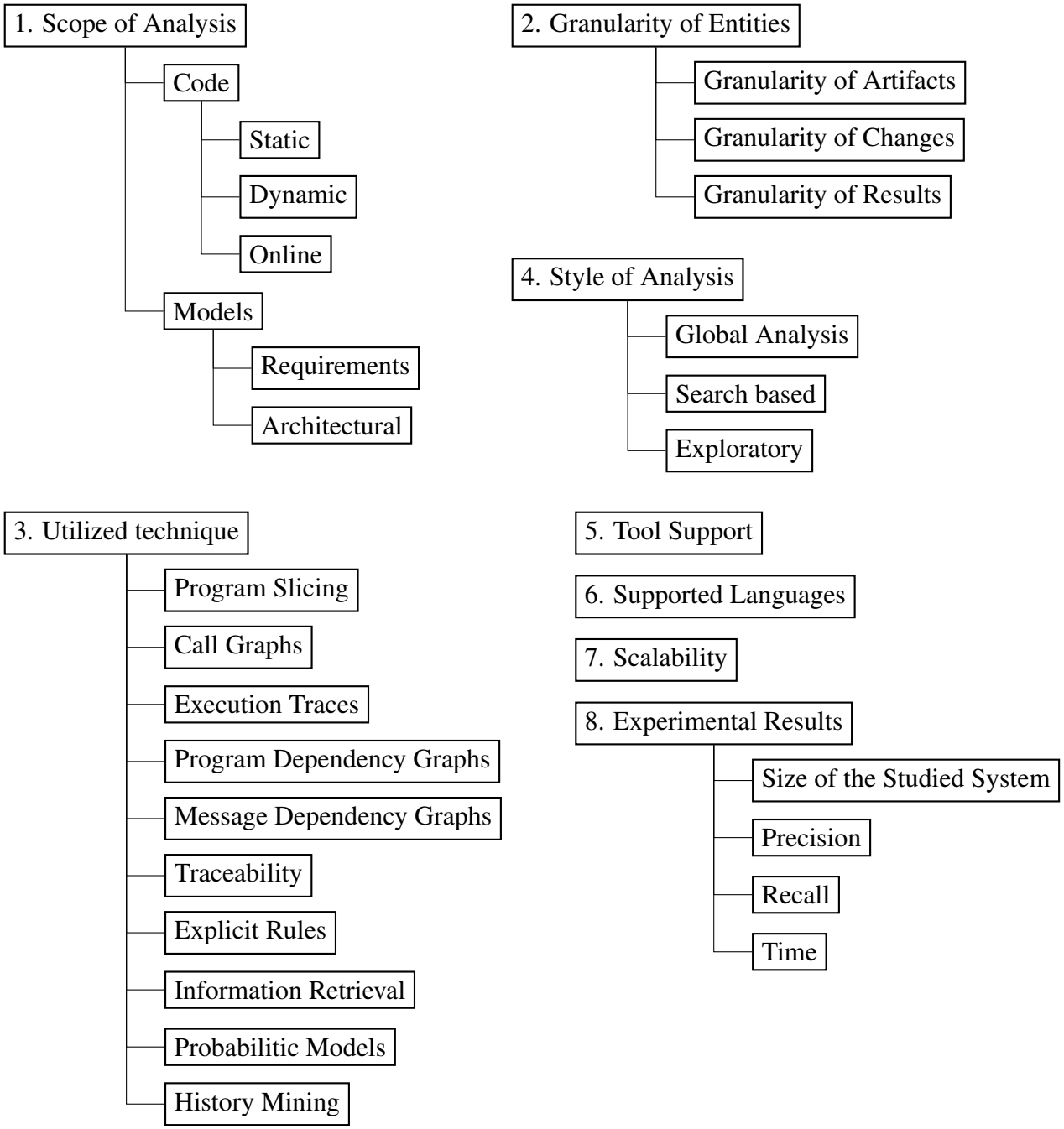

Figure 1: Criterion for Lehnert's taxonomy

- Scope of analysis: discusses what is analysed. Can be either actual source code, or more formal models.
- Granularity of analysis; how fine-grained are entities, changes, and proposed results. Suitable for developers and architects to choose from when they need to know evaluate what the impact of a change would be on variables and/or classes
- Utilized technique: what kind of technique is used with the approach
- Scalability: how scalable the approach is.
- Supported languages: cases when there is a programming or natural language support
- Tool support: cases when there is a tool available.
- Style of analysis - the style of analysis, performed either globally, search-based or exploratory
- Experimental results: if there has been a practical implementation, what the results are

After the publication of the taxonomy Lehnert also published an extensive review of approximately 150 approaches on the software change impact analysis that had been published before. In that paper the approaches were matched with their corresponding criteria and their results and outcomes were discussed. After analyzing the approaches Lehnert's also presented how the discussed studies are classified according to his taxonomy. His findings showed that at least 65% of the proposed IA approaches are concerned with source code analysis, making it the most. It was also concluded that 80% of the requirements approaches as well as 50% of the architectural approaches were largely relying on the theory and were lacking experimental results. (Lehnert 2011a)

2.5 Conclusion

Software Change Impact Analysis has been studied by many scholars over the course of the past few decades but no proper classification proposal had been done before with supporting evidence of the review and evaluation of the existing material. We think that Lehnert's review and taxonomy can be the starting point of having the IA domain more assembled, systematized and coordinated. Because of this we are introducing a new tool that will make it possible for the interested parties to select IA approach, according to the needs and requirements of their project.

3 IA Approach Selection Tool Proposal

In this chapter we will present our proposal for a UI tool that would enable the users and the interested parties to select the IA approach that would be suitable for their needs.

To make the process of the IA approach selection easy and more accessible so as to meet the needs of an individual or a company, we have created a GUI tool that has the all the approaches that have been presented and classified in Lehnert's review in accordance to taxonomy of Lehnert.

The tool gives the option to filter the IA approaches by searching through the scope of analysis, utilization techniques as well as granularity of entities that have been discussed and presented before.

In Lehnert's review of software change impact analysis, the approaches had already been classified according to the criterions in a PDF file table format. With our proposed tool, we are making an attempt to automate the process of the approach selection, so as the process is smoother and easier to maintain.

To retrieve said data from its original format, we have used a third party converter to convert the PDF file of the review into an Excel format. Through that then it was easily possible to retrieve a CSV file and subsequently retrieve JSON format of investigated approaches. JSON data is used to initiate and create database tables and their relations with the help of a programming script.

Example of the data table where Lehnert has classified the approaches, as well as the JSON data and the programming script can be found in Appendix A, Appendix B and Appendix C respectively.

The tool has been written with Python programming language with the assistance of Django framework. The source code of the tool can be found on yousource.it.jyu.fi. (Sahakyan 2019) and the directory structure is presented in Figure 2

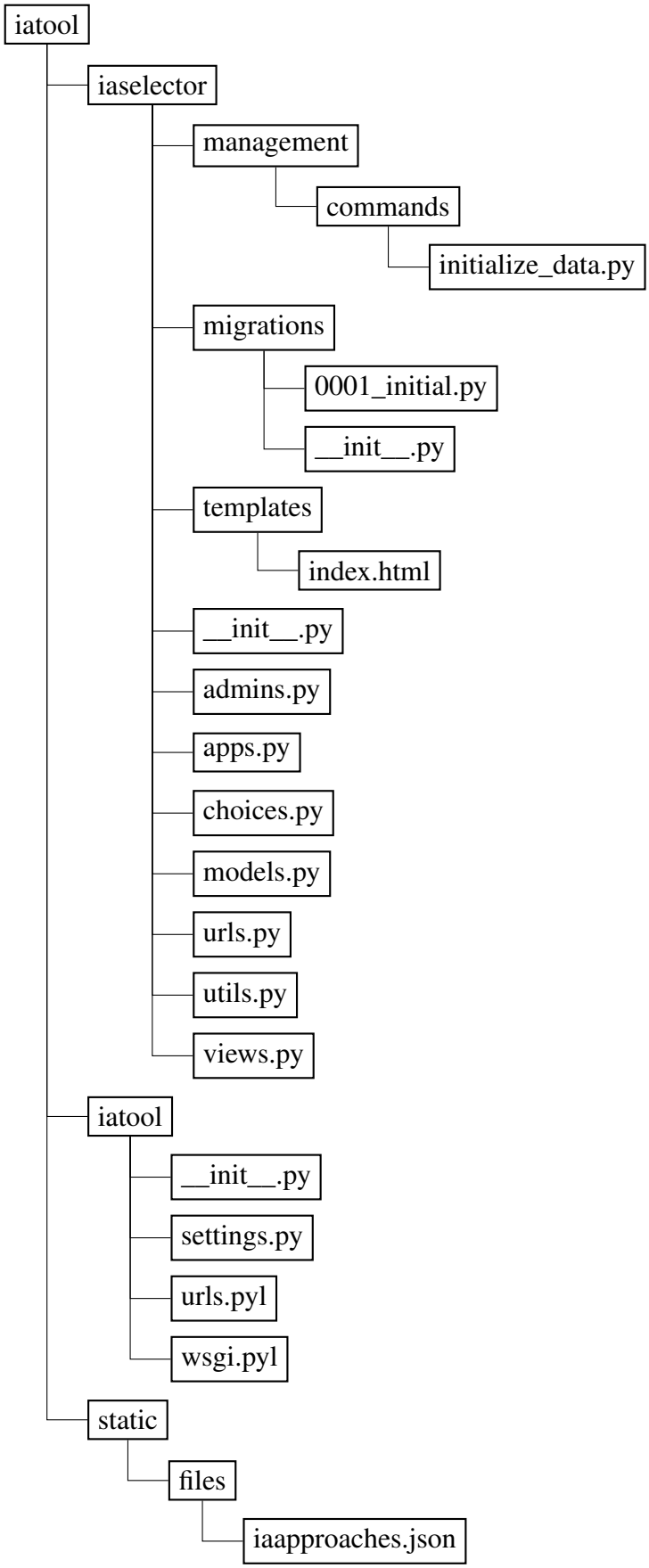

Figure 2: The Directory Structure of IA Tool

To be able to run it on the local machine one needs to have installed latest distributions of Python3, Django2 as well as SQLite. After the installation simply relocate to the main folder and run the commands.

Creates and initialize the database

```
python manage.py migrate
```

A script that inserts json data to the database

```
python manage.py initialize_data
```

Runs the tool on the local machine

```
python manage.py runserver
```

After this, the UI tool will be running on the localhost / 127.0.0.1 of the local machine and will have the representation shown in Figure 3.

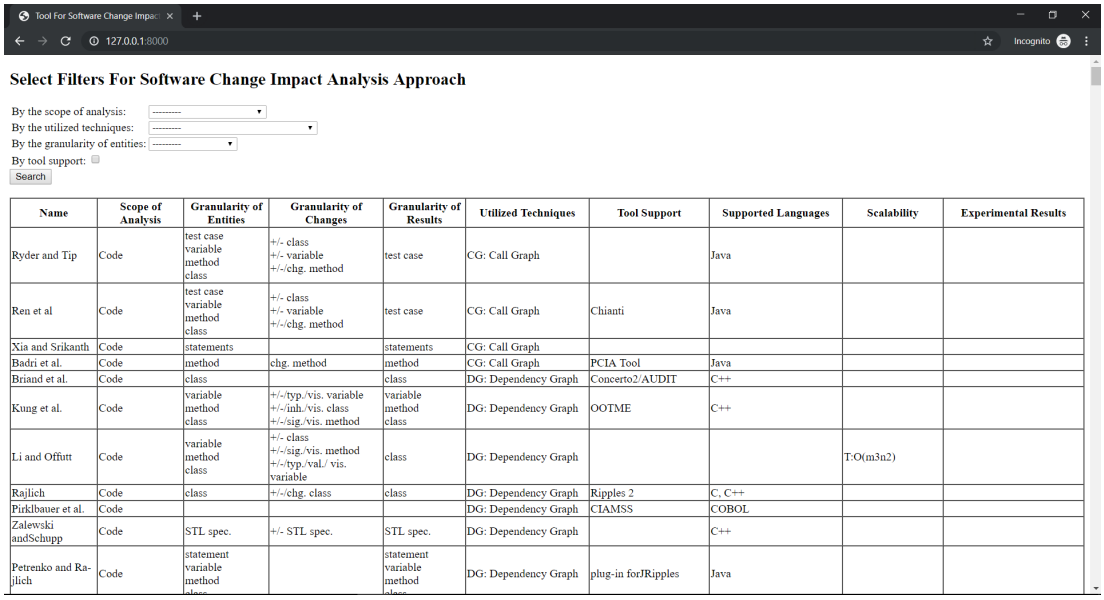

Figure 3: IA Tool. Main page

In the picture we can see that there are four search methods for the search: Scope of Interest, Utilized Technologies, Granularity of Entities and support of a tool. By filter through these options we are able to get the results we need.

4 General Data Protection Regulation (GDPR)

In this chapter we will discuss what European Union's General Data Protection Regulation is. The chapter is divided into two sections, where we briefly discuss:

- the history of data protection and origins of GDPR
- legal and theoretical explanation of what GDPR is and what it does

4.1 Historical setting

The history of data protection in the EU dates back to 1948 when the Universal Declaration of Human Rights was proclaimed and adopted by the General Assembly of the United Nations. Based on that the European Council drafted the European Convention for the Protection of Human Rights, which guarantees the right to the respect for the private life of the citizens of the member states.

Over the course of the following decades, with the increase of technological advances the public and private enterprises were increasingly more able to collect and process personal data of citizens, which soon created a discussion regarding information privacy and created the need for personal data protection. In 1981 the Convention for the Protection of Individuals with regard to Automatic Processing of Personal Data (Council of European Union 1981) was negotiated within European Council.

The Convention 108 obliged the signatories to provide national wide legislation in regard to the processing of personal data and set minimum standards for personal data protection. In the early 1990s, an EU level initiative was taken into action to match and coordinate data protection across the member states and in 1995 the EU adopted European Commissions Data Protection Directive. The prime objective of DIR95 was to regulate the processing of the personal data as well as its flow across the EU borders and thus was so for more than 20 years until GDPR's final approval in March 2016.

4.2 Overview

General Data Protection Regulation is a regulation in European Union law on data protection and privacy for all citizens of the European Union (EU) and the European Economic Area (EEA).

The aim of the legislature is to protect the fundamental rights and freedoms of the residents of the in regard to the protection and processing of the their personal data. It gives the individuals the control over their personal data, as well as harmonizes and stabilizes all data protection laws across all member states with a single set of jurisdiction, establishing the free movement of personal data

The regulation applies to organisations that collect data (data controller) from EU citizens, organisations that process (data processor) said data on behalf of data controller, as well as to individuals (data subject) who reside in the EU.

As an additional case, the jurisdiction of GDPR extended to organisations that are not based in the EU, but that also process personal information of data subjects who are located within the Union. Hence, GDPR is applicable to all organisations that process data of the subjects residing in the EU, regardless of the location of the said organisation.

GDPR discusses various domains of data privacy and processing, but for the purpose of this study we will discuss only those that are directly applicable to the topic of the thesis.

Because of GDPR now data controllers must provide a clearer information on what kind of data they are storing and ask the consent from the data subjects in the clear and simple language. For data subjects it also should be easy to understand what kind of information they are giving and simultaneously should be easy to revoke such access that was given before.

Chapter 3 of GDPR discusses the matters of concerning the rights of the data subjects. With GDPR residents of the EU are granted numerous new rights. Those are including:

Right to access: possibility to obtain the information from data controllers if data subjects personal data is being processed, where and for what purpose. The controller should be able

to provide a copy of said personal data, in readable electronic format

Right to erasure: possibility to be erased (also known as right to be forgotten), which means that they have the authority to request the data controller to erase their personal data from their systems.

Chapter 4 of GDPR dicusses responsibilities of the data processors and data controllers.

Records of processing activities: Data controller and data processor should maintain records on activities that are done with regard of data subject's private data. Said records must include information such as what categories of data are stored, what the purpose of the processing was and said records should be able to be accessible in electronic contract if they are requested by the authorities.

Notification of a data breach: If a data breach happens that "results in the risk for the rights and freedoms" of the data subjects, then data controller must notify the interested parties within 72 hours of the incident. When the data breach happens, the data controller should be able to provide information on what data has been compromised, the approximate number of data subjects that have been affected by the data breach, as well as what the nature of the data breach was. In cases if information is not available immediately, then it should be provided when already accessible without any delays. The controller has also the responsibility to document any data breaches that happen.

4.3 Conclusion

GDPR can be named to be one of the biggest changes and improvements in data protection history. It has affected numerous domains, industries, companies, businesses and individuals. It will reshape the way data protection will be handled and the topics that are covered in this chapter, merely touch the surface on what the regulation is in reality. Since this subject is not directly related to the study of this paper, we will not expand more on it and will continue further by discussing how impact analysis and GDPR are combined together.

5 GDPR and change impact analysis

So far, we have separately discussed and let the reader know what software change impact analysis and what General Data Protection Regulation are. We have informed the reader about the existing research, historical background, as well as intricacies of the law and we have introduced a tool for the selection of IA approaches.

These two subjects may seem not related to an uninterested reader, but in this chapter, we will make the attempt of explaining the connection that can be between GDPR and IA.

In this chapter, we also present findings from an interview done at a certain IT company and regarding GDPR compliancy.

5.1 Overview

General Data protection Regulation was approved and adopted in April 2016 with enforcement deadline of May 25th, 2018. This means that companies around the globe that would had been affected by the new law of European Union had only two years to prepare for the task of being compliant in the modern data protection environment.

European Union had not provided any solution regarding the fact on how the companies should start the process of adhering to the new legislation. But it is safe to assume that such endeavor would have proven pointless. Domain of the legislation is very wide and GDPR affects many industries, therefore no one solution would have been applicable for all of the existing industries.

Making a company compliant with a whole new set of laws is not an easy task. It requires thorough investigation of the processes of the systems that deal with public data, as well as understanding of the law and what exactly the new legislation requires of its subjects. In some cases when proper documentation procedure has been administered before, the investigation process would be more straightforward to execute, but in those cases when no real working routine, processes, documentation or archiving of information is practiced then the task would be much more demanding to undertake.

There are numerous other components that affect how difficult of a task it is to inspect the impact of GDPR-caused change on company's systems. One thing that is most certain is that change is inevitable in this situation. No matter how perfectly designed, implemented and documented a system can be, application of change is a default when attempting GDPR compliancy.

Because of the discussion in the "Software Change Impact Analysis" chapter, we can agree that it is not a good practice to add a new component to a framework without considering the rest of the complex structure of the system. It can cause ripple effects throughout the systems, long term costs as well as degradation of the overall quality of the code. For this reason, we are considering the utilization of software change impact analysis approaches when making companies GDPR compliant, to see if the existing methods are in any way beneficial for the process.

5.2 Interview

In this section, we will present an interview that was conducted with a representative of a certain Finland based IT company, where it is discussed how their systems had changed to become more GDPR compliant. The interview here is presented for the purpose of an example of the subjects.

The process and preparations of making the systems GDPR compliant for the software that is associated with this client had started in 2017. The client had requested information about several domains

-what personal data exists in the systems, -where those are stored, -how these are processed -are there any external interfaces where the data is being exchanged, whether it is incoming or outgoing

All the information regarding the requested topics were needed to be documented in a form of privacy impact assessment. The client had not requested actual actions to be implemented at that stage, rather than the acquired documentation would have been thoroughly analysed and the client would have decided how to proceed. The results were analysed and prioritised

in case there were any systems where critical data is stored.

Third party consultants were hired by the client afterwards to interpret the law and give actions on what to do and a template used by the consultants was given to the data-processor for the course of actions.

It is important to also note that this company is not the data-controller, but is merely the data-processor for the mentioned client.

Software developers, as well as other professionals who had worked on products that were associated with this client, were involved in the process of the prior assessment and impact analysis. After which, several solutions were implemented with regard to the law:

- Possibility to have a fast and easy way of getting data dumps was added to the system (*right to access*)
- A script for the anonymization of the data-subject's data was created (*right to erasure*)
- Audit Logging was added (*records of processing activities*)

GDPR requires that there are no misuses of data and therefore logs are created whenever someone imports or exports data or deletes or performs any action on it. A user id is stored with the time and what the action was.

At the time of the interview, the anonymization script had been requested to be used only once, whereas data dumps have been used more often.

No other formal technique was used for the assessment of the change impact analysis and only the template by the third-party provider was used for this purpose.

On a positive side the company now has everything documented, they know where what data is. It is clear what development and testing environments have. This has eased the process of creating new solutions and also in another positive way a process has been agreed in the case of a data breach. 72h to inform about the breach and whose data has been breached. (*notification of data breach*).

5.3 Choosing approach for GDPR

Currently, there can be numerous ways to tackle the task of handling GDPR compliancy. Each organisation decides on their own how they would want to achieve it, and even though EU has not proposed any solution for the subject, numerous third-party consultancy organisations have taken the responsibility of conducting investigations for the companies in their need to become more compliant with the new legislation. Other options can be, by making "in-house" solutions for impact analysis or manually going through the law with their own legal advisors and deciding for the further course of action. It all depends on the needs and goals of the company.

Machine learning has also been catching up with recent developments in data protection law with a new research project that enables the evaluation of GDPR with the help of AI. (Lippi et al. 2019)

In the previous chapter, we have introduced a new tool that can be used when choosing software change impact analysis approach. The tool is based off on Lehnert's review of existing approaches and taxonomy. We will apply this tool for the selection of a change impact analysis approach that could assist in making a company GDPR compliant.

We will apply this tool for the selection of a change impact analysis approach that could assist in making a company GDPR compliant.

As discussed above, options of becoming GDPR compliant can be many, but for the purpose of this study, we are assuming that a company has a good documentation history and track record of their systems. Because of that, in our IA tool, we are choosing *Requirements* scope from under *Scope of Analysis* drop-down, as well as *IR: Information Retrieval* from under *Utilized Technique* filter.

As a result, we have only two approaches. As we can see from the Figure 4, one of the solutions also supports *Architecture* scope and both have tool support. For the purpose of the example, we are interested only in the study of Jönsson, because he mainly focuses on Informational Retrieval, whereas the study of von Knethan and Gund is more concerned with Traceability.

Select Filters For Software Change Impact Analysis Approach

By the scope of analysis: Requirements
By the utilized techniques: IR: Information Retrieval
By the granularity of entities: ---------
By tool support:
Search

Name	Scope of Analysis	Granularity of Entities	Granularity of Changes	Granularity of Results	Utilized Techniques	Tool Support	Supported Languages	Scalability	Experimental Results
Joansson	Requirements	requirement		requirement	IR: Information Retrieval	SVDLIBC			Size: 400 require-ments Precision: 0.171 Recall: 0.177
von Knethenand Grund	Requirements Architecture	entire UML		entire UML	IR: Information Retrieval TR: Traceability	QuaTrace	UML		

Figure 4: IA Tool. Filtered results

In his PhD thesis, Jönsson discusses the impact analysis from a more organizational and requirements view. His thesis concludes that most of the studied material regarding impact analysis is how it is done from the perspective of software development and he proposes an IA approach by informational retrieval technique. Information Retrieval is the process of searching and recovering information about the system from the network of its structures. It can be either logs, documentation and other areas where information is present. This information retrieval approach of Jönsson uses latent semantic indexing (LSI) and use of term-by-document matrix, through which terms can be associated with a document. (Jönsson 2005)

If we apply this method when making a company GDPR compliant, a company can make a list of terms that are associated with their documentation and through that eliminate the documents that would clearly not be related to the subject. Through this they can work their way in the relevant documentation and create the course of actions for the future, as well as they would not have the need to go through each document one by one and hence would save time in the process.

5.4 Application of Jönsson's approach

In this section we will create a dummy system for to apply Jönsson's strategy of information retrieval with the purpose of application of GDPR compliancy.

For this purpose, we are creating a system that stores information regarding courses, teachers and students. In Figure 5 we present an example of a DB schema of a system. MySQL script for the creation of this database can be found in Appendix D.

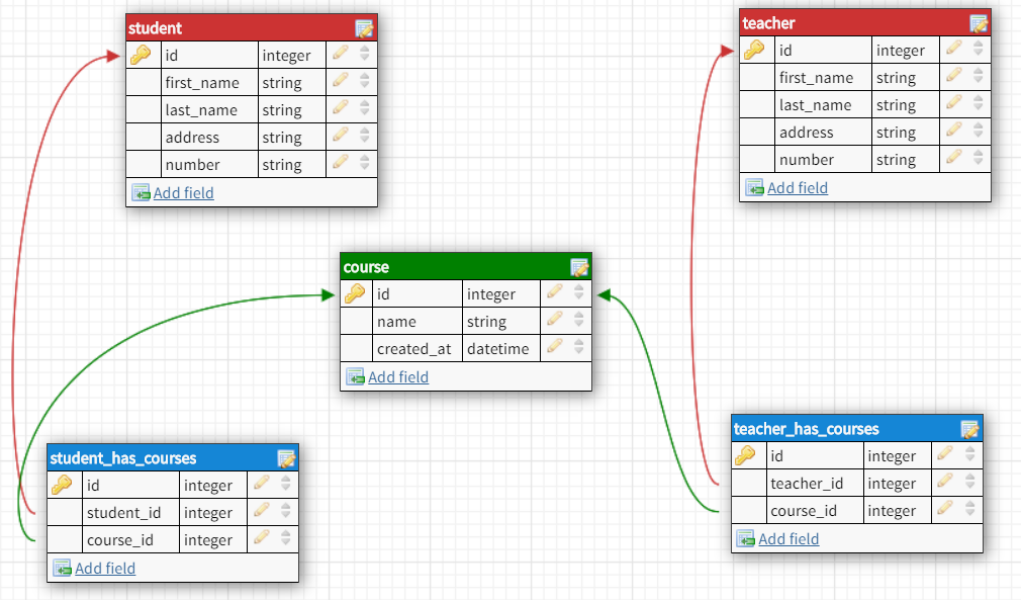

Figure 5: DB example

For the example let us assume that there are multiple documentation papers for the presented schema. In the Table 1 we show which mock-up documentation paper consists of what kind of data.

Table 1: Documentation Example

Document Name	Description
Course	Information on what data of a course is stored (i.e. name and created date), as well as connection with students and teachers.
Student	Information on what personal data is stored about a student
Teacher	Information on what personal data is stored about a teacher

We have already discussed that when making a system GDPR compliant there are several actions that need to be undertaken by the data-processor and data-controller. Currently, we are at the stage where data-processor and data-controller need to find out *if* their systems are GDPR compliant and what actions need to be taken to make sure that the complex meets the new requirements and for this purpose, we chose Jönsson's approach for information

retrieval.

Jönsson establishes four steps for the information retrieval method:

- Step 1: Screen for Relevance - disregard requirements that are strongly not connected to the architecture
- Step 2: Identify Keywords - manually decide the keywords for the look-up
- Step 3: Identify Dependencies Using LSI - identified keywords are added to the term-by-document matrix
- Step 4: Examine Results and Estimate Impact - requirement-component dependencies are manually examined

Since the system in our example is a simple one, and we do not have obvious areas that can be disregarded, we are skipping the first step. Then, we need to manually establish the keywords by which we need to do the lookup search regarding the data that can be associated with GDPR compliancy. Namely, in our case, we have sensitive information regarding natural citizens, and therefore our established keywords are *data* and *personal*. These keywords are placed in a term-by-document matrix's vertical axis, and each document is searched to be matched to contain exact terms. Example results are shown in Table 2.

Table 2: Example results of a term-by-document matrix

Term	Document Name
data	Student, Teacher
personal	Student, Teacher

Based on this example results, we can disregard the documents that are not connected to our list and later by examining the dummy documents we can make decisions and action lists on what kind of changes are needed to be implemented.

Even though we do not have *Course* document in our list, we still need a more narrow search, so that it is easier to establish the strong links across architectures, and if needed the connection to a course can still be found through the documents *Student* and *Teacher*.

A drawback of Jönsson's study is that there is a mention of an automatic tool for creation of the term-by-document matrix, but the tool itself is not presented for the general purpose, therefore the users would have to do the creation of a term-by-document matrix manually. This kind of information is something that should be mentioned when creating the listing of the approaches and filtering through them to find a suitable process.

Nevertheless, a search through a term-by-document matrix, can save more time, because of the dismissal of the non-needed documentation and giving more time in analysing the relevant records of a system, therefore we can conclude that the usage of a software change impact analysis approach can be beneficial for the process of becoming GDPR compliant.

6 Conclusion

The question that this paper aimed to answer was if it is possible to use techniques of software change impact analysis to make companies GDPR compliant. Through the creation of an IA approach selector tool we have been able to filter and find approaches that would be applicable for our example and we have proven that their application is beneficial due to time consumption and therefore also in cost reduction.

One of the main drawbacks of this study is that many companies that had to be compliant with GDPR due to their association with the EU have already taken necessary measures to do so, hence, for the case of GDPR, this study will not be needed anymore with already existing companies.

Although EU continues to expand, therefore there would be new organizations that would need assistance with becoming GDPR compliant, and research studies and new automations for the processes could help to make the transition smoother and easier for all the involved parties.

Contribution of this paper was also an IA approach selector tool as well as digitalisation of the existing approaches under Lehnert's classification. The tool can be used by all the interested parties to also select IA approaches in the cases that are not related to GDPR and are more of technical value. It has been made for non-commercial usage and can be used for further expansion and study of this subject

For the future studies, we suggest expanding on the subject of software change impact analysis, proposing own approach and IA method, as well as considering automation for the IA approach selection process.

Bibliography

Arnold, Robert S., and Shawn A. Bohner. 1993. ”Impact analysis-Towards a framework for comparison”. In *1993 Conference on Software Maintenance,* 292–301. IEEE Comput. Soc. Press. ISBN: 0-8186-4600-4. doi:10.1109/ICSM.1993.366933.

———. 1996. *Software change impact analysis.* 376. IEEE Computer Society Press. ISBN: 0818673842.

Bennett, Keith H., and Václav T. Rajlich. 2000. ”Software Maintenance and Evolution: A Roadmap”. In *Proceedings of the Conference on The Future of Software Engineering,* 73–87. ICSE ’00. Limerick, Ireland: ACM. ISBN: 1-58113-253-0. doi:10.1145/336512.336534.

Chapin, N. 1988. ”Software maintenance life cycle”. In *Proceedings. Conference on Software Maintenance, 1988.* 6–13. doi:10.1109/ICSM.1988.10133.

Chapin, Ned, Joanne E. Hale, Khaled Md. Khan, Juan F. Ramil, and Wui-Gee Tan. 2001. ”Types of software evolution and software maintenance”. *Journal of Software Maintenance and Evolution: Research and Practice* 13 (1): 3–30. doi:10.1002/smr.220.

Chen, S., K. G. Heisler, W. T. Tsai, X. Chen, and E. Leung. 1990. ”A model for assembly program maintenance”. *Journal of Software Maintenance: Research and Practice* 2 (1): 3–32. doi:10.1002/smr.4360020103.

Council of European Union. 1981. *Treat no 108.* https://www.coe.int/en/web/conventions/full-list/-/conventions/treaty/108.

Estublier, Jacky. 2000. ”Software Configuration Management: A Roadmap”. In *Proceedings of the Conference on The Future of Software Engineering,* 279–289. ICSE ’00. Limerick, Ireland: ACM. ISBN: 1-58113-253-0. doi:10.1145/336512.336576.

F. Schneidewind, Norman. 1987. ”The State of Software Maintenance”. *Software Engineering, IEEE Transactions on* SE-13 (): 303–310. doi:10.1109/TSE.1987.233161.

Grubb, Penny, and Armstrong A. Takang. 2003. *Software Maintenance: Concepts And Practice (Second Edition).* Singapore: World Scientific Publishing Co Pte Ltd. ISBN: 9789812384263.

Jones, Capers. 1986. "How not to measure programming quality". 20, number 3 (). https://archive.org/details/computerworld203unse/page/82.

Jönsson, Per. 2005. "Impact Analysis: Organisational Views and Support Techniques". PhD thesis, Blekinge Institute of Technology. https://pdfs.semanticscholar.org/b35e/eba2d78b53d6581bf12058f61a22bf649f3e.pdf.

Kilpinen, Malia Sofia. 2008. *The Emergence of Change at the Systems Engineering and Software Design Interface: An Investigation of Impact Analysis.* University of Cambridge. https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.612389.

Kitchenham, Barbara A., Guilherme H. Travassos, Anneliese von Mayrhauser, Frank Niessink, Norman F. Schneidewind, Janice Singer, Shingo Takada, Risto Vehvilainen, and Hongji Yang. 1999. "Towards an ontology of software maintenance". *Journal of Software Maintenance: Research and Practice* 11 (6): 365–389. doi:10.1002/(SICI)1096-908X(199911/12)11:6<365::AID-SMR200>3.0.CO;2-W.

Kneuper, Ralf. 2017. "Sixty Years of Software Development Life Cycle Models". *IEEE Annals of the History of Computing* 39 (3): 41–54. ISSN: 1058-6180. doi:10.1109/MAHC.2017.3481346.

Laplante, Phillip. 2001. *Dictionary of computer science, engineering, and technology / editor-in-chief, Phillip A. Laplante.* 543 p. CRC Press Boca Raton, FL ; London. ISBN: 0849326915.

Laplante, Phillip A., and Colin J. Neill. 2003. "Requirements Engineering: The State of the Practice". *IEEE Softw.* (Los Alamitos, CA, USA) 20, number 6 (): 40–45. ISSN: 0740-7459. doi:10.1109/MS.2003.1241365.

Lehman, M.M. 1980. "Programs, life cycles, and laws of software evolution". *Proceedings of the IEEE* 68 (9): 1060–1076. ISSN: 0018-9219. doi:10.1109/PROC.1980.11805.

Lehnert, Steffen. 2011a. "A review of software change impact analysis". Technische Universität Ilmenau. urn:nbn:de:gbv:ilm1-2011200618.

Lehnert, Steffen. 2011b. ”A Taxonomy for Software Change Impact Analysis”. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution,* 41–50. IWPSE-EVOL ’11. ACM. ISBN: 978-1-4503-0848-9. doi:10.1145/2024445.2024454.

Leveson, N. G., and C. S. Turner. 1993. ”An Investigation of the Therac-25 Accidents”. *Computer* (Los Alamitos, CA, USA) 26, number 7 (): 18–41. ISSN: 0018-9162. doi:10.1109/MC.1993.274940.

Lippi, Marco, Przemysław Pałka, Giuseppe Contissa, Francesca Lagioia, Hans-Wolfgang Micklitz, Giovanni Sartor, and Paolo Torroni. 2019. ”CLAUDETTE: an automated detector of potentially unfair clauses in online terms of service”. *Artificial Intelligence and Law* 27, number 2 (): 117–139. ISSN: 1572-8382. doi:10.1007/s10506-019-09243-2.

Maheshwari, Shikha, and Jain Ch Dinesh. 2012. *A Comparative Analysis of Different types of Models in Software Development Life Cycle.* Technical report 5.

Martín, James, and Carmen McClure. 2019. ”Software maintenance : the problem and its solutions / James Martín, Carmen Mcclure”. *SERBIULA (sistema Librum 2.0)* ().

Mens, Tom, Jim Buckley, Matthias Zenger, and Awais Rashid. 2003. ”Towards a Taxonomy of Software Evolution” ().

Naik, Kshirasagar, and Priyadarshi Tripathy. 2014. *Software Evolution and Maintenance.* John Wiley / Sons, Incorporated.

Parikh, Girish. 1986. ”Exploring the World of Software Maintenance: What is Software Maintenance?” *SIGSOFT Softw. Eng. Notes* (New York, NY, USA) 11, number 2 (): 49–52. ISSN: 0163-5948. doi:10.1145/382248.382820.

Ruparelia, Nayan B., and Nayan B. 2010. ”Software development lifecycle models”. *ACM SIGSOFT Software Engineering Notes* 35, number 3 (): 8. ISSN: 01635948. doi:10.1145/1764810.1764814.

Sahakyan, Arus. 2019. *Software Change Impact Analysis Approach Selector Tool.* https://yousource.it.jyu.fi/iatool/iatool/.

Sharpley, W.K. 1977. ”Software Maintenance Planning for Embedded Computer Systems”. In *Proceedings of the IEEE COMPSAC,* 520–526.

Sommerville, Ian. 2016. *Software engineering.* 810. Pearson. ISBN: 9781292096148.

Swanson, E. Burton. 1976. ”The Dimensions of Maintenance”. In *Proceedings of the 2Nd International Conference on Software Engineering,* 492–497. ICSE ’76. San Francisco, California, USA: IEEE Computer Society Press.

Yau, S. S., R. A. Nicholl, J. J. -. Tsai, and S. -. Liu. 1988. ”An integrated life-cycle model for software maintenance”. *IEEE Transactions on Software Engineering* 14, number 8 (): 1128–1144. ISSN: 0098-5589. doi:10.1109/32.7624.

Appendices

A Software Change Impact Analysis classified approaches

All studied approaches classified according to the criteria of Lehnert's taxonomy and corresponding review. (Lehnert 2011b) (Lehnert 2011a)

Approach	Scopes	Techniques	Granularity of Entities			Tool Support	Supported Languages	Scalability	Style of Analysis	Experimental Results			
			Entities	Changes	Results					Size	P	R	Time
Ryder and Tip [4]	Code	CG	class, method, variable, test case	+/- class, +/-/chg. method, +/- variable	test case	no	Java	-	-	-	-	-	-
Ren et al. [26]–[28]	Code	CG	class, method, variable, test case	+/- class, +/-/chg. method, +/- variable	test case	Chianti	Java	-	Expl.	123 kLOC, 11k changes	-	1.0	10 min
Xia and Srikanth [30]	Code	CG	statements	-	statements	no	-	-	-	-	-	-	-
Badri et al. [31]	Code	CG	method	chg. method	method	PCIA Tool	Java	-	-	77 classes	-	-	-
Briand et al. [32]	Code	DG	class	-	class	Concerto2/ AUDIT	C++	-	-	40 kLOC	-	-	-
Kung et al. [33]	Code	DG	class, method, variable	+/-/inh./vis. class, +/-/sig./vis. method, +/-/typ./vis. variable	class, method, variable	OOTME	C++	-	-	$>$ 140 classes	-	-	-
Li and Offutt [43]	Code	DG	class, method, variable	+/- class, +/-/sig./vis. method, +/-/typ./val./ vis. variable	class	no	-	T: $O(m^3n^2)$	-	-	-	-	-
Rajlich [34]	Code	DG	class	+/-/chg. class	class	Ripples 2	C, C++	-	Expl.	2 kLOC	-	-	-
Pirklbauer et al. [35]	Code	DG	-	-	-	CIAMSS	COBOL	-	-	-	-	-	-
Zalewski and Schupp [38]	Code	DG	STL spec.	+/- STL spec.	STL spec.	no	C++	-	-	-	-	-	-
Petrenko and Rajlich [39]	Code	DG	class, method, statement, variable	-	class, method, statement, variable	plug-in for JRipples	Java	-	Expl.	550 kLOC	$<$ 0.19	-	-
Black [40]	Code	DG	variable	-	variable	REST	C	-	-	725 LOC	-	-	-
Lee et al. [6]	Code	DG	class, method	-	class	ChAT	-	-	-	30 kLOC	-	-	-
Beszédes et al. [44]	Code	DG	class, method	-	class	-	C++, Java	T: $O(n*e+ n*k*m)$	-	400 classes	0.85	1.0	-
Bilal and Black [42]	Code	DG	class, method	-	class, method	REST, CodeSurfer	C++	-	-	-	-	-	-
Jász et al. [45]	Code	DG	method	-	method	CodeSurfer	C, C++, Ada	T: $O(n+e)$	-	1.4 mLOC, 83k methods	0.87	1.0	3h
Chen and Rajlich [46]	Code	DG	method, variable	-	method, variable	RIPPLES	C	-	Expl.	-	0.09	1.0	-
Gwizdala et al. [47]	Code	DG	class, method, variable	-	class	JTracker	Java	-	Expl.	400 classes	-	-	-
Bishop [48]	Code	DG	class, method, variable	+/- method	class	Incremental Impact Analyzer	Java	-	-	9 KLOC	-	-	350ms

Fasching [36]	Code	DG	-	-	-	CIAMSS	-	-	Gloabl	6k artifacts	-	-	-
Tonella [53]	Code	SL	variable	chg. statement	variable	reachability tool	C	-	-	2 kLOC	-	1,0	-
Korpi and Koskinen [50]	Code	SL	variable	-	variable	GRACE	Visual Basic	-	-	18.7 kLOC	-	-	5 s
Vidács et al. [55]	Code	SL	macro	chg. macro definition	class, method, variable, macro	Columbus C/C++ frontend	C, C++	-	Search-based	-	-	-	-
Binkley and Harman [54]	Code	SL	variable	-	variable	CodeSurfer	C	$O(n^2)$	Search-based	179 kLOC	-	1.0	-
Gallagher and Lyle [51]	Code	SL	variable	+/-/val. variable	variable	-	-	T: $O(n^2 e log e)$	-	-	-	-	-
Hutchins and Gallagher [52]	Code	SL	variable	val. variable	variable	Surgeon's Assistant	C	-	Search-based	-	-	-	-
Korpi and Koskinen [50]	Code	SL	variable	-	variable	GRACE	Visual Basic	-	-	18.7 kLOC	-	-	5 s
Binkley and Harman [54]	Code	SL	variable	-	variable	CodeSurfer	C	$O(n^2)$	Search-based	179 kLOC	-	1.0	-
Santelices and Harrold [56]	Code	SL	statement	chg. statement	statement	DUAForensics	Java	-	Search-based	21 kLOC	-	-	2 h
Apiwattanapong et al. [62]	Code	ET	method	chg. method	method	EAT	Java	T: $O(n)$ S: $O(n)$	-	33 kLOC	0,24	-	-
Breech et al. [60]	Code	ET	method	chg. method	method	DynamoRIO, RVM	Java, C++, C, Fortran	-	Search-based	131 kLOC	-	-	4 h
Law and Rothermel [57]	Code	ET	method	chg. method	method	Codesurfer	C	T: $O(n)$	Search-based	$>$ 6 kLoc	-	-	-
Law and Rothermel [58]	Code	ET	method	chg. method	method	-	C	T: $O(n)$ S: $O(n)$	-	$>$ 6 kLoc	-	-	58 min
Orso et al. [59]	Code	ET	method	chg. method	method	JABA	Java	-	-	60 kLOC	-	-	-
Breech et al. [61]	Code	ET	method	chg. method	method	-	C	T: $O(n^3)$ S: $O(n^2)$	-	40 kLOC	-	-	38 min
Gupta et al. [65]	Code	ET	variable	val. variable	variable	-	-	-	-	-	-	-	-
Gupta et al. [66]	Code	ET	method, statement	+/-/chg. method, +/-/chg. statement	method	-	-	-	-	-	-	-	-
Huang and Song [63]	Code	ET	method	+/- method	method	no	-	-	-	-	-	-	-
Vanciu and Rajlich [67]	Code	ET	method	-	method	Reveal	-	T: $O(s * T + s * n^2)$	-	190 classes, 1.6k methods	0.543	-	5h
Beszédes et al. [64]	Code	ET	method	-	method	JImpact	Java	T: $O(n)$ S: $O(m*n)$	Global	2,3 kLOC	0.35	-	-
Chaumun et al. [69]	Code	ER	class, method, variable	+/-/vis. class, +/-/sig./vis. method, +/-/vis./typ. variable	class, method, variable	-	C++	-	-	$>$ 1k classes	-	-	-

Sun et al. [71]	Code	ER	class, method, variable	+/-/inh./mod./vis. class, +/-/mod./vis. method, +/-/mod./ vis. variable	class, method, variable	JHDG	Java	-	-	157 classes	0.541	0.712	-
Han [68]	Code	ER	module, class, method, statement	+/-/inh. class, +/-/chg. method	class, method	-	C++	-	-	-	-	-	-
Arisholm et al. [70]	Code	ER	class	-	class	JDissect	Java	-	-	17 kLOC, 408 classes	-	-	-
Poshyvanyk et al. [75]	Code	IR	class, method	-	class	IRC^{2}M	C++	-	-	4 mLOC	< 0.28	< 0.66	-
Vaucher et al. [74]	Code	IR	class	+/-/chg. method	class	PTIDEJ	Java	-	-	790 classes	-	-	-
Antoniol et al. [73]	Code	IR	-	-	-	-	C++	-	-	-	0.487	0.696	-
Zhou et al. [76]	Code	PM	class, method, variable	+/-/m. class, +/-/m. method, +/-/m./val. variable	class, method, variable	Evolizer	Java	-	Expl.	-	0.815	0.623	-
Tsantalis et al. [77]	Code	PM	class, method	inh. class, +/- method	class	yes, unnamed	Java	-	Global	169 classes	0.554	-	-
Abdi et al. [78], [79]	Code	PM	class, method, variable	+/-/vis. class, +/-/vis. method, +/-/vis. variable	class	PTIDEJ	Java	-	-	394 classes	0.689	-	-
Abdi et al. [80], [81]	Code	PM	class	-	class	BNJ	-	-	-	-	-	-	-
Mirarab et al. [82]	Code	PM	adaptable	adaptable	adaptable	Smile and other, not named tools	Java	-	-	263 kLOC, $>$ 6k revisions	0.63	0.259	-
Gethers and Poshyvanyk [83]	Code	PM	class	-	class	-	C++, Java	-	-	1.9 mLOC	0.118	0.446	-
Hassan and Holt [95]	Code	HM	class, method, variable	CVS record	class, method, variable	-	C	-	-	$>$ 15k revisions	0.51	0.49	
Ying et al. [94]	Code	HM	source file	CVS record	source file	-	C++, Java	-	Global	$>$ 20k files, $>$ 100k revisions	0.4	0.2	55 min
Kagdi [103]	Code	HM	class, method, statement	change record	class, method, statement	sqminer, srcML, dwdiff, codeDiff	-	-	-	-	-	-	-
Gall et al. [86]	Code	HM	class	CVS record	class	-	Java	-	Global	500 kLOC	-	-	-
Zimmermann et al. [87]	Code	HM	method, variable	+/-/chg. method, +/-/val. variable	method, variable	ROSE	Java, C++, C, Python	-	Expl.	$>$ 34k files, $>$ 53k revisions	0.38	0.416	-
Gîrba et al. [92]	Code	HM	class	+/- method	class	Van, Moose	Smalltalk	-	Global	$>$ 500 revisions	-	-	-
Gîrba et al. [93]	Code	HM	package, class, method	+/-/chg. method, +/-/chg. statement	package, class, method	-	-	-	-	281 kLOC	-	-	-

Bouktif et al. [96]	Code	HM	source file	CVS record	source file	no	Java, C++, C	-	Global	$>$ 9k files	0.772	0.792	3 min
Robbes and Lanza [98]	Code	HM	package, class, method, statement, variable	+/- package, +/-/inh. class, +/- method, +/-/val. variable	-	SpyWare	Smalltalk	-	-	-	-	-	-
Robbes et al. [99]	Code	HM	package, class, method, statement, variable	+/- package, +/-/inh. class, +/- method, +/-/val. variable	-	SpyWare	Smalltalk	-	-	40 classes	-	-	1 min
Robbes and Lanza [100]	Code	HM	package, class, method, statement, variable	+/- package, +/-/inh. class, +/- method, +/-/val. variable	-	SpyWare	-	-	-	-	-	-	-
Fluri et al. [84]	Code	HM	class, method, variable	+/-/chg. class, +/-/chg. method, +/-/chg. variable	class, method, variable	-	Java	T: $O(n^2)$	Global	26 kLOC	-	-	-
Fluri and Gall [85]	Code	HM	class, method, statement, variable	+/-/inh./mod./vis./r. class, +/-/sig./mod./vis./r. method, +/-/chg. statement, +/-/vis./mod./typ./r. variable	class, method, statement, variable	ChangeDistiller	Java	-	-	1.4k classes	-	-	-
Popescu et al. [125]	Code	MDG, SL	component	-	component	Helios	Java, C#, C++	-	-	19 kLOC	-	-	-
Popescu [126]	Code	MDG, SL	component	-	component	Helios	-	-	-	19 kLOC	-	-	-
Kagdi and Maletic [102], [104]	Code	HM, DG	adaptable	change record	adaptable	sqminer, srcML, dwdiff, codeDiff	-	-	-	-	-	-	-
Ceccarelli et al. [108]	Code	HM, PM	source file	CVS record	source file	-	-	-	-	$>$ 10k revisions	0.8	$<$ 0.3	-
Canfora et al. [109]	Code	HM, PM	source file	CVS record	source file	-	Java, C, C++	-	-	$>$ 500 files, $>$ 1.7k revisions	0.31	$<$ 0.6	-
Wong et al. [117]	Code	PM, HM	-	-	-	-	Java	-	-	278 kLOC	0.618	0.355	-
Hattori et al. [14]	Code	DG, HM, PM	class, method, variable	+/-/vis./inh. class, +/-/vis. method, +/-/vis. variable	class, method, variable	Impala	Java	-	-	3.6 kLOC	0.875	0.775	-
German et al. [105]	Code	DG, HM	method	rename, merge, split, clone	method	-	C	-	-	-	-	-	-
Kabaili et al. [110]	Code	DG, ER	class, method, variable	+/-/inh. class, +/- method, +/- variable	class	-	C++	-	-	-	-	-	-
Canfora and Cerulo [106]	Code	HM, IR	source file	-	source file	yes, unamed	-	-	-	$>$ 1.4k files	$<$ 0.36	$<$ 0.67	-
Queille et al. [111]	Code	DG, ER	-	-	-	IAS	C, C++	-	-	2 kLOC	-	-	-

Barros et al. [112]	Code	DG, ER	-	-	-	IAS	-	-	-	-	-	-	-
Huang and Song [113]	Code	ET, DG	method, variable	+/- method, +/-/val. variable	method, variable	no	Java	-	-	-	-	-	-
Huang and Song [114]	Code	DG, ER, ET	class, method, variable	+/-/inh. class, +/- method, +/-/val. variable	class, method, variable	JDIA	Java	-	-	903 kLOC	-	-	103 s
Walker et al. [115]	Code	DG, PM, HM	type	CVS record	type	TRE	Java	T: $O(nlogn)$	Global	-	-	-	-
Maia et al. [116]	Code	DG, ET	class, method, variable	+/- class, +/- method, +/- variable	-	SD-Impala	Java	-	-	6 kLOC	0.274	0.566	-
Kagdi et al. [118]	Code	IR, HM	statement	CVS record	statement	srcML, srcDiff, sqminer	C++, C, Java	-	-	367 kLOC, 2k files	0.852	0.455	-
Sun et al. [119]	Code	SL, DG	package, class, method, statement, variable	+/- package, +/-/r. class, +/-/sig./r. method, +/- /r./t. variable	package, class, method, statement, variable	JHSA	Java	-	-	3.9 kLOC	0.184	-	-
Buckner et al. [182]	Code	-	class	-	class	JRipples	Java	-	Expl.	-	-	-	-
Mohamad [120]	Code	TR, SL	package, class, method	-	package, class, method	CIA-V	C++	-	Expl.	4 KLOC	-	-	-
Kagdi [20]	Code	DG, HM	file, class, method, variable	-	file, class, method, variable	codeDiff, sqminer	-	-	Global	600 KLOC	-	-	-
Lee [124]	Code	DG, ER	class, method, variable	+/-/inh. class, +/-/vis./r./sig. method, +/-/typ./val. variable	class, method, variable	ChaT	C++	-	Global	29 KLOC	-	-	-
Ren [29]	Code	CG, ET	class, method, variable	+/- class, +/-/sig. method, +/- variable	test case	Chianti	Java	-	Global	123 KLOC, 700 classes, 7k methods	-	-	-
Canfora and Cerulo [107]	Code	HM, IR	source file, statement	+/-/chg. statement	source file, statement	Jimpa	Java, C++	-	Global	272 kLOC, 1.5k files	< 0.15	> 0.7	400 s
Hoffman [128]	Code	-	class, method	-	class, method	JFlex	Java	-	Global	-	-	-	-
Moonen [127]	Code	Island Grammar	variable	-	variable	ISCAN	COBOL	-	-	901 kLOC	-	-	26 min
Aryani et al. [129]	Arch.	DG	component	-	component	-	-	-	-	-	-	-	-
Aryani et al. [130]	Arch.	DG, PM	domain var., domain func., UI comp.	-	domain var., domain func., UI comp.	-	-	-	-	104 kLOC	0.614	0.428	-
Briand et al. [131], [132]	Arch.	ER	entire UML	-	entire UML	iACMTool	UML	-	-	-	-	-	-

Dantas et al. [133]	Arch.	TR, HM	entire UML	-	entire UML	Odyssey-SCM	UML	-	-	60 kLOC	0.6	0.15	-
Xing and Stroulia [134], [135]	Arch.	HM	class	+/-/r./m. class	class	JRefleX	UML	-	-	144 classes	-	-	-
Xing and Stroulia [136]	Arch.	HM	package, class, interface, variable	+/- package, +/-/r./m./vis. class, +/-/r./m./vis. interface, +/-/r./m./vis. variable	package, class, interface, variable	JDEvAn	UML	-	-	800 classes	0.955	-	58 min
McNair et al. [137]	Arch.	HM	component, package, class	+/-/chg. component, +/-/chg. package, +/-/chg. class	component, package, class	Motive	Java	-	-	1.5k classes	-	-	-
Yoo and Choi [138]	Arch.	MDG	system	-	system	-	XML	-	-	-	-	-	-
de Boer et al. [139]	Arch.	DG	component, process, data object	+/-/chg. component, +/-/chg. process, +/-/chg. data object	component, process, data object	-	ArchiMate	-	-	-	-	-	-
Vora [140]	Arch.	ER, CG	class, method	-	component	-	TeCFRADL	-	-	10 kLOC	-	-	-
Feng and Maletic [141]	Arch.	ER, SL	component, interface, method	+/- interface, +/- method	component, interface, method	SOCIAT	UML	-	-	-	-	-	-
Tang et al. [142]	Arch.	PM	entire UML	-	entire UML	AREL	UML	-	-	-	-	-	-
Zhao et al. [143]	Arch.	SL	component, connector	-	component, connector	Ciasa	Wright	-	-	-	-	-	-
Wong and Cai [101]	Arch.	PM, HM	class	-	class	-	UML	-	-	14 revisions	0.021	0.061	-
van den Berg [144]	Arch.	TR, DG	adaptable	-	adaptable	no	UML	-	-	-	-	-	-
ten Hove et al. [149]	Req.	ER	requirement	+/- requirement	requirement	plug-in for BluePrint	SysML	-	-	-	-	-	-
Hewitt and Rilling [145]	Req.	DG	scenario, component	-	scenario, component	extended UCMNav2	UCM	-	-				
Lock and Kotonya [148]	Req.	TR, PM	requirement	-	requirement	ARChlVisT	-	-	-	-	-	-	-
Hassine et al. [150]	Req.	SL	entire UCM spec.	-	entire UCM spec.	CIA Tool	UCM	-	-	-	-	-	-
Goknil et al. [151]	Req.	ER	requirement, predicate, relation	+/- requirement, +/- predicate, +/-/typ. relation	requirement	no	-	-	-	-	-	-	-
Lee et al. [152]	Req.	TR	goal, use case	-	goal, use case	-	-	-	-	-	-	-	-
Spijkerman [153]	Req.	TR, ER	requirement, constraint, property, relation	+/- requirement, +/-/chg. property, +/-/val. constraint, +/-/typ. relation	requirement	-	-	-	-	-	-	-	-
Jönsson [154]	Req.	IR	requirement	-	requirement	SVDLIBC	-	-	-	400 requirements	0.171	0.177	-

O'Neal [156]	Req.	TR	requirement, misc. artifacts	-	requirement	-	-	-	-	120 artifacts, 1100 traces	-	-	-
O'Neal and Carver [155]	Req.	TR	requirement	+/chg. requirement	requirement	-	-	-	-	-	-	-	-
Antoniol et al. [157]	Misc. Art.	HM	file	CVS record	file	no	-	-	-	10k files, 3.7 mLOC	-	-	-
Beyer and Noack [158]	Misc. Art.	HM	file	CVS record	file	StatCVS, $cvs2cl^2$, CrocoPat	-	-	-	4 mLOC, 3.9k files		-	-
Askari and Holt [159]	Misc. Art.	PM, HM	file	CVS record	file	no	-	-	-	-	-	-	-
Sherriff and Williams [160]	Misc. Art.	HM	file	change record	file	Matlab	-	T: $O(n^2)$	-	12k files, 240k revisions	-	-	-
Jashki et al. [161]	Misc. Art.	HM	file	CVS record	file	Matlab	-	-	-	3k files, 31k revisions	-	-	-
Nadi et al. [162]	Misc. Art.	HM	-	-	-	DRACA	-	-	-	27k changes	0.885	0.698	-
Hammad et al. [166]	Arch., Code	ER	C++ class, C++ method	+/- C++ class, +/- C++ method	UML class	srcTrace	C++, UML	-	-	550 files, 200 changes	-	-	-
Sharafat and Tahvildari [167]	Arch., Code	PM	class, method, variable	+/- method, +/-/val. variable	class	-	Java, UML	-	-	58 classes	0.702	-	-
Sharafat and Tahvildari [168]	Arch., Code	PM	class, method, variable	-	class	-	Java, UML	-	-	58 classes	0.707	-	-
Kotonya and Hutchinson [170]	Arch., Req.	DG	component, requirement	+/- component, chg. property, chg. constraint	component, requirement	ECO-ADM	CADL	-	-	-	-	-	-
Xiao et al. [171]	Req., Code	CG, ER	BPEL task	+/- task, chg. task-property, chg. task-data	method	-	BPEL	-	-	-	-	-	-
Bohner [37], [172]	Arch., Code	DG	-	-	-	-	-	-	-	-	-	-	-
Bohner and Gracanin [173]	Arch., Code	DG	-	-	-	-	-	-	-	-	-	-	-
Hutchinson et al. [169]	Arch., Req.	TR	component, requirement	-	component, requirement	no	CADL	-	-	-	-	-	-
Khan and Lock [174]	Arch., Req.	TR	component, use case	-	component	-	-	-	-	-	-	-	-
Yu et al. [175]	Arch., Req.	CG	component, requirement	-	component	-	-	-	-	-	-	-	-
Briand et al. [176]	Arch., Req.	TR	class, method, sequence, use case, variable, message, test case	+/- use case, +/-/chg. message, +/-/sig./chg. method, +/-/vis./typ. variable	test case	RTSTool	UML	-	-	320k test cases	-	-	-

Ibrahim et al. [121]–[123]	Arch., Req., Code	TR	class, method, requirement, test case	chg. method	class, method, requirement, test case	Catia	C++, UML	-	-	4 kLOC	-	-	-
von Knethen and Grund [177]	Arch., Req.	TR, IR	entire UML	-	entire UML	QuaTrace	UML	-	-	-	-	-	-
Kim et al. [163]	Arch., Code	DG	source file, class, method, variable	-	source file, class, method, variable	iCIA	C, C++	-	-	7 mLOC	-	-	3 min
Hassan et al. [180]	Arch., Code	ER	component, interface, connector, port	+/- component, +/- interface, +/- connector, +/- port	component, interface, connector, port	set of Eclipse plug-ins	Ada, Perl, PHP, Java, AADL, XADL 2.0	-	-	-	-	-	-
Looman [179]	Arch., Req.	TR	component, requirement	+/-/chg. req., +/- req. predicate, +/-/chg. component	component, requirement	Alloy	AADL	-	-	-	-	-	-

B JSON data examples of the IA Approaches


```
[
 {
   "approach": "Ryder and Tip ",
   "scopes_of_analysis": "Code",
   "utilized_techniques": "CG",
   "granularity_of_entities": "class,method, variable, test
      case",
   "granularity_of_changes": "+/- class,+/-/chg. method,+/-
      variable",
   "granularity_of_results": "test case",
   "tool_support": false,
   "supported_languages": "Java",
   "scalability": null,
   "style_of_analysis": null,
   "experimental_results_size": null,
   "experimental_results_precision": null,
   "experimental_results_recall": null,
   "experimental_results_time": null
 },
 {
   "approach": "Ren et al.",
   "scopes_of_analysis": "Code",
   "utilized_techniques": "CG",
   "granularity_of_entities": "class,method, variable, test
      case",
   "granularity_of_changes": "+/- class,+/-/chg. method,+/-
      variable",
   "granularity_of_results": "test case",
   "tool_support": "Chianti",
   "supported_languages": "Java",
```

```
    "scalability": null,
    "style_of_analysis": "Expl.",
    "experimental_results_size": "123 kLOC,11k changes",
    "experimental_results_precision": null,
    "experimental_results_recall": "1.0",
    "experimental_results_time": "10 min"
  },
  {
    "approach": "Xia  and  Srikanth",
    "scopes_of_analysis": "Code",
    "utilized_techniques": "CG",
    "granularity_of_entities": "statements",
    "granularity_of_changes": null,
    "granularity_of_results": "statements",
    "tool_support": false,
    "supported_languages": null,
    "scalability": null,
    "style_of_analysis": null,
    "experimental_results_size": null,
    "experimental_results_precision": null,
    "experimental_results_recall": null,
    "experimental_results_time": null
  },
]
```

C Data initialization script

```
import json

from django.core.management.base import BaseCommand,
   CommandError
from django.conf import settings
from iaselector import models


class TechniqueAbbreviations:
    PM = 'Probabilistic Models'
    DG = 'Dependency Graph'
    MDG = 'Message Dependency Graph'
    SL = 'Slicing'
    IR = 'Information Retrieval'
    TR = 'Traceability'
    HM = 'History Mining'
    CG = 'Call Graph'
    ER = 'Explicit Rules'
    ET = 'Execution Trace'


class Command(BaseCommand):

    @staticmethod
    def add_scopes(data):
        for name in data:
            models.ScopeOfAnalysis.objects.get_or_create(name
               =name)

    @staticmethod
```

```
29      def add_techniques(data):
30          for abbreviation in data:
31              try:
32                  name = getattr(TechniqueAbbreviations,
                        abbreviation)
33              except AttributeError:
34                  name = ''
35              models.Technique.objects.get_or_create(
                    abbreviation=abbreviation, name=name)
36
37      @staticmethod
38      def add_granularity_of_entities(data):
39          for name in data:
40              models.GranularityOfEntity.objects.get_or_create(
                    name=name)
41
42      @staticmethod
43      def add_granularity_of_changes(data):
44          for name in data:
45              models.GranularityOfChanges.objects.get_or_create
                    (name=name)
46
47      @staticmethod
48      def add_granularity_of_results(data):
49          for name in data:
50              models.GranularityOfResults.objects.get_or_create
                    (name=name)
51
52      def add_criterions_data(self, data):
53          separated_unique_sets = dict()
```

```
        keys = ['scopes_of_analysis', 'utilized_techniques',
           'granularity_of_entities',
                'granularity_of_changes', '
                   granularity_of_results']

        for key in keys:
            separated_unique_sets[key] = set()
            for data_item in data:
                if data_item[key]:
                    list = [item.strip() for item in
                       data_item[key].split(',')]
                    separated_unique_sets[key].update(list)

        for key, values in separated_unique_sets.items():
            if key == 'scopes_of_analysis':
                self.add_scopes(values)
            if key == 'utilized_techniques':
                self.add_techniques(values)
            if key == 'granularity_of_entities':
                self.add_granularity_of_entities(values)
            if key == 'granularity_of_changes':
                self.add_granularity_of_changes(values)
            if key == 'granularity_of_results':
                self.add_granularity_of_results(values)

    def add_approaches(self, data):
        for data_item in data:
            model_data = {
                'name': data_item['approach'],
                'tool_support': data_item['tool_support'] if
                   data_item['tool_support'] else None,
```

```
81                 'supported_languages': data_item['
                       supported_languages'],
82                 'scalability': data_item['scalability'],
83                 'analysis_style': data_item['
                       style_of_analysis'],
84             }
85             approach = models.IAApproach.objects.create(**
                   model_data)
86
87             if data_item['scopes_of_analysis']:
88                 sliced = [item.strip() for item in data_item[
                       'scopes_of_analysis'].split(',')]
89                 for slice in sliced:
90                     instance = models.ScopeOfAnalysis.objects
                           .get(name=slice)
91                     approach.scope.add(instance)
92
93             if data_item['utilized_techniques']:
94                 sliced = [item.strip() for item in data_item[
                       'utilized_techniques'].split(',')]
95                 for slice in sliced:
96                     instance = models.Technique.objects.get(
                           abbreviation=slice)
97                     approach.technique.add(instance)
98
99             if data_item['granularity_of_entities']:
100                sliced = [item.strip() for item in data_item[
                       'granularity_of_entities'].split(',')]
101                for slice in sliced:
102                    instance = models.GranularityOfEntity.
                           objects.get(name=slice)
```

```
103                     approach.granularity_of_entity.add(
                            instance)
104
105             if data_item['granularity_of_changes']:
106                 sliced = [item.strip() for item in data_item[
                        'granularity_of_changes'].split(',')]
107                 for slice in sliced:
108                     instance = models.GranularityOfChanges.
                            objects.get(name=slice)
109                     approach.granularity_of_change.add(
                            instance)
110
111             if data_item['granularity_of_results']:
112                 sliced = [item.strip() for item in data_item[
                        'granularity_of_results'].split(',')]
113                 for slice in sliced:
114                     instance = models.GranularityOfResults.
                            objects.get(name=slice)
115                     approach.granularity_of_result.add(
                            instance)
116
117             experimental_result_instance = models.
                    ExperimentalResult.objects.create(
118                 size=data_item['experimental_results_size'],
119                 precision=data_item['
                        experimental_results_precision'],
120                 recall=data_item['experimental_results_recall
                        '],
121                 time=data_item['experimental_results_time'],
122             )
123
```

```
124             approach.experimental_result =
                    experimental_result_instance
125
126     def handle(self, *args, **options):
127         path = settings.BASE_DIR + settings.STATIC_URL + '
                files/iaapproaches.json'
128         with open(path, 'r') as f:
129             json_data = json.load(f)
130
131             self.add_criterions_data(json_data)
132             self.add_approaches(json_data)
133
134         f.close()
```

D MySQL for the example database

```
1  CREATE TABLE `student` (
2          `id` INT NOT NULL AUTO_INCREMENT,
3          `first_name` VARCHAR(255),
4          `last_name` VARCHAR(255),
5          `address` VARCHAR(255),
6          `number` VARCHAR(255),
7          PRIMARY KEY (`id`)
8  );
9
10 CREATE TABLE `course` (
11         `id` INT NOT NULL AUTO_INCREMENT,
12         `name` VARCHAR(255),
13         `created_at` DATETIME NOT NULL,
14         PRIMARY KEY (`id`)
15 );
```

```
16
17 CREATE TABLE `student_has_courses` (
18         `id` INT NOT NULL AUTO_INCREMENT,
19         `student_id` INT NOT NULL,
20         `course_id` INT NOT NULL,
21         PRIMARY KEY (`id`)
22 );
23
24 CREATE TABLE `teacher` (
25         `id` INT NOT NULL AUTO_INCREMENT,
26         `first_name` VARCHAR(255),
27         `last_name` VARCHAR(255),
28         `address` VARCHAR(255),
29         `number` VARCHAR(255),
30         PRIMARY KEY (`id`)
31 );
32
33 CREATE TABLE `teacher_has_courses` (
34         `id` INT NOT NULL AUTO_INCREMENT,
35         `teacher_id` INT NOT NULL,
36         `course_id` INT NOT NULL,
37         PRIMARY KEY (`id`)
38 );
39
40 ALTER TABLE `student_has_courses` ADD CONSTRAINT `
     student_has_courses_fk0` FOREIGN KEY (`student_id`)
     REFERENCES `student`(`id`);
41
42 ALTER TABLE `student_has_courses` ADD CONSTRAINT `
     student_has_courses_fk1` FOREIGN KEY (`course_id`)
     REFERENCES `course`(`id`);
```

```
43
44 ALTER TABLE `teacher_has_courses` ADD CONSTRAINT `
     teacher_has_courses_fk0` FOREIGN KEY (`teacher_id`)
     REFERENCES `teacher`(`id`);
45
46 ALTER TABLE `teacher_has_courses` ADD CONSTRAINT `
     teacher_has_courses_fk1` FOREIGN KEY (`course_id`)
     REFERENCES `course`(`id`);
```