



AR-MINE: Tool for Automating AUTOSAR software repository mining

Vaishnavi Soni

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:

Author:

Vaishnavi Soni

E-mail: vaso21@student.bth.se

University advisor:

Dr. Deepika Badampudi

Department of Software Engineering

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Rewriting the approach to automotive software development, AUTOSAR introduces a standardized structure that enhances adaptability, facilitates collaboration among multiple stakeholders, and optimizes the development process. However, the complexity introduced by AUTOSAR’s modular structure also presents challenges, particularly in understanding the evolution of the software over time. As automotive software projects evolve, tracking changes in software repositories based on AUTOSAR becomes crucial for comprehending the impact on various system features and ensuring the overall integrity of the software. In response to these challenges, there is a need to provide a systematic and efficient technique for analyzing the evolution of the software based on AUTOSAR, offering insights into code modifications, and their impact on both software and repository evolution.

Objectives. The primary objective of this thesis is to develop and evaluate AR-MINE, a specialized tool for automating the mining of software repositories based on AUTOSAR. By doing so, the thesis aims to provide a comprehensive solution for understanding the evolution of automotive software built on the AUTOSAR architecture.

Methods. The research employs a design science research methodology, involving the design, development, and evaluation of the AR-MINE tool. The iterative process includes understanding the challenges faced in software repository mining built on AUTOSAR architecture, designing a solution to address these challenges, implementing the tool, and evaluating its effectiveness in a real-world context.

Results. The results of this research showcase the effectiveness and utility of AR-MINE in understanding evolution of software built upon AUTOSAR and how it could potentially benefit various people involved in the software development lifecycle. The feedback survey provided valuable insights into the tool’s usefulness and feasibility.

Conclusions. The findings and insights derived from the development and evaluation of AR-MINE, showcase the challenges and opportunities in understanding the evolution of automotive software built upon the AUTOSAR architecture. By analyzing ARXML files and offering an intuitive interface, the tool could be an effective sidekick in reducing the complexities involved in understanding the software based on AUTOSAR. The tool’s industrial validation confirmed its capabilities and practicality, making the tool a valuable asset. Enhancing AR-MINE by adding more features that are relevant and applicable to a much larger scale and improving its efficiency by incorporating third party plugins in extracting and analyzing data from software repositories, are key steps towards its refinement.

Keywords: AUTOSAR, ARXML, Software repository, Data mining, Tool development, Software integration, Automotive Software

Acknowledgments

I would like to express my heartfelt gratitude and special thanks to my thesis supervisors, Mathews Peter, and Dr. Deepika Badampudi, for their unwavering support and guidance throughout my thesis. Despite their busy schedules, they took the time to guide me and keep me on track.

I'd also like to express my heartfelt appreciation to my Manager, Johan Zackrisson, whose encouragement, suggestions, and advice aided in the development of my ideas. I am also grateful for the opportunity to meet so many wonderful and talented people who guided me through my thesis. The thesis would not be possible without the crucial and gracious cooperation of all Volvo Car employees who participated in interviews, and meetings, and assisted me during the research and development phase.

Finally, I'd like to express our heartfelt love and gratitude to my parents, family, and friends for always being there for me and providing moral support. This thesis would not have been possible without their assistance.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
2 Aim and Objectives	5
2.1 Objectives	5
2.2 Research Questions	6
3 Background	7
3.1 Terminology	7
3.1.1 AUTOSAR	7
3.1.2 Tracking Software Evolution	9
3.1.3 Mining Software Repository	10
3.1.4 Storage Solution	10
3.1.5 Visualization	11
4 Related Work	13
4.0.1 Limitations from Related Work	15
5 Method	17
5.1 Research Method Selection	17
5.2 Research Design - Design Science Methodology	19
5.3 Data Collection	20
5.3.1 Literature Review	21
5.3.2 Interviews	24
5.3.3 Tool Evaluation Questionnaires	25
6 Result and Analysis	27
6.1 RQ1: Problem and Feature Identification	27
6.1.1 Findings	27
6.1.2 Features Identification	29
6.2 RQ2: Automating the Mining of ARXML repo	29
6.2.1 Feature Selecting Criteria	30
6.2.2 Implemented Features	30
6.3 RQ3: AR-MINE Feasibility	40

7	Discussion	49
7.1	RQ1: What are the features and metrics that are required to understand the growth of software built on AUTOSAR?	49
7.2	RQ2: How can the automation of mining and tracking recognized features in the ARXML repository be achieved?	50
7.3	RQ3: How feasible is AR-MINE in assisting the practitioners in tracking the growth of the AUTOSAR based software?	51
8	Threats to Validity	53
8.1	Internal Validity	53
8.2	External Validity	53
8.3	Construct Validity	54
9	Conclusion and Future Work	55
9.1	Conclusion	55
9.2	Future Work	55
	References	57
A	Supplemental Information	61
A	Interview Guide	63

List of Figures

3.1	AUTOSAR Structure [6]	8
3.2	ARXML Generation [34]	9
5.1	Design Science Methodology [30]	17
5.2	Research Design Structure	19
6.1	AR-MINE Overview	33
6.2	Program flow of AR-MINE	34
6.3	Code Snippet 1: Basic Usage	35
6.4	Snippet 1: GUI	35
6.5	Snippet 2: GUI	36
6.6	Database File Snippet 1	36
6.7	Database File Snippet 2	36
6.8	Comparison of the Total data extracted from both the commits	37
6.9	Additions of features in both the commits	38
6.10	Deletions of features in both the commits	39
6.11	Total Changes in both the commits	39
6.12	Q1: AR-MINE Satisfaction	41
6.13	Q2: AR-MINE Ease of Use	42
6.14	Q3: AR-MINE Efficiency	42
6.15	Q4: AR-MINE Ease of understanding	43
6.16	Q5: AR-MINE Effectiveness	43
A.1	GUI	61
A.2	Data Overview	62

List of Tables

5.1	Research Data Sources	21
5.2	Exclusion Criteria for Literature	22
5.3	Inclusion Criteria for Literature	22
5.4	Interview participants	24
6.1	Features identified for automating growth monitoring from Interviews	30
6.2	Comparative Analysis of the extracted Data	37
6.3	User Feedback Questionnaire	40
6.4	User Feedback Likert Scale	41
A.1	Interview Questionnaire	63

In the ever-changing field of software development, repository mining has become a promising technique for drawing insightful conclusions from the enormous archives of shared artifacts, code, and documentation [20]. We discover the many advantages of repository mining as well as how it relates to the ongoing development of software systems as we go across its domains.

A large range of studies into the analysis of artifacts created and preserved during software evolution are included in the term "mining software repositories" (MSR) [20]. These repositories include a variety of sources, such as source code storage version control systems, communication archives, requirements or bug-tracking systems. They offer a unique perspective on the evolutionary software development life-cycle because of the vast information they include. A thorough grasp of the developmental elements that occur throughout the course of a project may be obtained by examining individual versions, modifications, and the information pertaining to these changes.

A range of methodologies within MSR have been discovered by software engineering researchers through empirical and methodical studies. Similar to knowledge discovery and data mining, (MSR) aims to provide fresh insights into the software evolution process [11]. Repository mining provides a comprehensive view of how software projects change over time by recognizing connections and exposing patterns. Through this analysis, the points of modifications, teamwork, and decision-making procedures are highlighted, greatly advancing our knowledge of the different evolutionary traits throughout a software system.

A noteworthy contribution to the field of repository mining and commit comparison is the paper by M. Martinez et al. [26], which addresses the challenges in detecting specific code change patterns within Git repositories. The authors identify a gap in existing open-source tools, noting the absence of a tool that, given a Git repository, can return all instances of a given code change pattern. Their tool, named "Coming," takes a Git repository as input and mines instances of code change patterns present in each commit. By computing fine-grained code changes between consecutive revisions and analyzing these changes, the tool identifies instances of user-specified change patterns, as specified using XML. The evaluation of "Coming" on a set of 28 pairs of revisions from Defects4J software demonstrates its effectiveness in finding instances of change patterns, showcasing its potential in understanding software evolution.

As we get into the details of repository mining, we focus on the automobile industry. The use of software in automobiles has grown dramatically in recent years, and it is gradually becoming a set of criteria for a vehicle manufacturer's capacity to compete. According to Hermans et al. [21] automobile software drives 80% of all future developments, and its validation is as crucial as the hardware. In addition to the expanding amount of software, The complexity of automotive systems has increased from isolated functionality on separate Electronic Control Units (ECUs) to highly interactive distributed systems.

It is difficult for a single human to understand the underlying workings of these large embedded systems due to system dependencies and an ever-increasing volume of code [13]. Furthermore, as the need for ECUs grows, so does the time and money needed for software development.

AUTomotive Open System ARchitecture(AUTOSAR) offers standardized interfaces for all Software Components(SWCs) that are essential for creating automotive applications. [35] The outcome of mapping the target hardware or ECUs and software is an AutosarXML file (ARXML) called "Configuration Description, which contains all system information such as bus-mapping, topology, and container and parameter mapping. [35] The SWC or Software Component Descriptions file contains information on all SWC interfaces, and they can be evaluated and implemented separately based on this description. As a result, software integration becomes less challenging.

The complexity inherent in automotive software, particularly within the framework of AUTOSAR, poses significant hurdles for developers and project managers alike. As revealed in a survey [27], the primary drawback of AUTOSAR, cited by 65% of respondents, is its complexity. Respondents emphasized the tradeoff between complexity and productivity, highlighting that managing large projects with numerous developers and highly interconnected functionality becomes exceedingly challenging within the AUTOSAR framework. In addressing these challenges, respondents emphasized the crucial role of tools in easing automotive software development. As one respondent aptly noted, "expertise is needed but tool environment helps. Tools are a must," echoing the sentiment that AUTOSAR should be more tool-oriented to mitigate its inherent complexities.

It is within this landscape of complexity and the pressing need for effective tools that the significance of the present research becomes evident. The research conducted within the 'Continuous Deployment' team aims to alleviate the complexities associated with the development process for software built using AUTOSAR, through AR-MINE - a mining tool developed to aid stakeholders in getting insights from automotive software built upon AUTOSAR. By extracting essential information from ARXML files, AR-MINE seeks to provide information pertaining to the evolution of software, facilitating better understanding and potentially improve decision-making and project management in this increasingly intricate domain.

The repositories are very frequently updated. Practitioners must maintain track

of numerous elements simultaneously. The number of changes is enormous making it impossible to track the modifications as the project matures and grows. Practitioners frequently rely on meetings and reports to understand the growth of the software project, which can lead to mismanagement due to erroneous or biased data as they have to rely on human opinions rather than factual data. There is very limited research on the evolution of these automotive software.

In the present research, we increase our comprehension of the software based on AUTOSAR and more specifically "Configuration Description", also known as an AUTOSAR XML (ARXML)file. We aim to extract the required information from the ARXML files, and develop a tool(AR-MINE) to assist and deliver the information necessary for tracking the growth of the automotive software.

2.1 Objectives

The aim of our research is to address the limitations in the previous studies (as described in Chapter 4) by developing a tool that monitors and tracks the software development process by mining ARXML files.

The tool (AR-MINE) aims to address the limitations of the existing research in the following ways:

- When building AR-MINE, the input of experienced professionals is taken into consideration together with their features and feedback.
- The tool accounts for features and datapoints from ARXML files which are cardinal in structure.
- Helps in explicitly comparing commits, irrespective of the time they are generated.
- An assessment of the tool based on user feedback in a real industrial setting.

Aim: The research aims to provide a more comprehensive understanding of the evolution of the project, in an attempt to improve the efficiency and effectiveness of the AUTOSAR based software development processes.

The objectives of the research are to:

OBJ1: Gaining a deeper understanding on AUTOSAR and software built on it.

OBJ2: Determine the features that experienced practitioners believe are critical for tracking AUTOSAR-based softwares.

OBJ3: Develop a tool that extracts and stores relevant data from ARXML files, enabling retrieval and comparison of different commits.

OBJ4: Conduct user evaluations in an industrial setting to assess the tool effectively.

Commit Comparison:

AR-MINE uses ARXML files that are created with each software repository commit.

The tool then aims to analyze these generated ARXML files by extracting features from them and compares them to identify the changes that occur across the commits.

2.2 Research Questions

In this thesis, we answer the following research questions formulated based on the aims and objectives mentioned in section 2.1.

RQ1: What are the features and metrics that are required to understand the growth of software built on AUTOSAR?

Justification: Extracting the features that help the developers in understanding the growth of the software. All the features are not necessary for the analysis, hence it is important to determine the features that must be considered from the software repositories (ARXML files).

Method: Interviews are used to answer the above question.

RQ2: How can the automation of mining and tracking the recognized features in the ARXML files be achieved?

Justification: Automating the technique used for mining software repositories can have a substantial influence on the software development process as a whole. Developing a tool to track the identified features will enable a much better experience for the practitioners and the stakeholders involved.

Method: Design Science Methodology (DSM) is used to answer this.

RQ3: How feasible is AR-MINE in assisting the practitioners in tracking the growth of the AUTOSAR based software?

Justification: Evaluating AR-MINE's usefulness and feasibility from the perspective of its users. The questionnaire provided helps to collect the practitioner's views and insights. This RQ assesses AR-MINE's ease of use, reliability of tracking, and evolution of AUTOSAR-based software.

Method: Surveys are used to answer the above question.

In this chapter, the key concepts and terminology related to this research are presented.

3.1 Terminology

This section contains information on the terminology associated with the thesis.

3.1.1 AUTOSAR

ECU (Electronic Control Unit)

In the automotive industry, an Electronic Control Unit (ECU) functions as an embedded system responsible for overseeing and managing various subsystems within a vehicle. By receiving input signals from sensors and monitoring the status of system output parameters, the ECU employs a control algorithm to analyze this data. Subsequently, it issues commands to control devices, ensuring the overall optimal performance of the vehicle [28].

Additionally, the ECU has the capability to enhance performance optimization by storing sensor data for application in adaptive control systems. ECUs are categorized based on their specific roles within the system, each assuming distinct names such as Body Control Module, Transmission Control Unit, and Engine Management Systems [28].

AUTOSAR (AUTomotive Open System ARchitecture)

AUTOSAR, a standard software architecture for ECUs. [1] The layered design of AUTOSAR provides all of the capabilities required for software and hardware independence. It differentiates three major software layers that operate on a Microcontroller: the application layer, the runtime environment (RTE), and the base software (BSW). [1] The RTE interfaces the applications from many automotive fields with the base software.

In addition to architecture and interfaces, AUTOSAR specifies a technique for configuring the whole AUTOSAR stack and improving interoperability across different tool chains. On the one hand, this is vital for collaboration within development projects,

but it is also essential for lowering development expenses. AUTOSAR enables the specification of all aspects required to integrate software components (SWCs) on an ECU and the integration of various ECUs to the overall network connection through several different bus systems. The approach describes the dependencies of activities on work products and is intended to enable AUTOSAR activities, descriptions, and tool usage.

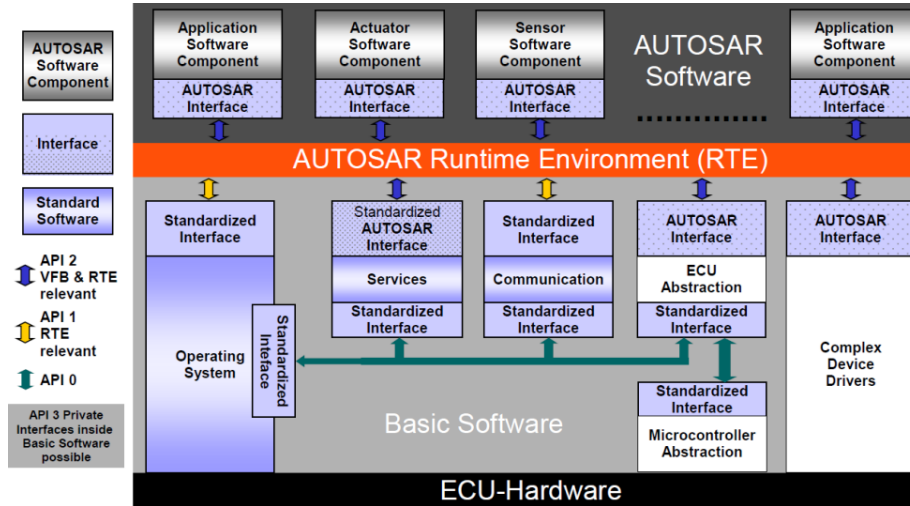


Figure 3.1: AUTOSAR Structure [6]

Standardization of a software platform and development process is required to enable the exchange of models and software across different parties and software integration. The AUTOSAR standard, which regulates a substantial portion of the automotive software development process, is an important effort in this direction. [22] The Application Layer fulfills the functionality of ECU, and it is carried out with the assistance of one or more software components (SWCs).

The next layer, the Runtime Environment, facilitates communication between SWCs, SWC and BSW modules, such as the OS and communication services. The RTE offers the required interfaces for this, and RTE ensures that components can interact and that the system continues to work normally regardless of where the components are placed. The methodology defined by AUTOSAR is the only standardized solution available today, which allows describing the software architecture together with the network topology in a unified machine-readable format [17].

One of the primary benefits derived from this standardization is scalability across different vehicle and platform variants. By compartmentalizing the functionality and establishing standardized interfaces, AUTOSAR enables seamless integration of functional modules from multiple suppliers. Moreover, it ensures the consideration of safety requirements and promotes the development of reusable software components and applications. This holistic approach not only enhances the flexibility of automotive software but also fosters a collaborative ecosystem that can accommodate diverse functionalities, suppliers, and safety considerations, ultimately advancing the state

of automotive technology [33].

The terms "software built on AUTOSAR," "software based on AUTOSAR," "AutoSAR software," and "AutoSAR repositories" are all used in this research to describe the automotive software and repositories that we are evaluating and tracking using AR-MINE.

ARXML (AUTOSAR XML)

It is a standardized XML-based format used in the automotive industry to describe software architectures, communication networks, and configuration parameters of ECUs in vehicles. It provides a structured representation of automotive software systems and their components [3].

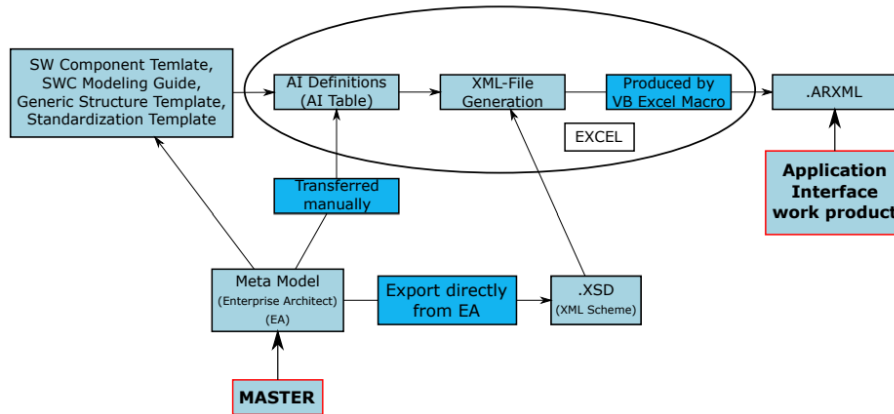


Figure 3.2: ARXML Generation [34]

AUTOSAR offers standardized interfaces for all SWCs that are essential to creating automotive applications. [34]

The outcome of mapping the target hardware ECUs and software components SWCs is a "Configuration Description", also known as an (AUTOSAR XML) ARXML file, which contains all system information such as bus-mapping, topology, and container and parameter mapping. [34] The SWC Descriptions file contains information on all SWC interfaces, and SWCs can be evaluated and implemented separately based on this description. As a result, integration becomes less challenging. Various generators can utilize the information from the descriptions to support the configuration and generation of the RTE and the AUTOSAR basic software (including the operating system).

3.1.2 Tracking Software Evolution

Historic Analysis

Software evolution analysis involves the examination of software modifications, their

underlying reasons, and their consequences. This process utilizes various sources of a software system to conduct a retrospective analysis, including release history with source code and change details, bug report data, and information extracted from the operational system [12].

The primary objective of this field is to evaluate the influence of a change, such as the addition or modification of a feature, on the architecture, design, and implementation of a software system. The ability to gauge the impact of changes enables the estimation of effort required for maintenance and evolution tasks, as well as the assessment of how a change affects the existing architecture and design of a system [12].

3.1.3 Mining Software Repository

- **Software Repositories:** The software repositories are a storage location maintained online or offline by several software development organizations where software packages, source code, bugs, and many other information related to software and their development process are maintained. [32]
- **Mining:** Mining refers to the process of extracting valuable information or patterns from a given dataset. In the context of AR-MINE, mining involves extracting relevant data and insights from ARXML repositories to facilitate software development tasks, such as understanding software architectures (for e.g. memory usage, etc), identifying patterns, detecting issues, and analyzing the growth of the project.
- **Data Extraction:** Data extraction is the process of retrieving specific data or information from a given source. In AR-MINE, data extraction involves fetching ARXML files from the repository to acquire the necessary software-related information for analysis and processing.
- **Parsing:** Parsing is the process of analyzing a string of data or a file to determine its structure and extract relevant information. In AR-MINE, parsing involves analyzing the ARXML files to extract specific data elements such as software design patterns, code snippets, or other relevant information.
- **Data Cleaning:** Data cleaning, also known as data preprocessing, refers to the process of identifying and rectifying errors, inconsistencies, or irrelevant data in a dataset. In the context of AR-MINE, data cleaning involves ensuring the quality and consistency of the extracted ARXML data by removing duplicates, handling missing values, and resolving any inconsistencies.

3.1.4 Storage Solution

- **Database:** A database is a repository that stores data from multiple sources in a centralized location, providing efficient storage and retrieval of information. In AR-MINE, a database, SQLite is used to store the cleaned and processed data extracted from the ARXML files, enabling easier access, retrieval, and analysis.

- **Querying:** Querying the database refers to the process of retrieving specific data or information from the centralized database. To evaluate how the project evolves, it is important to know the changes that take place from one ARXML to another, whenever there is a commit. The AR-MINE involves querying to get the change patterns.

3.1.5 Visualization

- **GUI Dashboard:** A GUI (Graphical User Interface) dashboard is a user interface that provides visual representations and interactive controls to display and analyze data in a graphical format. In the context of the AR-MINE tool, a GUI dashboard is developed using Python's package Streamlit. It allows users to interact with the AR-MINE tool through a web-based interface, providing a user-friendly and intuitive way to access and analyze the ARXML data.

In the current era of software development, there is an abundance of research papers available that focus on the evolution of software, with particular emphasis on commit and repository analysis. But, the ones focusing on evolution of software based on AUTOSAR are very few.

Despite the availability of software tools to aid in the analysis of repository changes for various software artifacts, such as the VCS-Analyzer by Fontana et al. [8] and the change management tool presented by Li et al. [38] most of these tools are not designed for the analysis of the evolution of AUTOSAR based software [14]. There is a need for more specialized tools that can facilitate the analysis of AUTOSAR based software.

In a seminal work by Durisic et al. [14], a systematic technique and a tool named ARCA is introduced, aiming to automate the analysis of software built on AUTOSAR while delving into the impact of meta-model changes on both software and repository evolution. This tool encompasses three primary functionalities: firstly, quantifying and presenting alterations between distinct versions of the AUTOSAR's software meta-model; secondly, presenting the outcomes of various software features characterizing the evolution of the meta-model; and thirdly, quantifying and presenting changes induced by specific features of software with a new AUTOSAR version release.

The ARCA tool analyzes .mod files, also known as modification files, to gather insights into the growth and complexity of the software. It measures the software's size, length, complexity, coupling, and cohesion. In contrast, AR-MINE plans to extract features from description files to identify metrics that are crucial for stakeholders to track.

Durisic et al.'s [14] research examined the detailed changes in "meta-models" across software built on different AUTOSAR release versions. In contrast, our research focuses on specific changes made to the System Description file of the software within the single AUTOSAR version. We carefully examined the evolution of a single software entity throughout a specific release to gain a detailed understanding of its development path. While Durisic et al.'s study covered the broad range of meta-model transformations and their overall effects, our current investigation focuses on the minute details of code modifications.

In a seminal work by J. Falleri [16], an innovative algorithm was introduced for the computation of fine-grained edit scripts utilizing the Git version control system. The algorithm, inclusive of move node actions, was effectively implemented in a freely available and extensible tool. Specifically tailored for parsing JavaScript files, the algorithm was written in Java, and its outputs were stored in XML files. The primary focus of the paper was to unravel the evolution of source code files.

There exists many Mining software repository tools that study the evolution of the software like [25], [10] and [15]. The paper [16] meticulously evaluated the tool's performance, gauging both running time and memory consumption. The results demonstrated that the tool performed reasonably well on authentic data, suggesting its practical applicability in daily use. An empirical evaluation further underscored the efficacy of the algorithm, revealing favorable outcomes. However, it's noteworthy that none of the above studies extend its application to the AUTOSAR domain. The paper by F. Bantelay et al. [9] introduces a novel approach to mine evolutionary couplings by leveraging both interaction histories and commit histories. These couplings are expressed at both file and method levels and are utilized for enhancing commit and interaction predictions. The authors conducted an empirical study on 3272 interactions and 5093 commits from Mylyn, an open-source task management tool. The dataset was divided into training and testing sets to evaluate the performance of combined and individual models. Precision and recall metrics were employed for model evaluation. The findings indicate that the combined models demonstrate statistically significant increases in recall compared to individual models for change predictions. Specifically, at the file level, the combined models achieved a maximum recall improvement of 13% for commit prediction, albeit with a marginal 2% precision drop.

Another paper by Gall et al. [18] states that software typically grows and becomes more complex, inducing more time and effort for performing changes. Software archives such as source code version-control systems and issue-tracking systems (for bugs and change requests) are rich sources to examine what changes have what impact on the software. A software evolution analysis platform called Evolizer analyzes change histories and potential support for evolution. Change types, a core part of the analysis, help discover significant changes and change patterns.

Mario et al. [24] introduce a novel method for extracting frequently recurring code change patterns from code bases, utilizing open Git repositories as a data source. These patterns encapsulate typical changes within a commit, offering substantial information about the relationships between edits. The proposed method aims to enhance auto-completion recommendations, streamline code editors, provide insights into the development process, understand API usage, and automatically tag commits with relevant patterns. The data extraction process involves analyzing changes between commits, transforming each change into a graph representation, and subsequently applying frequent graph mining techniques.

However, the method [24] has certain limitations, such as the input filtering mechanism which is based on size and could lead to the removal of larger files. The paper

acknowledges the tool's reliance on graph modeling, highlighting its reduced degree of freedom.

The paper by M. Martinez et al. [26] introduced "Coming," a tool adept at extracting instances of code change patterns within each commit from a given Git repository. By systematically computing fine-grained code changes. While Coming provides a detailed examination of the evolution of various software projects, it does not focus on the domain of AUTOSAR. Software built on AUTOSAR is typically characterized by its substantial size, increased complexity, and unique attributes that set it apart from other software domains. The present paper builds upon this concept by specializing in the context of AUTOSAR, contributing to the broader field of version control research and enhancing our insights into software evolution practices.

Our study places a particular focus on extracting features that are defined and elaborated in section 7.1.4 from the AUTOSARXML files, providing insights into the evolution of automotive projects. The study sheds light on the potential implications of such changes for software development and maintenance. Overall, our research aims to provide a more comprehensive understanding of the evolution of the project, with the ultimate goal of aiding the stakeholders and practitioners involved in the software development processes.

4.0.1 Limitations from Related Work

There have been limited studies focusing on automatic repository mining and commit analysis related to ARXML files. The majority of the existing studies have been focused on XML extraction.

The limitations and research gaps identified in the existing review are as follows:

- No existing approaches for mining software repositories, including the Coming Tool [26], have been implemented or evaluated specifically for software built on AUTOSAR, particularly concerning ARXML. Our research fills this gap by implementing and evaluating these approaches within the context of AUTOSAR.
- Our research draws inspiration from existing tools such as the Coming tool [26], it is important to note that our work focuses on developing a tool tailored to address certain shortcomings by adding some of the planned features and focusing on the enrichment of the pattern specification to incorporate cardinality of elements (numbers of children and siblings), to showcase the absence of elements and to propose different matching patterns in analyzing various commits.
- In the Coming Tool [26] mentioned earlier, researchers have mainly focused on studying consecutive commits, like commit "r" followed by "r+1" [26]. However, the AR-MINE tool stands out by letting users choose any commit they want from the software repository. This gives users more control, flexibility, and different options for analyzing how the software changes over time.

5.1 Research Method Selection

The following section explains the research methodologies employed in the study.

The **Design Science methodology (DSM)** was chosen as the research technique because it encourages interaction between academic and industrial researchers in the field of software engineering [29]. According to Offermann et al. [30] the three primary stages of the Design Science method are "problem identification," "solution design," and "evaluation," which might interact with one another during the course of the study project. Every stage is broken down into steps. Transitions between steps are indicated by arrows, whereas less often used transitions are indicated by dotted lines. This strategy fits well with the goals of this study, which includes data collection, designing a static analysis tool for repository mining, and assessing the tool's performance, usefulness, and effectiveness.

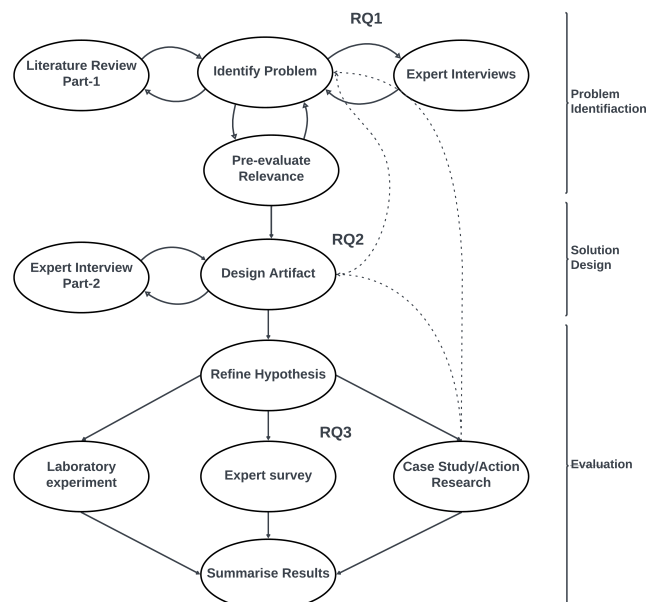


Figure 5.1: Design Science Methodology [30]

In our research, the **Interviews** and **Literature Review** were employed as part of DSM for problem identification and collecting quantitative data from a predefined

population. This approach helped us to address RQ1 by enabling us to gather, identify, and generate a list of possible features that were considered important by the practitioners in tracking the growth of software based on AUTOSAR.

The semi-structured interviews and literature review helped in increasing the understanding of the problem, and overall scope of the solution. This developed list of features will be used as a basis for the information in developing AR-MINE. The process of data collection and analysis will be examined in depth later in this section.

Several prominent methodologies were considered, however they were found to be inappropriate for this study's needs. The purpose was to establish a strategy that encouraged collaborative research between academia and industry. Compared to other techniques, DSM was determined to have the best fit [37].

- **Experiment:** Experimentation is a research approach used to evaluate the relationship between dependent and independent variables in a controlled environment. [31] However, in the context of this research, the aim is to evaluate the study in a real-world setting. Therefore, experimentation may not be the most appropriate research approach for our study, as it may not reflect the complexities and nuances of real-world scenarios. Instead, a design science approach that allows for in-depth analysis and continuous improvement through iterations of the phenomenon in its natural setting is a suitable method for our research.
- **Case Study:** A case study examines a real-life event and draws lessons from several sources. Initially, it was viewed as a promising study tool to solve RQ 2. Case studies rely heavily on the issue context to draw conclusions [31]. However, their lack of attention on creating and evaluating artifacts does not correspond with our study aims. Our objective is to produce a feasible solution through an iterative manner, rather than using a case study technique.

Additionally, we used Survey, which is a part of the DSM approach to evaluate the feasibility and usefulness of the tool, which was instrumental in addressing RQ3. By leveraging the benefits of this empirical investigation method, we were able to gather valuable insights and make informed decisions based on the data obtained.

Each strategy serves different purposes, and in our case, we plan to use the iterative improvement to enhance the MEP5 project development process at Volvo Cars. To support this, the DSM offers various iterative and continuous feedback methods, including questionnaires, interviews, observations, artifacts-oriented searches, and focus groups.

Our focus will be on Interviews to increase our understanding of the problem and data collection which will help us in answering RQ1, the tool building with Design Science Methodology will accommodate the RQ2, and surveys to evaluate the effectiveness of the produced mechanism in improving stakeholders understanding of the

tool and to answer research question RQ3.

5.2 Research Design - Design Science Methodology

This section presents the structure of the research method Represented in figure 5.2.

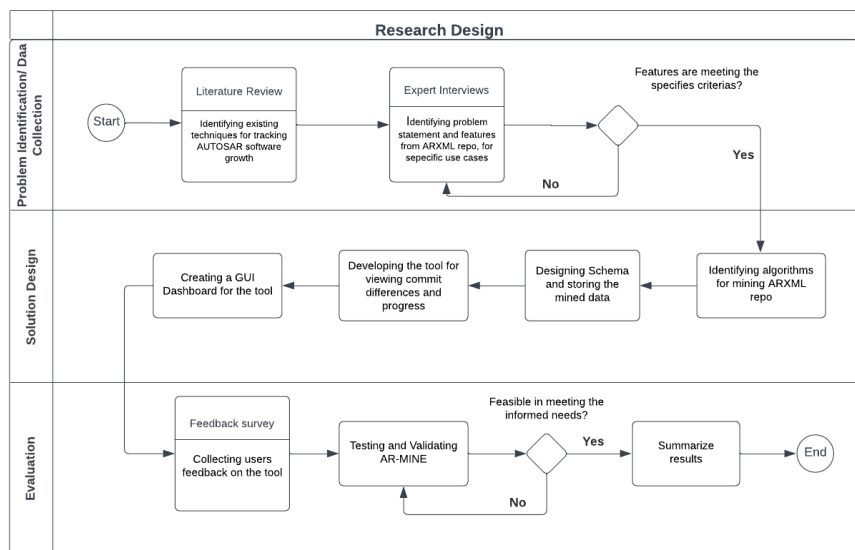


Figure 5.2: Research Design Structure

Design science study uses artifacts to help practitioners solve complex organizational difficulties. The artifact might include software, logical structure, mathematical equations.

Design science emphasizes iterative development and continuous artifact improvement. The problem's perspective and complexity shift with each iteration. Each iteration aims to improve the artifact's features and quality through scaling and problem-solving. The artifact is evaluated using appropriate processes based on the problem. Design science employs several methodologies for evaluating designs.

Automotive companies use AUTOSAR and employ ARXML repositories, which contain a wealth of information and data that must be mined in order to understand and visualize the project's overall progress. The interviewees previously used Git to monitor the project, but it involves a lot of manual work. Our goal is to incorporate software repository mining technique to understand the automotive software.

The solution is evaluated using the observation design evaluation method. So, it will enable us to carefully examine whether or not the solution we created to address the needs of the stakeholders and the research study is successful or not.

The principles of design science by Hevener et al. [23] were put into practice in our research. The recommendations are as follows:

1. **Design an artifact:** Our research utilizes the attributes identified by RQ1 to create a tool that visualizes the commit differences, as an artifact.
2. **Scope the problem:** All automotive businesses using AUTOSAR are in a way affected. Nearly 70% of the world’s automakers are AUTOSAR partners, and a sizeable fraction of them are either using or planning to use AUTOSAR, according to P. Sivakumar et al. [33] In order to manage and analyze the ARXML files that are generated every time a developer commits, practitioners give up valuable development time. The agility of the team is also being hampered by AUTOSAR’s complexity. Our artifact’s goal is to help practitioners by assisting them in tracking the growth of such software.
3. **Evaluating the design:** Focus group meetings and technical demos were utilized to evaluate the resulting artifact. During each evaluation, the focus group tests the artifact using example ARXML files generated in different revisions and based on the performance, offer feedback.
4. **Research contributions:** Our study of developing a tool(AR-MINE) acts as an interesection between academic literature on mining of software repositories and requirements of automotive industry that uses AUTOSAR. AR-MINE motivates further exploration of software repository mining in the automotive domain, encouraging the use of mining technique and user-friendly attributes comprehensible to the stakeholders.
Additionally, we present a solution that not only benefits our specific context but also serves as a model for other automotive organizations seeking to tackle the complexities of AUTOSAR with customized tools.
5. **Research rigor:** We conducted regular discussions and stakeholder interviews to identify the most pertinent and necessary features for our repository, which will help AR-MINE to be much more effective.
6. **Design as a search process:** The literature serves as the foundation for our research. In the process of putting our solution into practice, we also studied how researchers have approached similar kinds of problems in the automotive domain.

Our design science process model, which applies the design science criteria, is depicted in Figure 5.2. To raise the caliber of our solution, the steps taken in phase two were repeated several times.

5.3 Data Collection

The data needed to complete and analyze our thesis work was gathered through semi-structured interviews, a literature review which is a part of the Design Science methodology, and feedback questionnaires. Table 5.1 summarizes the various data sources.

Data Source	Description	Question
Semi-structured interviews	Features responsible for mining the instance of changes patterns.	RQ1
Literature review	Study of previous literature proposed on software mining and commit analysis.	RQ1
Feedback Questionnaire	User evaluation of the tool that helps find out the ease of use, effectiveness of the tool	RQ3

Table 5.1: Research Data Sources

Feature Identification

To identify/mention features, semi-structured interviews were conducted with some of Volvo Cars' experienced developers, architects and a manager. The features are identified that could be automated and were required by the practitioners to streamline the development and integration process.

Feature Selection

The following step is to utilize the features identified to determine which subset of features to take into account when implementing the AR-MINE tool. The features chosen are determined by the feedback provided by the developers in interviews. If the devised set of features are not appropriate for tool implementation, they should be removed or alternatives must be explored.

5.3.1 Literature Review

As a part of the Design Science Method, prior research was conducted that introduced various methodologies and tools for the automated mining, comparison, and analysis of software repositories and commits based on AUTOSAR. A rapid Literature Review [19] was conducted in search of the formal literature on this subject. To retrieve pertinent literature, a set of keywords was meticulously chosen, including "AUTOSAR", "Software Repository", "Mining", "Tool", "Automated", "Analysis", "Software Architecture", "Commit Analysis", "Change" and "Evolution". Different combinations of above mentioned keywords were utilized to identify papers relevant to our research. For the rapid literature review, a thorough investigation was conducted by searching reliable databases such as Scopus, IEEE Xplore, and Google Scholar. Detailed inclusion and exclusion criteria for the studies included in the literature review are presented in Table 5.2 and Table 5.3, respectively.

Description	Motivation
Age of the literature	Excluding publications that were outdated or published before 20 years to ensure the inclusion of the most current and relevant research in the review.
Relevance	We check if the references are useful for our research by looking at their titles, abstract and content.

Table 5.2: Exclusion Criteria for Literature

Description	Rationale
Commit, Repository, Data Analysis and comparison	research involves studying previous literature on automation of commit and data comparison and analysis
Full-text Access	The research's history, context, and supporting data are absent from the condensed version.
Literature in English	Textual inconsistencies in translated literature might lead to a misinterpretation of the research.

Table 5.3: Inclusion Criteria for Literature

The quest for literature pertaining to our study focused on mining ARXML repositories to perceive software evolution and commit analysis within the AUTOSAR domain, revealed that there was a significant lack of research dedicated to the detailed exploration of ARXML mining in the context of comprehending software evolution. This particular research gap was the primary focus of our investigation. Although there have been previous endeavors in commit analysis within various software repositories, the domain of AUTOSAR has remained relatively uncharted. In light of this, we encountered a limited body of literature that directly addressed these specific aspects. However, we did uncover a subset of papers that touched upon commit analysis, repository mining, and software evolution in other contexts.

To bolster the research, inclusion criteria were developed. The first set of criteria revolved around the fundamental aspects of our study, emphasizing the importance of works that dealt with "Commit," "Repository," and "Data Analysis and Comparison." This enabled us to gather literature that closely aligned with the core of the research objectives. Our study was deeply rooted in the automation of commit and data comparison and analysis.

Additionally, the importance of "Full-text Access" was emphasized. It was crucial that the literature we referenced offered a comprehensive view of the research, ensuring that we did not miss any vital context, history, or supporting data that might be absent in abstracts. Furthermore, we insisted on "Literature in English." This ensured that the context and information provided in the literature were closely

aligned with the author's original intent, reducing the risk of misinterpretation.

Finally, our exclusion criteria focused on the "Age of the Literature." To ensure that our review incorporated the most recent and relevant research, we excluded publications that were outdated or had been published more than 20 years ago.

In summary, our literature review was guided by these inclusion and exclusion criteria, which not only served to confine our search but also ensured that the selected literature was relevant, comprehensive, and aligned with the specific needs and context of our research.

Focus Groups

In the context of our research on AR-MINE and its application at Volvo Cars, we recognized the need to harness the valuable insights that could be derived from focus groups. Focus groups, which consist of small groups of individuals sharing a common demographic or expertise, can be instrumental for various research objectives. They offer a dynamic platform for collaboration and knowledge exchange, Tremblay et al. [36] advocate their utility in design science.

In our pursuit of refining the AR-MINE tool and adapting it to the ever-evolving demands of the problem environment, we employed exploratory focus groups as a means to evaluate and enhance the deliverable artifact. The primary objective of the artifact is to address the problem statement, and its design and functionality must be continuously scrutinized and improved.

These focus groups allowed for rapid assessment and fine-tuning of the solution, and they ensured that any required adjustments in the solution approach were promptly integrated into the implementation. They proved invaluable not only in validating the design and functioning of the artifact but also in the process of exploring the context of the problem environment for additional insights. As the implementation of the solution progressed, the focus groups became more refined, facilitating deeper and more critical feedback that significantly contributed to the overall quality of the artifact.

In our context at Volvo Cars, we organized exploratory focus groups with experienced practitioners who played pivotal roles in handling and working with ARXML (Software Repos built on AUTOSAR). These sessions were scheduled tri-weekly and conducted through platforms like Teams. Each meeting was carefully structured with a well-defined agenda, covering topics such as the solution approach, the limitations of the existing solution, and the expected deliverable for the next iteration. The duration of these focus group meetings typically ranged from 30 to 45 minutes, varying based on the inclusion of the complexity of the deliverable. Given the intricate nature of our project, some iterations required longer discussions.

These focus group meetings ensured that the solution implementation progressed smoothly and allowed us to swiftly address any bottlenecks or roadblocks encoun-

tered during development. They were indispensable in tailoring our solution to the specific needs of our problem environment.

5.3.2 Interviews

Interviews help in gathering data from a sample. It gives a better understanding of the perspective of the interviewee. Given the intricate nature of AUTOSAR, interviews would be highly beneficial. They can be classified into fully structured, semi-structured, or unstructured. We are conducting a semi-structured interview with the stakeholders to have a good balance between pre-defined questions as well the dynamic benefits from the semi-structured interviews.

Interview Planning

We conducted interviews with experienced practitioners who work with AUTOSAR and handling ARXML repositories at Volvo Cars. After the set of people were identified, we collected their opinions individually to get their perspective. All the interviews were performed virtually through Teams.

Interviews were conducted with a group of eight Volvo Cars employees who possessed diverse roles within the organization and shared valuable expertise in ARXML. These interviews were critical in enhancing the functionality of the tool and gaining a comprehensive understanding of the problem from various perspectives. In Table 5.4, we provide an overview of each interviewee's role, the duration of their interviews, and their years of experience at Volvo Cars.

SNo.	Role	Duration	Experience
1	Senior Software Engineer	41 min	5+years
2	SW Function Developer	35 min	2+years
3	SW Integrator	46 min	3+years
4	SW Architect	72 min	10+years
5	SW Engineer	40 min	2+years
6	SW Release Manager	42 min	8+years
7	SW Developer	61 min	2+years
8	SW Integrator	53 min	6+years

Table 5.4: Interview participants

Performing interviews

Table A.1 in the Appendix section includes the list of all the interview questions that were asked. The interview questions were thoughtfully crafted to address key aspects of the research, focusing on the following areas:

Understanding the problem environment: The significance of monitoring a

project's growth and progress were questioned i.e. if knowing the growth of the project was essential. Additionally, recommendations for enhancements were requested.

Features: Inquiries were made about specific areas of interest within ARXML repositories and the rationale behind their choices. They also sought explanations for their decisions and their insights into the most useful metrics to consider when utilizing a software repository to gauge project growth. In order to understand project status, and maturity, metrics that they would like to extract from ARXML files stored in PESW git repos were the focus of the inquiry.

Future Actions and Use Cases: If the growth of the project was essential what decisions they would make and further information and features that were to be extracted from the repositories were discussed, along with possible next steps and uses for them.

5.3.3 Tool Evaluation Questionnaires

After the development of the tool , a questionnaire was used for the practitioners to answer the questions about the tool's effectiveness. The questionnaire involves five questions based on different metrics described in section 5.3.3. This step will answer the RQ3.

Metrics

1. **Satisfaction:** Showcasing whether the personnel were pleased with AR-MINE and its capabilities.
2. **Ease of use:** Measure describing how easily users can interact with AR-MINE.
3. **Efficiency:** Measure indicating how optimal the solution is when compared to the traditional approaches i.e. manual approach(Could be in terms of time and resources)
4. **Ease of understanding:** Measure indicating whether the solution is simple and easy to comprehend.
5. **Effectiveness:** Measure indicating the degree at which AR-MINE comes up with a reliable result, compared with a manual approach.

6.1 RQ1: Problem and Feature Identification

This section contains the findings related to RQ1. A step-by-step process is used below to clearly explain the problem identification and feature selection process. It consists of findings and feature selection, as well as motivation for selecting the features used in the tool.

6.1.1 Findings

The motivation of our thesis was to help practitioners with the capability of tracking software growth. In our literature review, we encountered various methods and tools employed for mining of software repositories and especially ones related to historic analysis.

Focus groups and interviews demonstrated that there is a need to understand the evolution of software for various practitioners for a variety of reasons depending on their roles. However, we rigorously stick to assisting practitioners in tracking the elements that they feel vital for understanding the progress of the software based on AUTOSAR, as AUTOSAR is quite complicated and specialized tools would be a valuable asset to the practitioners in understanding it. [27]

Literature Review

In our literature review, we examined existing research on software evolution, particularly focusing on commit and repository analysis. Despite the abundance of literature in this area, we found a notable gap concerning AUTOSAR mining within software repositories using ARXML files. Existing tools for repository analysis are generally not tailored to comprehensively analyze the evolution of AUTOSAR software, particularly in linking changes in ARXML to different system features. As a response to this gap, our research takes inspiration from Coming tool, which employs commit comparison, aiming to address the specific challenges of AUTOSAR mining using ARXML files. This adaptation seeks to enhance our understanding of software evolution within the context of AUTOSAR, contributing to the advancement of methodologies in this specialized domain.

Interviews

1. **On Problem Understanding:** The interviewees were posed with the question, "Have you assessed the growth of the AUTOSAR software? If yes, then why, and what are the current methods you use to track it?"

Scope of the question: To know the significance of monitoring software growth from a wider perspective gain insights into the usual methods employed by the interviewees to evaluate software growth

Responses: The responses from the interviewees showed that everyone is interested in knowing the growth of the software. The following were some of the responses:

- "Tracking how the project evolves is extremely important as communication and distributing information between the different stakeholders is the biggest challenge of the department because of its size and collaboration needs."
 - "The importance that each involved person is informed about the progress of a project can unfortunately vary. Some information such as time plans and high-level functionality layout would be relevant for most."
 - "Yes, especially to understand future capacity needs such as CPU runtime and memory consumption/growth."
 - "Yes, to make the AUTOSAR software integrating process easier."
2. **On Challenges faced:** The interviewees were asked "What are the current methods used and the challenges that they face for monitoring the growth of the AUTOSAR software?"

Scope of the question: The primary aim of this inquiry was to gain insights into the issues they face, with the intention of assessing whether existing solutions address these issues or not.

Responses: In response, practitioners commonly cited the manual retrieval of information as a prevalent challenge. Noteworthy excerpts from their responses include:

- "Manually, by a combination of multiple methods such as scanning the software repository, asking other colleagues who are also working on the same projects, which is a time taking and a little unreliable process"
- "I rely on the responsible SWA(Software Architects) to pass on high-level project information. I also try to keep updated on what the Product Owner & Scrum Masters hears and knows."
- "Manually going through the git repository. Generally, I think additional reoccurring meetings are an uninspiring solution to improve but from my experience, I believe there's an improvement to be found."

- "We do not have any tool to see statistical changes in the project, we try to do so by reading the content given by developers and inferring from git comments."

3. **On Feature identification:** The interviewees were asked "What are the features they were interested in extracting from ARXML files to facilitate the monitoring and analysis of AUTOSAR software growth?"

Scope of the question: The objective of this question was to know and document the features they deem crucial for tracking and assessing the software's evolution.

Responses: Most participants responded by indicating the features they wish to track, aligning with their respective roles within the organization. Some of them are:

- "software architecture, its components and the interfaces between them."
- "Number and frequency of commits of a specific SWC could possibly give a hint on maturity."
- "Changes like number of ports changes or changes in SWC, changes is calibration, change in software configurations."

The interviewees mentioned meetings with developers, skimming through large ARXML repos, and going through Git comments were the only ways to know the growth of the project and if it can be replaced by the tool. When asked about the average time they consume to do all these evaluations, they responded that it takes at least one to two days to go through the repo and know how they evolve as every commit generates a new ARXML file with new changes. Sometimes it also takes longer to do so depending on the requirement. For developers and integrators, it is a challenge to mine and go through the repository manually.

6.1.2 Features Identification

Table 6.1 displays information regarding the results of the interviews. According to the data in the table, the most prominent and frequently mentioned features in the interviews were software compositions, children software components (CSWCs), Pports, Rports, and Runnables associated with respective SWCs. The table shows the frequency of the features that appeared in the interviews.

6.2 RQ2: Automating the Mining of ARXML repo

This section provides the results of RQ2. The following step-by-step- procedure explains how the artifact (AR-MINE) was developed and built to automate automotive software tracking. It includes of iterations and improvements to the artifact in order to produce a well-designed artifact.

Features	Description	Frequency
Sw Composition	Names of software components with similar functionality	2
Sw Components	Names software components that encapsulate one or more algorithms	4
Rports	Names of communication signal that requires certain services or data	3
Pports	Names of communication signal that provides certain services or data	3
Interfaces	Specifies the type of (SWCs), if is an application, sensor, or actuator component.	1
Runnables	The sequence of operations performed by SWCs	2

Table 6.1: Features identified for automating growth monitoring from Interviews

Iteration 1 - Initial design

In the focus group meetings, the set of features that will help the practitioners to track the growth of the software repository were evaluated.

6.2.1 Feature Selecting Criteria

A set of criteria were developed to aid in the feature selection process. The features were filtered out using these criteria, and non-viable factors that were too complex/unfit to implement were ignored.

The following are the criteria that were considered:

C1: The feature is frequently mentioned in the Interviews.

C2: The feature extraction complexity is feasible.

C3: The difficulty of analyzing feature's data is achievable.

'Interfaces' was not considered as one of the implemented features as it was mentioned in the interviews only once and the feature extraction complexity was not feasible.

In our discussion, we systematically analyzed and correlated the features identified during the interviews that are related to understanding the evolution of software based on AUTOSAR.

6.2.2 Implemented Features

Software Components- These are the core building components for AUTOSAR applications. An SWC is a self-contained piece of software that performs a specific

automotive function, such as sensor data processing, engine control, or user interface management. AUTOSAR defines several types of SWCs, including application SWCs and communication SWCs. . Almost all the interviewees recommended this feature. [2]

Software Composition- AUTOSAR allows users to create sophisticated automotive software systems by mixing several SWCs. Software compositions are reusable blocks that explain how SWCs are linked together via their ports (PPorts and RPorts). This allows for a modular architecture in which functionalities are contained within SWCs and communication is facilitated via standardized interfaces. [2]

Rportprototype(Rports)- RPortPrototypes also referred to as RPorts, are the well-defined connection points for communication between different software components. A require-port (in technical terms: RPortPrototype) requires certain services or data. The communication signals between SWCs grow as the software grows which is interesting for software architects to observe in the tool. [5]

Pportprototype(Pports)-PortPrototypes also referred to as Ports, are well-defined connection points for communication between different software components. Provider-port (or PPortPrototype) on the other hand provides those services or data. [5]

Runnables- Runnables are executable chunks of code included within a SWC. They carry out specified activities prompted by events, signals, or timers. An SWC can contain numerous runnables that handle various functions. For example, a SWC that processes sensor data may contain different runnables for filtering raw data, conducting computations, and communicating the findings to other components. [4]

Artifact Design

The solution is intended to be initiated with a Python command, leading to the display of a dashboard containing comprehensive comparative analysis. AR-MINE was specifically crafted to handle two commits, involving two ARXML files. The process involves taking these files, mining the features, and presenting a web-based dashboard. The choice of a GUI is deliberate, as it offers a user-friendly interface that can be easily understood and analyzed by stakeholders with diverse levels of knowledge, ranging from developers to managers within the organization.

Iteration 2 - Improving the design

Following the initial iteration, group meetings were held with the stakeholders. During these sessions, unanimous agreement was reached on the motivation behind incorporating each feature into the solution. The employees showed satisfaction with the ease of use of the solution, executed through a single command in the Command Line Interface (CLI).

However, the group made a request to incorporate a feature, a storage solution for the extracted information. This addition aimed to establish a SQL based storage so-

lution, and provide a reliable and sophisticated means of storing data extracted from ARXML files. The adoption of a database facilitates easier comparisons, allowing for direct execution of queries on the database. This approach is deemed more efficient and effective in handling the mined features.

Furthermore, they emphasized the inclusion of specific criteria for the tool that it must satisfy for much better user experience in tracking the growth of the software. This also aims to facilitate a thorough evaluation of the tool's functionality and ascertain whether the solution aligns with the requirements and objectives of the stakeholders.

The requested acceptance criteria for the tool:

- Identification of feature changes, including added, changed, and deleted features within the repository.
- Total count of features altered in a single commit when compared to another.
- Comparison of any given commit with another commit.

Iteration 3 - Design Acceptance

The third iteration involved a thorough examination through the assessment of 10 distinct ARXML files, representing 10 different commits, with varying structures and commit lengths. AR-MINE consistently produced the expected results for all the commits under scrutiny. The processing time for each pair of comparisons varied in accordance with the total number of lines within a given commit. In Iteration 3, identified bugs were addressed, contributing to an overall enhancement of the tool's reliability. Furthermore, adjustments were made to the algorithm to expedite the execution and analysis of substantial ARXML files. A crucial aspect of this phase was the introduction of the tool through a technical demonstration.

Stakeholders expressed their satisfaction with AR-MINE and conveyed a preference for the addition of new features in the future, with the goal of reducing manual file analysis.

Tool's Overview

AR-MINE is a software repository mining tool leveraging an Etree algorithm for data mining. The tool is composed of four distinct microservices. AR-MINE comprises of a web-based GUI. These microservices were intentionally designed to operate semi-independently, each with a specialized responsibility.

The framework of AR-MINE is illustrated in Figure 6.1, where lines depict significant data channels, their direction, and corresponding response data. The microservices being: Etree Parser, Database, Query Interface, and the Front-End. Each microservice plays a unique role in the overall functionality of AR-MINE.

The figure 6.1 shows the overview of the tool, where ARXML files are taken from the repository. Data is gathered and then the parsing algorithm is applied to obtain

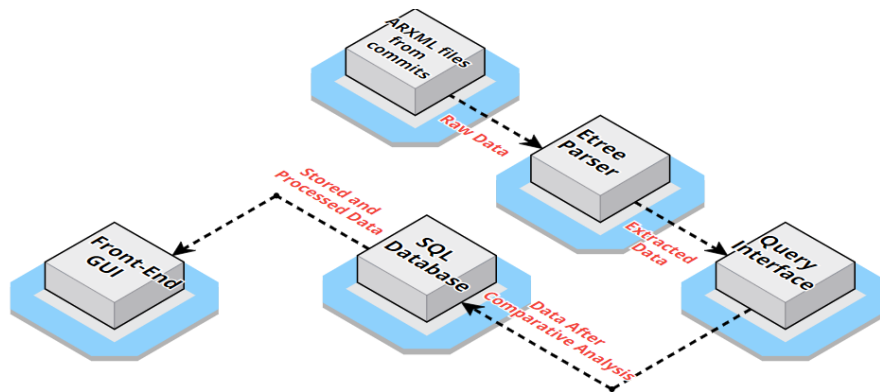


Figure 6.1: AR-MINE Overview

the mined data.

Etree Parser

The Etree algorithm loads the data from ARXML files. It uses the data and parses the data-points and the information required for knowing the growth of the project. The xml.etree. ElementTree module implements a simple and efficient API for parsing ARXML files.

Database

The parsed data is then stored in the database, the database used here is SQLite. The parsed information is stored in a cardinal structure for better understanding on how SWCs are linked with what ports and runnables.

Query Interface

It is in charge of running queries that compare the ARXML files that are stored in the database as tables, i.e. comparing the database tables. To evaluate how the project evolves, it is important to know the changes that take place from one ARXML to another, whenever there is a commit.

Front-End

The Front-end service, is the user's perspective of AR-MINE and allows them to visualize the differences between the commits i.e. ARXML files. It provides a dashboard with the commit id and the number of changes from one commit to another.

Following the development of AR-MINE, practitioners/stakeholders will be able to access it. Surveys were used to collect the feedback after the demo of the tool was given. AR-MINE is written in Python and Streamlit. The system handles a large portion of the data and has the potential to be optimized in the future.

Tool's Program Flow

Figure 6.2 illustrates the program flow of the automated solution AR-MINE. The

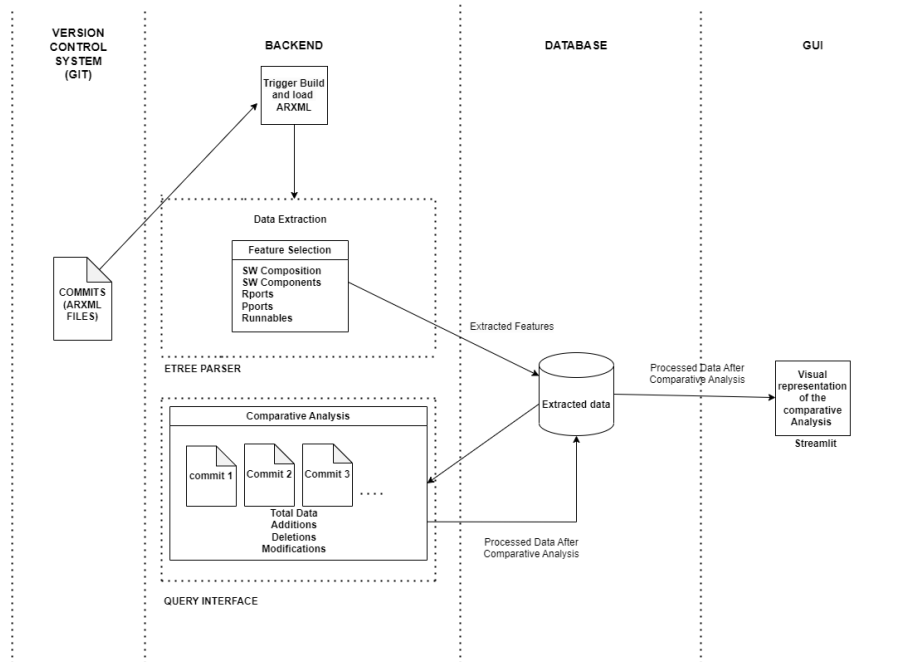


Figure 6.2: Program flow of AR-MINE

step by step working of the system is as follows:

1. Two ARXML files, representing distinct commits randomly selected from the software repository, are designated as input. Command-line arguments specify their filenames, and it is imperative that these files reside in the tool's directory.
2. The system employs an advanced parser algorithm to meticulously analyze the contents of the provided ARXML files. The primary objective is to extract specific data, particularly focusing on features identified in the context of Research Question 1 (RQ1).
3. Extracted data, including identified features, are stored in the database. This storage phase serves as a crucial intermediary step, laying the foundation for subsequent comparative analysis.
4. Applying SQL queries, AR-MINE conducts a thorough comparative analysis between the two selected ARXML files. Results from these queries are stored back into the database, encapsulating the findings of the comparative evaluation. Simultaneously, a .db file is generated within the tool's directory.
5. To enhance user interaction and comprehension of the comparison results, AR-MINE utilizes Streamlit, a Python library for developing web-based graphical user interfaces (GUI). This GUI serves as a visual representation of the comparison outcomes, offering stakeholders an intuitive and insightful view of the analyzed data. This step aims to facilitate a clearer understanding of the results, enabling users to interpret and utilize the findings more effectively.

Usage

The evaluation of commits through AR-MINE involves the use of a specific command in the Command Line Interface (CLI).

```
PS C:\Users\VSQNI1\Downloads\flaskn> & c:/Users/VSQNI1/Downloads/flaskn/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\VSQNI1\Downloads\flaskn> & 'c:/Users/VSQNI1/Downloads/flaskn/.venv/Scripts/python.exe' 'c:/Users/VSQNI1/.vscode/extensions/vms-python.pyth
on-2023.10.1/pythonfiles/lib/python/debugpy/adapters/..\.debugpy\launcher' '56298' '--' 'c:/Users/VSQNI1/Downloads/flaskn\AR-MINE_v1.0.0.py'
5f0f724.arxml
6cde721.arxml
```

Figure 6.3: Code Snippet 1: Basic Usage

Each commit is identified by a unique commit ID, and the user is required to provide two ARXML files, symbolizing two separate commits, as command arguments. The comparison queries are executed automatically on the extracted data.

The command example is illustrated in Figure 6.3.

Results

Figures 6.4, 6.5, 6.6 and 6.7 present the reports derived from the analysis of two sources: Streamlit and the .db file. These reports comprehensively integrate all features identified in RQ1, offering a representation of outcomes within the context of the growth of software built on AUTOSAR.

port	runnables	Serial Num	parent	r_port	p_port	runnables	Se
0	one	Run_DiagMgrFast_Init	45	SWccEm	None	None	
1	one	Run_DiagMgrFast_Step	46	SWccEm	None	None	
2	one	Run_EmDcBst_Init	47	SWccEm	None	None	
3	one	Run_EmDcBst_Step	48	SWccEm	None	None	
4	one	Run_HvtrCtrlActvDcha_Init	49	SWccEm	None	None	
5	one	Run_HvtrCtrlActvDcha_Step	50	SWccEm	None	None	
6	one	Run_HvtrCtrlCoolgReq_Init	51	SWccEm	None	None	
7	one	Run_HvtrCtrlCoolgReq_Step	52	SWccEm	None	None	
8	one	Run_HvtrCtrlFltMgr_Init	53	SWccLtr	None	None	
9	one	Run_HvtrCtrlFltMgr_Step	54	SWccEm	None	None	

Figure 6.4: Snippet 1: GUI

Figures 6.4 and 6.5 illustrate the graphical representation of alterations in features between two commits, encompassing compositions, components, children ports, and runnables. Whereas, figures 6.6 and 6.7 encompasses all the data extracted from the files before and after the comparison process.

In an attempt to accommodate the acceptance criteria devised by the stakeholders during the focus group meetings, two random ARXML files representing two commits were selected, and the data from both the commits was stored in the database. Subsequently, a comprehensive comparative analysis was conducted on the extracted data. Below is the acceptance criteria 6.2.2

- The aggregate data obtained from both commits.

r_port	p_port	runnables	sort	p_port	runnables	Serial Number
75	None	yivcCtrl	None	45	ccEm	4E
76	None	yivcEm	None	46	ccEm	41
77	None	yivcEm	None	47	ccEm	4E
78	None	yivcEm	None	48	ccEm	4E
79	None	yivcEm	None	49	ccEm	5C
80	None	yivcEm	None	50	ccEm	53
81	None	yivcEm	None	51	ccEm	52
82	None	yivcEm	None	52	ccEm	5E
83	None	yivcEm	None	53	ccEm	5F
84	None	yivcEm	None	54	ccEm	5E

Figure 6.5: Snippet 2: GUI

component_parent	rport_name	pport_name	runnables_name
Filter	Filter	Filter	Filter
DiagMgrFast	NULL	NULL	NULL
2	sVccPhaDiagFltPrsnt	NULL	NULL
3	EcuMVccActivationMode	NULL	NULL
4	DiagMgrFastEvTrigg	NULL	NULL
5	NULL	sVccDiagDGateDrvrModReq	NULL
6	NULL	NULL	Run_DiagMgrFast_Init
7	NULL	NULL	Run_DiagMgrFast_Step
8	EmCalc	NULL	NULL
9	sVccEmCurTarTqCmp	NULL	NULL
10	sVccEmCurTarTqIdx	NULL	NULL
11	sVccEmCurCtrlAgOffs	NULL	NULL
12	sVccEmCurCtrlAgRot	NULL	NULL
13	sVccEmCurCtrlID	NULL	NULL
14	sVccEmCurCtrlIQ	NULL	NULL
15	sVccEmCurCtrlID	NULL	NULL
16	sVccEmCurCtrlIQ	NULL	NULL
17	sVccEmCurCtrlPsiEm	NULL	NULL
18	sVccEmCurCtrlIUD	NULL	NULL
19	sVccEmCurCtrlIUQ	NULL	NULL

Figure 6.6: Database File Snippet 1

Name	Type	Schema
T_5f0f724	CREATE TABLE	T_5f0f724 (composition_name text, component_name text)
T_5f0f724_PORTS	CREATE TABLE	T_5f0f724_PORTS (component_parent text, rport_name text, pport_name text, runnables_name text)
T_6cde721	CREATE TABLE	T_6cde721 (composition_name text, component_name text)
T_6cde721_PORTS	CREATE TABLE	T_6cde721_PORTS (component_parent text, rport_name text, pport_name text, runnables_name text)

Figure 6.7: Database File Snippet 2

- A granular view of additions made in one commit relative to the other.
- Deletions recorded in one commit as compared to its counterpart.
- The total changes witnessed across both commits.

For a detailed breakdown of the diverse total features extracted from both the commits, refer to Table 6.2. The tabulated numerical data provides valuable insights into the dynamic evolution of the software project. The findings of the comparative analysis were visually represented using a line graph, which effectively illustrates the distinctions among the commits.

	Commit	Composition	Component	Rports	Pports	Runnables
Data	Commit1	64	672	4800	2112	1408
	Commit2	56	416	3520	1696	896
Addition	Commit1	10	287	1292	439	530
	Commit2	2	31	12	23	18
Deletion	Commit1	2	31	12	23	18
	Commit2	10	287	1292	439	530
Changes	Both	12	318	1304	462	548

Table 6.2: Comparative Analysis of the extracted Data

Visualizing the aggregate data from both commits provides a common and easily understandable representation of the changes made to the software. This facilitates communication between stakeholders, enabling them to share and discuss the evolution of the software in a clear and concise manner. It reveals the trends and patterns in the changes made to the software. In addition, it also includes the introduction of potential interactions between components.

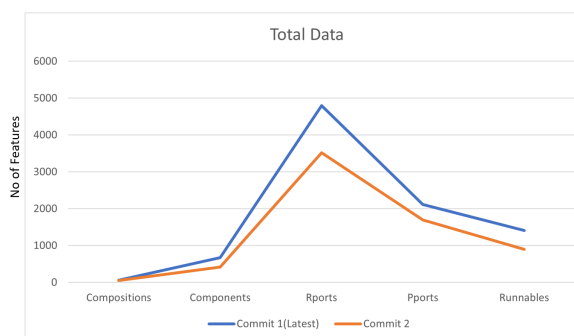


Figure 6.8: Comparison of the Total data extracted from both the commits

The graphical representation in Figure 6.8 encapsulates the comprehensive dataset derived from both commits and their corresponding comparative evaluations. Notably, Commit 1, is the most recent addition to the software repository. Specifically, Commit 1 exhibited a composition of 64 software elements, accompanied by 672 software components. Moreover, this commit manifested a compilation of 4800 Rports, 2112 Pports, and 1406 runnables.

In contrast, Commit 2, the antecedent state of the repository, displayed a slightly different configuration. It was characterized by 56 software compositions, along with 416 software components. Additionally, Commit 2 encompassed 3520 Rports, 1696 Pports, and 896 runnables.

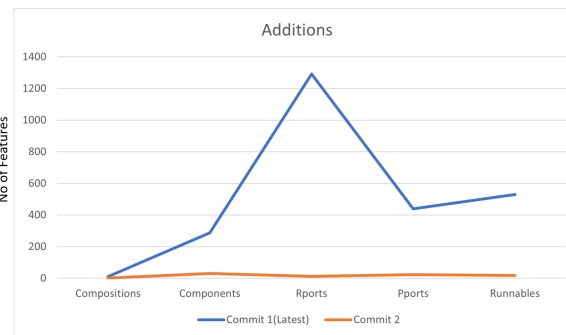


Figure 6.9: Additions of features in both the commits

In the view of additions of features made in one commit relative to the other, practitioners can identify specific areas where new features have been introduced. This provides insights into the evolution of the software and can potentially help them plan for future development. It also pinpoints the specific areas where changes have been introduced.

Figure 6.9 gives a visual look at how different features changed when the two commits were compared. Focusing on Commit 1 first. It showed an increase of 10 new software compositions, along with 287 new software components, 1292 Rports, 439 Pports, and 530 runnables. This means Commit 1 got bigger and more complex compared to the earlier commit.

Now, for Commit 2, the changes were a bit smaller. It had 2 new software compositions added, along with 31 new software components. On top of that, there were 12 new Rports, 23 Pports, and 18 runnables introduced when compared with Commit 1 (Latest). Figure 6.9 and these changes help us understand how these commits evolved and grew differently from each other. This information adds to the research by showing how the software developed over time.

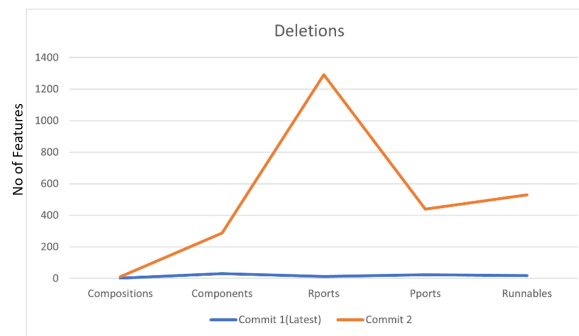


Figure 6.10: Deletions of features in both the commits

A detailed view of deletions in one commit compared to its counterpart, helping engineers focus on areas where code was removed. In Figure 6.10, the focus shifts to the removal of features within both commits. Beginning with Commit 1, a reduction is evident, involving 2 software compositions and 31 software components. Additionally, 12 Rports, 23 Pports, and 18 runnables were removed, signifying a refinement of its composition.

Transitioning to Commit 2, the scope of deletions becomes more pronounced, which is unsurprising given its status as a prior commit. With fewer features added compared to the latest iteration, Commit 2 exhibited a more substantial removal of elements. Specifically, this included the elimination of 10 software compositions, 287 software components, 1292 Rports, 439 Pports, and 530 runnables. These deletions in Commit 2, when contrasted with the features in Commit 1 (the latest version), emphasize the evolution of the software in terms of its feature set over time.

The insights gleaned from 6.10 and these deletion trends contribute to a deeper understanding of our research, enabling a comprehensive view of how the software's makeup has changed across different iterations.

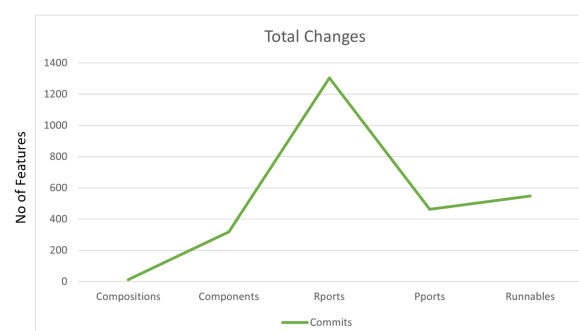


Figure 6.11: Total Changes in both the commits

Visualization of total changes offers a comprehensive view of the collective modification activities, that could enable practitioners to pinpoint regions characterized by frequent alterations or volatility.

The provided Figure 6.11 offers a visual depiction of the cumulative alterations observed in the two distinct commits under consideration. These commits encompass a modification of 12 software compositions, 318 software components, 1304 Rports, 462 Pports, and 548 runnables, each contributing to the overall evolution of the software system. This graphical representation serves to highlight the comprehensive nature of changes implemented within these commits.

6.3 RQ3: AR-MINE Feasibility

This section contains information related to the results regarding AR-MINE's functionality and usability, using the feedback provided by the developers.

User Feedback

Following a thorough comprehension of the interviewees' perspectives on monitoring software evolution, a technical demonstration was conducted to showcase the solution, AR-MINE. Subsequently, user feedback was collected to assess the effectiveness of AR-MINE. The questions posed to the interviewees for user feedback are outlined in Table 6.3.

ID	Questions
Q1	To what extent do you find the overall functionality of AR-MINE, satisfactory in facilitating the requirements?
Q2	How user-friendly is AR-MINE for monitoring the growth of AUTOSAR software?
Q3	How efficient is AR-MINE in effectively tracking the growth of AUTOSAR software?
Q4	In your experience, how easy it is to understand AR-MINE for tracking the evolution of software based on AUTOSAR?
Q5	How would you rate the effectiveness of AR-MINE in tracking the growth of AUTOSAR software?

Table 6.3: User Feedback Questionnaire

The composition of interviewees leans significantly towards practitioners who possess knowledge about the structure and content of ARXML files. The roles represented encompass a diverse spectrum, including manager, developers, integrators, and architects within the organization.

All participants acknowledge the significance of tracking the growth of software based on AUTOSAR, yet their perspectives vary for various reasons, as identified in RQ1. This diversity results in a heterogeneous mix of roles and experiences related to assessing software growth.

Table 6.4 presents the distinct roles and their responses to the Likert scale question.

Roles	Q1	Q2	Q3	Q4	Q5
Senior SW Engineer	Agree	Neutral	Strongly Agree	Agree	Agree
SW Function Developer	Strongly Agree	Agree	Agree	Agree	Agree
SW Integrator	Neutral	Disagree	Agree	Strongly Agree	Strongly Agree
SW Architect	Strongly Agree	Agree	Agree	Agree	Neutral
SW Engineer	Agree	Agree	Neutral	Strongly Agree	Agree
SW Release Manager	Agree	Neutral	Strongly Agree	Agree	Neutral
SW Developer	Agree	Strongly Disagree	Agree	Agree	Agree
SW Integrator	Strongly Agree	Agree	Agree	Agree	Strongly Agree

Table 6.4: User Feedback Likert Scale

Figure 6.12 shows the result related to question one of the questionnaire (satisfaction), where the employees were asked whether AR-MINE was helpful in tracking the growth of the software. The results show that 50% of people agree, 37.5% of people strongly agree and the remaining were neutral i.e., 87.5% agree that AR-MINE is useful in mining ARXML files. The Pie chart shows that the majority of the employees were positive about the satisfaction of the tool and think that the tool has the potential to help practitioners.

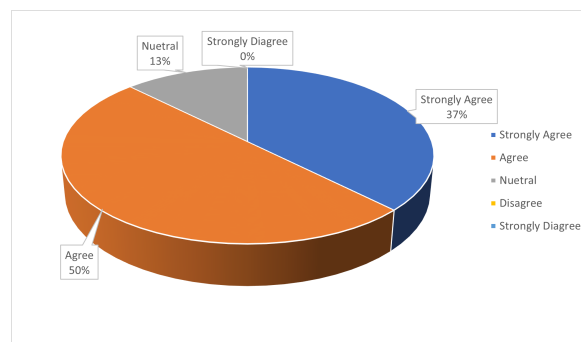


Figure 6.12: Q1: AR-MINE Satisfaction

Figure 6.13 shows the result related to the second question of the questionnaire, where the employees answer whether AR-MINE is user-friendly. The response obtained for this question was that 50% of people agreed, 25% of people were neutral, 12.5% of people disagreed and 12.5% Strongly disagreed. It had an equal distribution of solutions compared to other questions, but there was a total of 50% of positive and 50% of negative responses.

While the solution for feature extraction resonated positively, stakeholders criticized the ease of use as they had to manually do the build process to generate ARXML files from Git and download them locally. Since it requires the incorporation of Git plugins into the tool and given the time constraints it was not possible to achieve that.

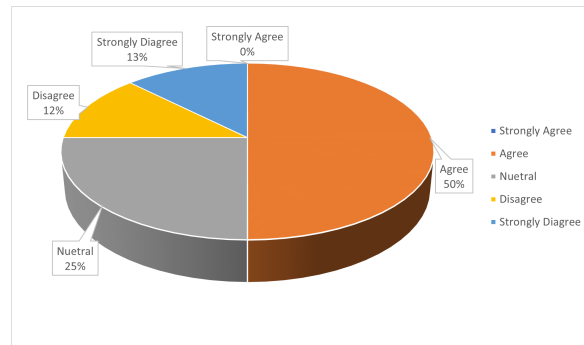


Figure 6.13: Q2: AR-MINE Ease of Use

Figure 6.14 shows the result related to question three of the questionnaire, where the employees were asked whether AR-MINE helps in decreasing the overall time of tracking the growth of software when compared to their manual approach. The responses show 62.5% of people agree, 12.5% of people Strongly Agree, the and remaining were Neutral. This does explain why the majority of the people were positive and found the tool efficient and some of the employees were neutral.

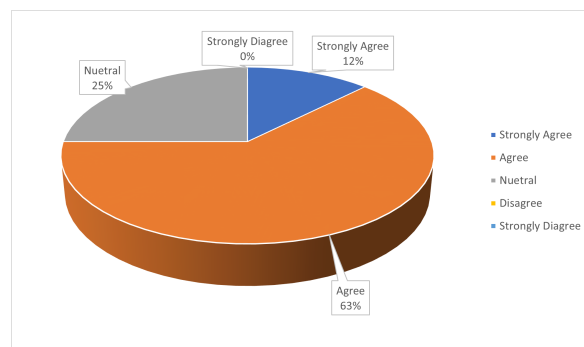


Figure 6.14: Q3: AR-MINE Efficiency

Figure 6.15 shows the result related to question four of the questionnaire (Ease of understanding), where the developers were asked whether AR-MINE is easy to understand. 75% of people agree, and 25% of people strongly agree i.e., 100% agree that AR-MINE is easy to understand.

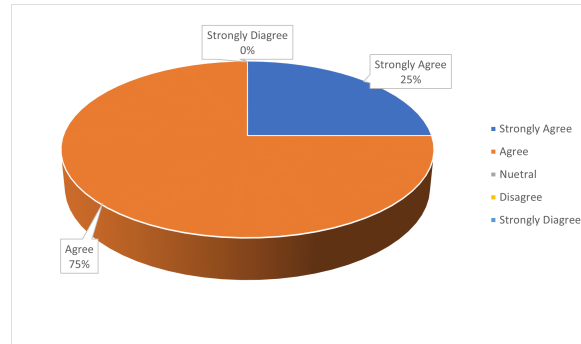


Figure 6.15: Q4: AR-MINE Ease of understanding

Figure 6.16 shows the result related to question five of the questionnaire (Effectiveness), where the developers answer whether the mining of ARXML files is accurate and reliable. The graph shows that 50% of people agree with 37.5% of people strongly agreeing while the remaining were neutral i.e., 87.5% agree that the results of AR-MINE are reliable. This shows that the practitioners feel AR-MINE is good at what it does.

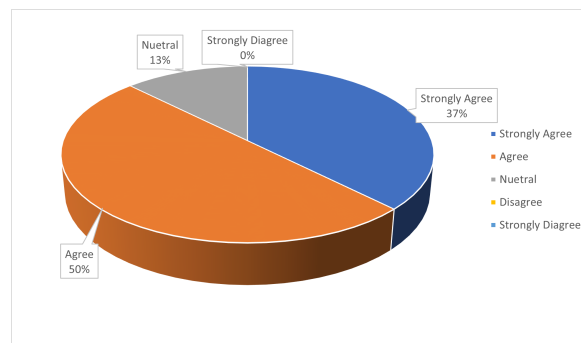


Figure 6.16: Q5: AR-MINE Effectiveness

After conducting the survey, individual discussions were initiated with participants to delve deeper into their feedback regarding the tool. These conversations unveiled varying perceptions of the tool(mentioned in the scenarios below), shaped by the participants' expertise and organizational roles. Though the primary goal of the thesis is to aid the practitioners in tracking the growth of the software built on AUTOSAR as it is complex and difficult to manage [27] through mining the elements they deemed necessary for evaluating the growth from the ARXML files, the tool could potentially, but not necessarily help the practitioners in the following hypothetical scenarios:

Scenario 1: A release manager is prepared to roll out a new software release. Guaranteeing the release's stability and completeness necessitates a careful review procedure to avoid the unintentional loss of important functionality.

Challenge: Manually checking code and documentation to ensure that all components are included in the new version can be time-consuming and error-prone. Overlooking deleted components may result in unanticipated capability loss in the deployed product

The manager checks the ARXML files from the stable version (Commit1) and the next release candidate (Commit2). AR-MINE compares the changes in "Components" across versions. A considerable drop in the number of components (e.g., "-100") in Commit2 triggers a red alert, suggesting that a large number of SWCs are missing in comparison to the stable version. Based on AR-MINE's observations, managers can explore the reason of the missing components:

- Consult with developers to learn why certain SWCs have been removed/added.
- Examine code annotations to determine the purpose of deleted components.

Potential Benefits:

- Early detection of missing functionality.
- Reduced risk of release issues.
- Improved Release Confidence.

Scenario 2: A software integrator is given a task for integrating many SWCs created by separate teams into a single, coherent software stack. Maintaining consistent composition and communication routes across versions is critical to effective integration.

Challenge: Traditionally, ensuring SWC composition consistency requires manually comparing code and documentation, which may be time-consuming and error-prone. Inconsistencies in SWC assembly might cause integration concerns and destabilize the whole system architecture.

The integrator checks "Composition" entries and communication data (Pports - Providing Ports, Rports - Receiving Ports) in ARXML files of various versions (for example, Commit1 and Commit2). AR-MINE identifies changes in the number of SWCs, their connections, and communication pathways. Let us say he identifies following inconsistencies:

- Differences in "Composition" (for example, a drop in Commit2) may indicate that SWCs are absent.
- Variations in "Pports" and "Rports" (for example, the rport "sVccDiagDGateDrvrModReq" is missing in one of the commits) indicate probable changes in data flow between components.

Benefits:

- Early Problem Identification
- Maintaining the coherence of the system

Scenario 3: A software architect is tasked with managing the growth of the architecture and identifying areas for improvement. The task requires a thorough understanding of how the number of components, their connections, and total complexity vary with time.

Challenge: Traditionally, monitoring architectural changes has involved human code inspection and analysis, which may be time-consuming and provide a limited overall picture.

Architect examines ARXML files from multiple versions (e.g., Commit1, Commit3, and Commit5) to monitor changes across time. AR-MINE offers information on the amount of SWCs ("Components"), their functionality ("Runnables"), and communication channels ("Pports" and "Rports"). Suppose AR-MINE consistently increases "Components" (e.g., 10% each version) and "Runnables" (e.g., 15% each version) throughout several releases.

This points to a rising codebase complexity. This information can be utilized to push for architectural restructuring or to investigate component consolidation solutions.

Benefits:

- Data Driven Decisions
- Proactive Optimization
- Improved maintainability.

Scenario 4: A senior engineer wishes to evaluate a sophisticated code update submitted by a developer. Historically, determining the possible impact of such changes on total system complexity has been difficult without a thorough understanding of the changes.

Challenge: Manually assessing the impact of code changes on complexity is a time-consuming task.

AR-MINE examines the ARXML files prior to (Commit1) and following (Commit2). Let us say it detects the difference in the number of "Components" between Commit1 and Commit2 and finds out that there is a big increase (+200) in "SWC" indicates a considerable increase in complexity. He can then investigate:

- The justification for greater functionality.
- Alternative implementation options that might produce comparable outcomes with less complexity.

Benefits:

- Improved decision-making
- Reduced risk of regression

- Early identification of optimization opportunities

Scenario 5: A software developer suspects that a recent upgrade (Commit2) caused a flaw in the system.

Challenge: Debugging would be the go-to approach but it may be time-consuming and need manual code analysis to discover root causes.

AR-MINE examines the ARXML files from the working version (Commit1) and the malfunctioning version (Commit2). It displays changes in different aspects (e.g., Composition, Components, Runnables, Pports, and Rports) across versions. Assume AR-MINE detects a change in a certain SWC (present in Commit1 but absent in Commit2) and changes to functionality ("Runnables") within another SWC (Commit2). This might indicate that data flow to the missing SWC is causing an issue. The developer may concentrate his/her debugging efforts on these sections of the code.

Benefits:

- Reduced Debugging Time
- Data Driven Debugging

Scenario 6: A software function developer has been charged with developing a new functionality for the AUTOSAR application. To enable seamless integration and avoid conflicts, she must understand how her new feature will interact with existing components. This entails identifying possible dependencies with existing Software Components (SWCs) from which her new feature may get data.

Challenge: Historically, dependency analysis has been a time-consuming and error-prone process. Manually sorting through code and documentation may be time-consuming, and failing to consider critical dependencies might result in integration difficulties later in the development cycle.

AR-MINE analyzes the current system's ARXML file (representing Commit2) to a baseline version (which lacks the additional capability - Commit1). AR-MINE examines the "Deletion - Commit1" row for "Rports" (the ports used by the developer's new SWC to receive data from others). If the "Deletion - Commit1" entry for "Rports" has a large value (e.g., 100), it indicates that the new SWC may have previously relied on data from components that are no longer available in Commit2. The developer can then:

- Investigate Removed Components.
- If the eliminated components were critical for the new SWC functionality, she can collaborate with the team to seek alternate data sources.

Benefits:

- Reduced time investment.

- Early Identification of Issues
- Streamlined Integration

The scenarios showcase that AR-MINE may not directly resolve issues related to software development/management. However, it could assist the practitioners in being an auxiliary tool that could potentially give rise to better outcomes when compared to their approach without a customized-tool.

7.1 RQ1: What are the features and metrics that are required to understand the growth of software built on AUTOSAR?

In addressing RQ1, our study sought to determine the essential features and metrics that the practitioners deem necessary to be extracted from the software repository for understanding the growth of the software based on AUTOSAR. Contrary to the usual metrics such as SLOC(Source Lines of Code), completed user stories, these metrics are much more specific to software that is built upon AUTOSAR. The study benefits from [26] [16] where they showcased the importance of analyzing minute code edits and extracting code change patterns using specific elements from the repositories and makes a contribution in terms of giving new insights on the relationship between different features and the growth of software in the context of AUTOSAR and automotive domain. Existing research focused on the study [14] takes a step further towards understanding AUTOSAR's infamous complexity and comprehending a better view of the software built on it.

However, the generalizability of the results is limited by the sample size and also by the organization the sample belongs to. The sample belongs to a single automotive company. Due to the lack of data from a broad audience(belonging to different automotive companies), the results cannot confirm the usage of same set of features across the entire automotive industry. It is evident that knowing the growth is important, but it cannot be concluded that the entire automotive industry is deeply invested in tracking the growth of software through the same set of features. However AUTOSAR is a standard framework used by a wide range of companies and the fact that it brings increased productivity and complexity to the table remains true. This solidifies the usage of these features, as these elements are found in every ARXML file regardless of the company. Further research is needed to establish the generalizability and potential future avenues would be gathering more information related to the complexities of AUTOSAR and where does knowing the growth fall among them. Profound research on metrics for understanding the growth of software in the automotive domain would also help in knowing where the current study stands in terms of a wider audience.

Durisic et al. [14] and Falleri et al. [16] emphasized the importance of analyzing meta-model changes and fine-grained code edits. Metrics like the size of the code, cohesion, and coupling are general features. The features identified in our study align with these works, focusing on extracting information from ARXML files. The similarities with the Coming tool [26], which also extracts code change patterns, are evident. However, the uniqueness lies in tailoring these features to the specific requirements of software based on AUTOSAR.

While Durisic et al. concentrated on meta-model transformations and features like coupling, cohesion, and size of the code, our focus was on source code modifications within the System Description file(ARXML) to enhance the granularity of understanding. This distinction becomes particularly important when considering the intricacies of software built on AUTOSAR, as highlighted by Gall et al. [18]. Our research, building on Durisic et al.'s foundational work, emphasizes the significance of features like Sw Composition, Sw Components, Rports, Pports, and Runnables in providing comprehensive insights into software evolution.

7.2 RQ2: How can the automation of mining and tracking recognized features in the ARXML repository be achieved?

Concerning RQ2, the automation of mining and tracking recognized features in the ARXML repository is a critical aspect of our research. Design Science Methodology enabled us to build upon the results of RQ1, and confine them based on the requirements from the initial iteration. The prototype was then iteratively improved based on the feedback. In the later stages, the practitioners provided an acceptance criteria that AR-MINE must satisfy. AR-MINE was accepted after a technical demonstration and thorough examination to ensure that it functions accordingly.

The results fit in with [26] where the author emphasized the importance of automated tools in uncovering code change patterns and introduced Coming tool. But, the results contribute to a different and much more specific scenario of ARXML files and AUTOSAR.

The contribution of AR-MINE lies in its adaptability to various commits, offering users more control over the analysis. This flexibility is a notable improvement over the consecutive commit-focused approach of the Coming tool. Our research aligns with the broader trend of advancing automation in software repository mining, addressing the limitations of existing tools.

However, it is essential to acknowledge certain constraints within the development timeline. Due to time constraints, we were unable to advance beyond a third iteration, primarily focusing on automating the feature extraction from the ARXML repository. Further iterations could have potentially led to improvements in the monitoring and upgrading functionalities. Additionally, the testing of the tool was not

7.3. RQ3: How feasible is AR-MINE in assisting the practitioners in tracking the growth of the AUTOSAR based software?

thoroughly conducted. Although compliance checks were performed during technical demos, testing within the production environment could provide valuable insights into the tool's behavior. The ability to work with extensions such as Git plug-ins would be an optimal solution to consider, as it drastically improves the user experience.

7.3 RQ3: How feasible is AR-MINE in assisting the practitioners in tracking the growth of the AUTOSAR based software?

The capability of AR-MINE in tracking the growth is a critical consideration for validating the contribution of the research. AR-MINE was generally well-received for its potential to automate and simplify AUTOSAR software repository mining. Practitioners appreciated the ease of use facilitated by a single command execution in the command-line interface (CLI). The user feedback consistently highlighted the tool's effectiveness and efficiency in extracting features from the repository. While the automated solution for feature extraction resonated positively, stakeholders criticized the ease of use as they had to manually do the build process to generate ARXML files in GIT, and download them locally. The tool's efficiency and effectiveness in this domain were appreciated and well rated than the ease of use.

However, The reliability of this data is impacted by factors such as the sample size of the interview and also the role of the interviewee. Tracking the growth being a use case of a broad scope, invites biased perspectives depending upon the role and how much does tracking the growth would be beneficial to them on a personal level. Due to the lack of concrete data on the tool's impact on role specific use cases, the results cannot confirm the absolute potential of the tool in satisfying the scenarios mentioned 6.3. Further research is needed to assess how AR-MINE could meet role-specific requirements.

In conclusion, our study, inspired by seminal works in the field, contributes to the field of software repository mining. The adaptation and extension of existing tools for AUTOSAR based software ensure relevance and effectiveness in a domain with a reputation of being highly complex. The discussion above showcases the synergies and differentiators of our work compared to related research, highlighting the novel contributions and advancements brought forth by AR-MINE in the context of studying the evolution of software built on AUTOSAR.

8.1 Internal Validity

Internal validity refers to the extent to which research accurately measures what it was intended to without any bias. It indicates the measure to which the findings are credible and reliable [7].

Potential threats to internal validity may arise due to a relatively small sample size of interview candidates because of various constraints such as insufficient time and late delivery of the tool, making it harder to gather a new audience and, in turn, familiarise them with the tool and the data behind it.

Despite being a small set, interviewees were highly experienced in the automotive domain and were familiar with ARXML file structure and the data to be gathered, making them a credible choice for validating our tool. We ensured to keep the buffer time between the technical demonstration and the feedback questionnaire as low as possible to ensure that the context and relevance is not reduced. The experience of the candidates would benefit us in defending the relatively small sample set for interviews.

8.2 External Validity

External validity refers to the degree to which the research findings can be generalized or applied to other settings or conditions beyond the study context [7].

In our investigation, we focused on ARXML files, exploring the potential of mining them and monitoring the growth of the software, ultimately creating an auxiliary tool to facilitate the research objectives. While our study can serve as a valuable guide for addressing similar challenges in specific use cases, it is important to note that the research was conducted in collaboration with Volvo Cars.

The selection of participants for the interviews and survey were exclusively from Volvo Cars which may introduce a threat towards perspectives and experiences specific to this organization, potentially overlooking insights from individuals outside this context. Consequently, the outcomes and insights are particularly tailored to ARXML data and hold significant relevance for industries involved in automotive software development using AUTOSAR, specifically for Electronic Control Unit

(ECU) software development. It is important to consider that the need as well the approach could differ from company to company. As it could be dependent on the company's requirements and how important it is for the company to understand and monitor the growth of the software.

8.3 Construct Validity

Construct validity examines whether the measures used in the research truly capture the intended concepts [7]. In the case of AR-MINE, a potential threat to construct validity might be employees at Volvo Cars misinterpreting the questions related to the tool's measures. In order to minimize the threat, information on the tool's usage was thoroughly presented and the context of each question from the feedback questionnaire was clearly explained. Additionally, in the semi-structured interviews, where discussions and opinions were allowed, we framed questions specifically around the topic of AR-MINE, software repository mining, and related concepts. This approach aimed to keep the discussions aligned with the research focus and minimize deviations that could impact the validity of the constructs being measured. If employees deviated significantly, we took steps to guide the conversation back to the research-related aspects, ensuring the integrity of the construct validity.

9.1 Conclusion

Our research was conducted to gain a comprehensive understanding of software built upon AUTOSAR and develop a customised tool that helps in tracking the growth of automotive software through ARXML files. The following issues were intended to be addressed:

- Gaining a deeper understanding on AUTOSAR and software built on it.
- Identifying the features that practitioners believe were necessary to understand the growth of software built on AUTOSAR.
- Developing a tool that extracts and stores relevant data from ARXML files, enabling retrieval and comparison of different commits, and facilitating a better understanding related to the growth of the automotive software.
- Validate and evaluate the developed tool in an industrial setting.

We conducted interviews to identify the features that influence the overall ECU software integrating process in order to answer RQ1. From RQ1, a list of features was finalized, which served as a baseline for determining and implementing a method for developing the tool. Section 6.2.1 describes all the features used in the development of AR-MINE, as well as the motivation for selecting each feature.

9.2 Future Work

The user evaluation of the AR-MINE was mostly positive. However, the tool could be improved in certain areas to make it much better and easier to use. Due to time constraints, it was not possible to add more capabilities.

One of our interview candidates stated that the tool's usability could be improved further. Front-end libraries that are industry standard, such as React, can be used to make it more user-friendly and tailored to the needs of the practitioners. The addition of the Git plugin would increase the efficiency in retrieving the ARXML files, making the tool more self-contained, secure, and robust could be another possible improvement, given that it was created with a simple, lightweight web development framework (Streamlit).

These are the areas of improvement:

- Enhancing the AR-MINE tool by adding more features and improving its efficiency in extracting and analyzing data from software repositories that are built on AUTOSAR.
- Adding Git plugin to make the ARXML retrieval process easier and user-friendly.
- Exploring metrics for validating and measuring the effectiveness of the tool.
- Exploring the potential of data visualization techniques.
- Conducting extensive testing and validation of AR-MINE.

References

- [1] “AUTOSAR.” [Online]. Available: <https://www.autosar.org/standards/classic-platform>
- [2] “AUTOSAR Software components and Compositions - MATLAB Simulink.” [Online]. Available: <https://www.mathworks.com/help/autosar/ug/autosar-software-components-and-compositions.html>
- [3] “Basic Concepts — AUTOSAR 0.4.0 documentation.” [Online]. Available: <https://autosar.readthedocs.io/en/latest/basics.html>
- [4] “Configure AUTOSAR Runnables and Events - MATLAB Simulink - MathWorks Nordic.” [Online]. Available: <https://se.mathworks.com/help/autosar/ug/configure-autosar-runnables-and-events.html>
- [5] “Ports — AUTOSAR 0.4.0 documentation.” [Online]. Available: https://autosar.readthedocs.io/en/latest/autosar4_guide/ports.html
- [6] Alpha, “AUTOSAR Basics Archives | autosartutorials.com.” [Online]. Available: <https://autosartutorials.com/category/autosar-basics/>
- [7] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, “Identifying, categorizing and mitigating threats to validity in software engineering secondary studies,” *Information and Software Technology*, vol. 106, pp. 201–230, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584918302106>
- [8] F. Arcelli Fontana, M. Rolla, and M. Zanoni, “Capturing software evolution and change through code repository smells,” in *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*, T. Dingsøy, N. B. Moe, R. Tonelli, S. Counsell, C. Gencel, and K. Petersen, Eds. Cham: Springer International Publishing, 2014, pp. 148–165.
- [9] F. Bantelay, M. B. Zanjani, and H. Kagdi, “Comparing and combining evolutionary couplings from interactions and commits,” in *2013 20th Working Conference on Reverse Engineering (WCRE)*, 2013, pp. 311–320.
- [10] O. Baysal and A. J. Malton, “Correlating social interactions to release history during software evolution,” in *Fourth International Workshop on Mining Software Repositories (MSR’07:ICSE Workshops 2007)*, 2007, pp. 7–7.
- [11] K. Chaturvedi, V. Sing, and P. Singh, “Tools in mining software repositories,” in *2013 13th International Conference on Computational Science and Its Applications*, 2013, pp. 89–98.

- [12] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, *Analysing Software Repositories to Understand Software Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 37–67. [Online]. Available: https://doi.org/10.1007/978-3-540-76440-3_3
- [13] S. Dersten, J. Axelsson, and J. Fröberg, “Effect analysis of the introduction of autosar: A systematic literature review,” 08 2011, pp. 239–246.
- [14] D. Durisic, M. Staron, and M. Tichy, “Arca – automated analysis of autosar meta-model changes,” in *2015 IEEE/ACM 7th International Workshop on Modeling in Software Engineering*, 2015, pp. 30–35.
- [15] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, “Boa: Ultra-large-scale software repository and source-code mining,” *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 1, dec 2015. [Online]. Available: <https://doi.org/10.1145/2803171>
- [16] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, “Fine-grained and accurate source code differencing,” *ASE 2014 - Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 09 2014.
- [17] S. Fürst and M. Bechter, “Autosar for connected and autonomous vehicles: The autosar adaptive platform,” in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, 2016, pp. 215–217.
- [18] H. Gall, B. Fluri, and M. Pinzger, “Change analysis with evolizer and changedistiller,” *IEEE Software*, vol. 26, pp. 26–33, 02 2009.
- [19] M. J. Grant and A. Booth, “A typology of reviews: an analysis of 14 review types and associated methodologies,” *Health Information and Libraries Journal*, vol. 26, no. 2, pp. 91–108, 5 2009. [Online]. Available: <https://doi.org/10.1111/j.1471-1842.2009.00848.x>
- [20] A. E. Hassan, “The road ahead for mining software repositories,” in *2008 Frontiers of Software Maintenance*, 2008, pp. 48–57.
- [21] T. Hermans, P. Ramaekers, J. Denil, P. D. Meulenaere, and J. Anthonis, “Incorporation of autosar in an embedded systems development process: A case study,” in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2011, pp. 247–250.
- [22] —, “Incorporation of autosar in an embedded systems development process: A case study,” in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2011, pp. 247–250.
- [23] A. Hevner, A. R. S. March, S. T. Park, J. Park, Ram, and Sudha, “Design science in information systems research,” *Management Information Systems Quarterly*, vol. 28, pp. 75–, 03 2004.
- [24] M. Janke and P. Mäder, “Graph based mining of code change patterns from version control commits,” *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 848–863, 2022.

- [25] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, “Do faster releases improve software quality? an empirical case study of mozilla firefox,” in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 179–188.
- [26] M. Martinez and M. Monperrus, “Coming: A tool for mining change pattern instances from git commits,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 79–82.
- [27] S. Martínez-Fernández, C. P. Ayala, X. Franch, and E. Y. Nakagawa, “A survey on the benefits and drawbacks of autosar,” in *Proceedings of the First International Workshop on Automotive Software Architecture*, ser. WASA ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 19–26. [Online]. Available: <https://doi.org/10.1145/2752489.2752493>
- [28] D. K. Nilsson, P. H. Phung, and U. E. Larson, “Vehicle ecu classification based on safety-security characteristics,” in *IET Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members’ Conference*, 2008, pp. 1–7.
- [29] P. Offermann, O. Levina, M. Schönherr, and U. Bub, “Outline of a design science research process,” in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, ser. DESRIST ’09. New York, NY, USA: Association for Computing Machinery, 2009. [Online]. Available: <https://doi.org/10.1145/1555619.1555629>
- [30] P. Offermann, O. Levina, M. Schönherr, and U. Bub, “Outline of a design science research process,” 01 2009.
- [31] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 1st ed. Hoboken: Wiley, 2012.
- [32] T. Siddiqui and A. Ahmad, “Data mining tools and techniques for mining software repositories: A systematic review,” in *Big Data Analytics*, V. B. Aggarwal, V. Bhatnagar, and D. K. Mishra, Eds. Singapore: Springer Singapore, 2018, pp. 717–726.
- [33] P. Sivakumar, B. V. Kumar, and R. S. Devi, *Software Engineering for Automotive Systems: Principles and Applications*. CRC Press, 2022.
- [34] U. Sreeram, “Automated Generation and Integration of AUTOSAR ECU Configurations.” [Online]. Available: <https://scholar.uwindsor.ca/etd/8149>
- [35] —, “Automated Generation and Integration of AUTOSAR ECU Configurations.” [Online]. Available: <https://scholar.uwindsor.ca/etd/8149>
- [36] M. C. Tremblay, A. R. Hevner, and D. J. Berndt, “Focus groups for artifact refinement and evaluation in design research,” *Commun. Assoc. Inf. Syst.*, vol. 26, p. 27, 2010. [Online]. Available: <https://api.semanticscholar.org/CorpusID:9790612>
- [37] C. Wohlin and P. Runeson, “Guiding the selection of research methodology in industry–academia collaboration in software engineering,” *Information*

- and Software Technology*, vol. 140, p. 106678, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584921001361>
- [38] Y. Zhang and D. Sheth, “Mining software repositories for model-driven development,” *IEEE Software*, vol. 23, no. 1, pp. 82–90, 2006.

Appendix A

Supplemental Information

Changes in 5f0f724_PORTS with respect to 6cde721_PORTS

	_port	runnables	Serial Num
0	one	Run_DiagMgrFast_Init	
1	one	Run_DiagMgrFast_Step	
2	one	Run_EmDcBst_Init	
3	one	Run_EmDcBst_Step	
4	one	Run_IvtrCtrlActvDcha_Init	
5	one	Run_IvtrCtrlActvDcha_Step	
6	one	Run_IvtrCtrlCoolgReq_Init	
7	one	Run_IvtrCtrlCoolgReq_Step	
8	one	Run_IvtrCtrlFltMgr_Init	
9	one	Run_IvtrCtrlFltMgr_Step	

Changes in T_6cde721_PORTS with respect to T_5f0f724_PORTS

	parent	r_port	p_port	runnables	Se
45		sVccEm	None	None	
46		sVccEm	None	None	
47		sVccEm	None	None	
48		sVccEm	None	None	
49		sVccEm	None	None	
50		sVccEm	None	None	
51		sVccEm	None	None	
52		sVccEm	None	None	
53		sVccLirr	None	None	
54		yVccEm	None	None	

Figure A.1: GUI

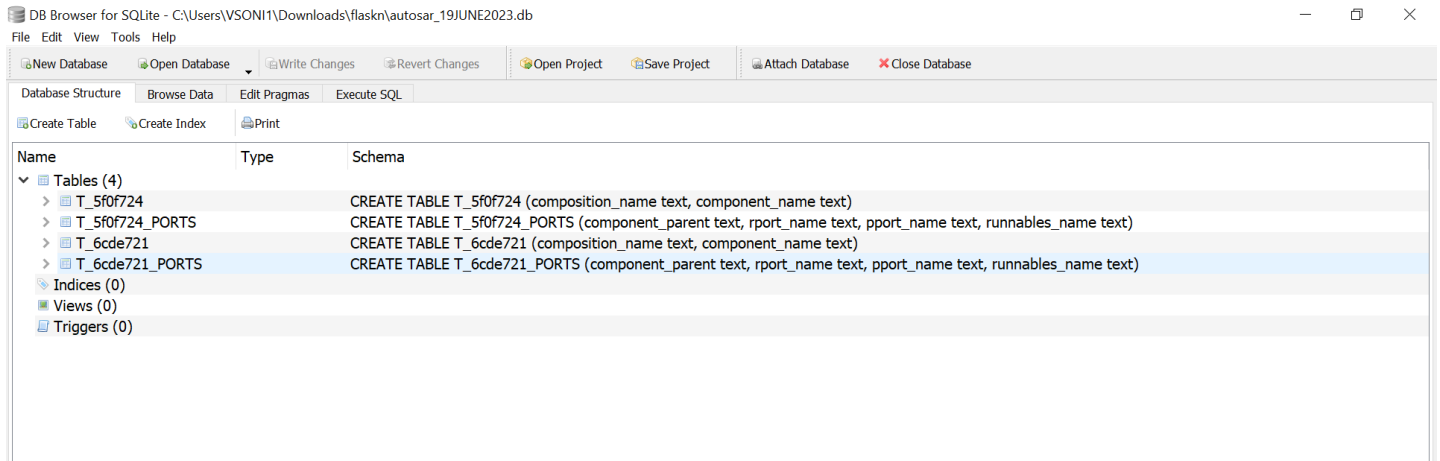


Figure A.2: Data Overview

Appendix A

Interview Guide

Introduction

S.No	Questions
1	What is your name and designation?
2	What is your regular work at Volvo cars?

Understanding

S.No	Questions
3	Is it important for you to know the growth/progress of a project?
4	How do you currently track and measure the impact of growth initiatives on the project? How can this be improved?

Features

S.No	Questions
5	What are the project features in the software repositories that you are interested in, and why?
6	What metrics do you find most valuable when using a software repository to know the growth of the project, and why?
7	Considering PESW git repos that store Autosar file artifacts, what are the most important features that you would like to extract specifically from ARXML files to understand the current-status, growth-potential, and maturity of the project?

FutureActions

S.No	Questions
8	How would you potentially use the extracted information?
9	What are the action points and potential future usage of the information/feature extracted from the repo?

Table A.1: Interview Questionnaire

