Universiteit Utrecht

Faculty of Science

Exploring Change Impact Analysis in Low-Code Development Platforms

Master Thesis

Niels van der Wal
n.e.wal@students.uu.nl
5762499

Business Informatics

Supervisors:

dr. S. Jansen. First Supervisor
Utrecht University

dr. F.J. Castor de Lima Filho. Second Supervisor
Utrecht University

dr. M. Overeem. External Supervisor
AFAS Software

December 10, 2022

Acknowledgements

I want to thank my supervisors Slinger Jansen, Fernando Castor Lima de Filho for their assistance during writing my thesis. I would like to thank Michiel Overeem for his many hours of discussions and assistance in the last months in which I learned a lot of this domain. Furthermore I would like to thank AFAS for letting me do my thesis project at their company. I was able to effectively work on my project and get a better understanding of how LCDPs work. At last I want to thank my parents, friends and my girlfriend for supporting me in the past year.

Abstract

Low-Code Development Platforms (LCDPs) are being increasingly used in various domains. These platforms accelerate the development of applications by letting developers build models in a higher abstraction. They also provide deployment and life-cycle management functionality. As these platforms evolve, problems can arise. For example, when metamodels and models do not conform to each other, the built software will malfunction, and users of the application might not have the desired interactions. Therefore the right decisions need to be taken. Change Impact Analysis (CIA) can be done in order to assist developers of these platforms to support the evolution of the platform. There is not much known about CIA in the context of LCDPs. Therefore we conduct a literature review on the currently available approaches. We found that almost no publications mention CIA in an LCDP context. There are however various papers that present interesting directions for CIA within Model-Driven Engineering which could be applied within LCDPs. Some of these techniques can be used for general-purpose modeling languages such as UML, but some are only applicable to domain-specific languages. These approaches generally made use of techniques such as explicit rules and traceability in order to analyze changes. There are also several other techniques that are more suitable for more advanced approaches. Recommendations are given about how an LCDP supplier should build up their CIA for their platform. With this knowledge, several directions are presented which researchers can explore to improve this domain. Some of these gather more grounding of the current approaches, provide users with improved feedback, or make their approach more correct and complete. At last, recommendations are given to bring the found approaches closer to the LCDP domain by suggesting case studies with other LCDPs.

Contents

1 Introduction

Low-Code Development Platforms (LCDPs) are a recent type of software development platforms (Di Ruscio et al., 2021). These LCDPs let developers build models that are transformed into applications. Through the high abstraction used in these platforms, developers using these platforms can focus on the business logic of the application instead of spending time on building the underlying infrastructure that is needed to support the application (Sahay et al., 2020). Some examples of low-code techniques that can help with building these applications are template-based frameworks or the creation of workflows using connectors (Cabot, 2020). There has been increased interest in these platforms because they enable a large group of non-coding developers to build applications without the need of writing a significant amount of code. In their evaluation of Low-Code Development Platforms, research firm Vincent et al. (2019) claims that LCDPs will be used in over 50% of all medium to large enterprises by 2023. This means that the software development industry cannot ignore this technology. In these platforms, business logic, workflows, or additional input fields can be added without the need for code. This is especially useful for companies without an IT department. There are a growing number of domains in which these platforms can be used. For example the manufacturing industry (Sanchis et al., 2020; Waszkowski, 2019) and Internet of Things (Ihirwe et al., 2020). In addition to companies, individuals can also make use of LCDPs to build applications. Examples are WordPress and Shopify (Lethbridge, 2021; Dushnitsky and Stroube, 2021) that make it easy to build websites and web shops without writing codes.

Throughout an LCDP, changes in different parts of the system by one group of professionals can lead to inconvenience for others. This has historically been called the Ripple Effect (Yau et al., 1978; Arvanitou et al., 2015). For example, when a field is removed in the model, a function in another part of the system depending on that field will malfunction. These kinds of changes need to be actively communicated between development teams. Unanticipated changes can cause various problems with respect to the functionality and stability of the platform. It also takes a lot of valuable time to trace back the changes that caused these interactions. Therefore, measures should be taken to determine what impact changes have on an evolving system. This is called *Change Impact Analysis* (CIA). According to Bohner and Arnold (1996), CIA is "the process of identifying the potential consequences of a change or estimate what needs to be modified to accomplish a change".

By performing CIA, professionals involved in the development, maintenance, and use of LCDPs can make better engineering decisions that ensure the quality of the platform and applications developed using the platform. The use of LCDP will become more widespread in the coming years (Vincent et al., 2019). It will be used in more domains and applications are becoming more complex. This means more groups of professionals will be developing, maintaining, and using these platforms. To keep these platforms available and upgradeable, the industry can use an overview of available approaches for Change Impact Analysis in their platforms.

2 Problem statement

In traditional software development projects, developers can do CIA to see what impact their code changes have on other parts of the system. Various techniques and tools are available for this purpose (Li et al., 2013; Lehnert, 2011a). These techniques are for example mining software repositories, doing dependency analysis, and measuring coupling between entities in an application. Most of these techniques have been developed initially for code-based software development.

There is no clear overview of which approaches can be applied to provide CIA for LCDPs. The industry and research field can be improved when more information on applicable techniques and tools are known. Therefore, the current knowledge of these topics should be synthesized.

There has been one research project in which CIA has been researched in the context of LCDPs (Overeem and Jansen, 2021). This project resulted in a framework to discuss CIA in LCDPs. The authors developed a framework and applied that framework to their platform to show how CIA is implemented in the development of this platform and an application that was built using that platform. This gave an overview of the used techniques and practices that were used for this platform. These were, for example, using message dependency graphs, traceability links, and analysis of model difference results. Since only one platform was used as a case study to validate the framework, it is currently unknown which other applications of CIA exist in the research community and industry.
To get a better understanding of the research progress on this topic, a Systematic Literature Review will be conducted. The different approaches that are presented in the literature are categorized and presented as solutions for CIA problems. Additionally, the relevant beneficiaries of these solutions are presented.
The aim of conducting this SLR will be to create an overview of the currently available approaches to facilitate CIA in LCDPs. Therefore, we will collect available approaches and find which are applicable. The output of the Systematic Literature Review will present gaps in the literature from which research directions can be explored. This can facilitate a starting point for new research in this field. Additionally, the output of this SLR will present a set of recommendations for developers of LCDPs, concerning the techniques they can use to facilitate impact analysis of their platform. Working with these recommendations can assist the relevant stakeholders of an LCDP to make better choices and develop a timeline of how they want to evolve their platform over time. This will hopefully lead to more stable platforms and the development of better applications.

3 Literature Background

3.1 Low-Code Development Platforms

According to Sahay et al. (2020) LCDPs are development platforms that make use of visual interfaces to configure the data schema of the application that is developed. In these interfaces, a developer can create entities, establish relationships between entities, define constraints, and create dependencies between these entities. These platforms are provided in the cloud. Users of these platforms have a subscription-based on Platform-as-a-Service (PaaS) model to get access to the platform and start developing and deploying applications. The main goal of these platforms is to reduce the amount of hand-written code and decrease the time for the development and maintenance of applications. Removing these routine tasks can save developers time that can be used for other tasks. This improves the speed of development. The developers make these applications by building models. Historically models have been used for several activities during the development of software. An example is doing software maintenance (Fernández-Sáez et al., 2018). Modeling languages such as UML make it possible to create an overview of software design. The graphical notation employed in these diagrams can achieve more uniformity in documentation than using textual documentation. Developers can communicate their thoughts and ideas before building the application.
According to Waszkowski (2019), these platforms combine several approaches: (1) Model-driven software development, (2) Rapid application development, (3) Automatic code generation, and (4) Visual programming. These platforms are getting increasingly popular among major IT players. Due to requiring less development experience, people without traditional software development (primarily programming) experience can develop software easier than before these platforms were introduced. As these platforms often still allow developers to run their custom code in the platforms, it is still important that the use of code does not result in complecations (Lethbridge, 2021). Therefore, it is important that features exist that improve the maintainability and understandability of these applications. Lethbridge has the assumption that developers often use too much custom code. Developers of LCDPs should therefore keep in mind that many end-users will continue doing that and adequate support for these practices should be employed in these platforms. Examples would be enabling documentability, automated testing, and improving re-use and collaboration when using these platforms. Another idea is to create multi-vendor open standards for low-code languages.
There is a wide range of LCDPs with each their own features and qualities. Therefore it is important to choose a platform that suits the goals of the end-users. Farshidi et al. (2021) developed a model to improve decision making when settling for an LCDP. It was validated through four industry case studies and was proven to support decision makers to make better decisions.

3.1.1 Compared to Model-Driven technologies

By several authors low-code is seen as a synonym for model-driven development (Cabot, 2020; Bock and Frank, 2021). Cabot (2020) sees it as a restrictive view of model-driven development (Atkinson and Kuhne, 2003), in which developers only target a concrete type of application. These applications are data-intensive applications that are accessible using web browsers and mobile phone applications. According to Cabot, there are some small differences. In LCDPs, platforms make use of a fixed-language solution. The language itself is not exposed to the user of the platform and can't be changed. In MDD solutions, there is some flexibility for languages to be defined and adapted. There are, for example, several standards, such as UML and BPMN, that can be used in model-driven approaches. This means that there is some flexibility available for adopters of model-driven adopters, but not for those in low-code. Two critical papers seek to determine what differences there are between LCDPs and related technologies such as model-driven development. One was written by Di Ruscio et al. (2022) and the other by Bock and Frank (2021).

Di Ruscio et al. (2022) researched how Low-Code Development and Model-Driven Engineering (MDE) are related to each other. According to their paper, MDE includes a wide range of software paradigms focusing on the use of models as primary artifacts during the development cycle of applications. In MDE, models are used for various processes such as testing, simulating, verifying, and generating code for the system. Although the models do not always have to be used for code generation. However, this is the case for low-code development.
Di Ruscio et al. (2022) state that the goal of MDE is: "to increase productivity by automating different steps in software development employing models while augmenting the overall quality". This does not mean that Model-Driven approaches always aim at reducing code. MDE makes use of domain-specific languages (DSLs) for their specific domain. This can be more effective than writing code in industry programming languages because descriptions using a DSL can be made more intentional and requires fewer workarounds to cater the code to the domain. This results in easier creation, verification, and maintenance of models. This shares many similarities with LCDPs as they also aim to increase productivity with the use of models instead of writing code. On the contrary to MDE, LCDPs actively work on reducing complexity for end-users (citizen developers) during the installation and operation of the platform. LCDP developers generally do not have to install the modeling software and are assisted by the platform to host, analyze, and allocate resources for the developed applications. Often this is done by working in cloud-based development environments. In MDE reducing this type of complexity of lifecycle management is not one of the key points.

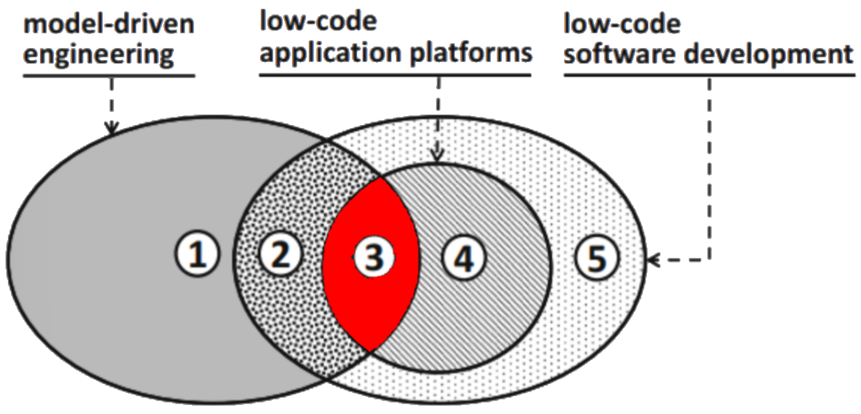

Figure 1: A Venn diagram showing commonalities between model-driven and low-code development approaches (Di Ruscio et al., 2022).

As there are commonalities between both MDE and LCDPs, it is possible to show which approaches are entirely model-driven and which are low-code (Di Ruscio et al., 2021). There are also approaches that make use of a combination. Figure 1 shows the Venn diagram developed by the authors. It shows five categories of approaches that can fall under MDE or low-code Platforms. Areas 2 and 3 are part of both MDE and Low Code Software Development. The approaches in region 1 of this diagram are not focused on reducing the amount of code used to implement the system, whereas the approaches in region 5 aim to reduce the amount of code used to implement a system. These platforms can't be considered model-driven by Di Ruscio et al.. In this region, approaches contain platforms that facilitate the development of applications with reduced code. These platforms also provide users with lifecycle management and deployment facilities. These platforms do not use models that conform to defined language/metamodels. Instead, a relational database or schemaless documents such as XML or JSON are used.
In region 3 (the area colored red), the LCDPs that are discussed in this project are located. The platforms in this area of the diagram use models to facilitate the development of software applications. They do this while reducing the required code and by offering deployment and lifecycle management facilities. Examples of these platforms are Mendix, OutSystems, and PegaSystems.
The main differences between the model-driven and low-code approaches are summarized in Table 1.

	Model-driven approaches	**Low Code**
Models used for	Not always to generate applications. They are used to improve development.	Always to generate applications.
Access	Generally on-premise.	Generally cloud-based platform accessed using a browser.
End-users	Typically professional software developers.	Developer with generally no software development background.
Domain	Applied in many domains. Can target very technical and domain specific areas.	Mostly business domain. But more domains specific platforms are introduced.

Table 1: The main differences of the Model-Driven and Low-Code approaches based on (Di Ruscio et al., 2022).

There are more differences between Model-Driven and Low-Code approaches. Di Ruscio et al. (2022) state that most MDE solutions are based on desktop solutions. Software such as the Eclipse Modelling Framework and MetaEdit+ do not run in the cloud. While most LCDPs are cloud-based. Since these platforms are generally cloud-based, it makes it easier for new users to start working with these platforms. There is no long installation and customization procedure required to start building and deploying applications.
As mentioned earlier, the target users for LCDPs are the so-called "citizen developers". They generally lack knowledge of coding and other software development skills. Therefore, these platforms need to be catered towards those with a non-technical background. There are

however exceptions where these platforms are used by software architects and programmers. MDE solutions are generally used by professional software developers during development processes. This makes them less useful for the purpose for which LCDPs have been introduced.
Model-Driven Engineering solutions have been used for various domains. The domains can be mostly software focused. Examples are chatbots or IoT appplications. But these domains can also be focused on physical systems, which requires domain experts for the specific technology. This could be for example automotive or aviation engineering. The optimized Domain-Specific Languages can be adapted specifically for the domain which improves engagement with domain experts. Low-code tools have been mostly used to create business applications to automate business processes. There are proposals for platforms specifically for other domains than business applications, but they are not widespread in the industry.

Bock and Frank (2021) analyzed a similar problem. Although their approach was more systematic and led to a model in which they presented which features were present in LCDPs and how frequent they were found while examining a set of LCDPs. From initial research on definitions for LCDPs, they suggest two conclusions when looking at the characterization of LCDPs. (1) LCDPs are intended to help achieve objectives that have been the core of the information systems research domain. Among these are increasing productivity and reducing development and maintenance costs. LCDPs also improve the organization's capabilities to adapt systems to requirements that can change during development. In addition, they also help empower users. (2) It is unclear in what way LCDPs are different from existing software development facilities.
The authors studied ten LCDPs during a previous project (Frank et al., 2021) which led to a set of features that can be identified within these LCDPs. The identified characteristics were classified into five perspectives. These were the *Static*, *Functional*, *Dynamic*, and *Interaction*. And at last, they had a category for *other aspects* of an LCDP. For each feature in these categories, a score was given based on the frequency of being included in the platforms studied. The commonly shared features were mainly found in the *Static perspective*. Examples were the features about having a *data modeling component*, having *components for data structure specification* and having an *internal database management system*. The feature of providing a *GUI designer* was also found in the studied platforms. In these GUI designers, developers can create graphical user interfaces to input data that can be integrated with other implementation artifacts. Less commonly included features are *domain-specific reference functions* and *reusable artifacts*. Only one of the platforms provided the reuse of specific ready-made functions from the vendor. Other systems had catalogs with reusable functions and examples that were very generic or rather limited.
Their observations lead to an assessment of the low-code trend. The outcome of this study (Bock and Frank, 2021) was as follows:

- *LCDPs integrate various classical development components in one environment*: The authors say that an LCDP is an environment that includes several already known components. This results in less need for switching between systems and having to put less effort into integrating the artifacts to other components outside of the environment.

- *Reuse is addressed at a generic architectural level, not at a domain-specific level*: the studied platforms rarely offer artifacts that can be reused at a domain-specific level. The parts that can be used are often very generic and not suitable for specific contexts.

- *Productivity gains mainly ensue from reducing the efforts of routine tasks*: A lot of time and effort is saved by providing good integrations within the system. There is less need for synchronizing the artifacts that used to be produced by separate components.

- *No new technology*: No new technologies are used within LCDPs. They are merely a combination of components. It is not radically new or innovative. Some LCDPs have been used for years but haven't been called "LCDPs". They adopted the name LCDP to make it more interesting for customers.

- *Conceptual modeling is not at the core of marketing, but at the core of the platforms*: Conceptual modeling components in the platform are the most important components. They reduce the need for traditional code. This matches well with MDD. The modeling languages that are used for this purpose are too simplistic for the authors. They do not keep up with the state of the art in conceptual modeling research.

- *Incomplete account of related research*: The research field of this domain draws inspiration from the field of Model-Driven Development. Although exact technical concepts and procedures are not usually adopted. Some of those constructs such as *visual programming*, *weaving reusable services into customized services*, and *on-the-fly computing* have been investigated in this context. The authors think that some of these constructs have been ignored during their research but could be a welcome addition to this field. For example, using *reference models*, and, working on *domain specific modelling languages.*

- *Risk of lock-in effects*: When businesses dedicate themselves to one platform, they depend on that platform to offer them support. When a platform decides to stop supporting its platform. There are almost no ways to export built applications and other artifacts to other platforms.

Comparison of both papers Both articles have a different outlook on the domain of LCDPs. Di Ruscio et al. (2022) see potential in LCDPs and are under the impression that this is a new type of platform that combines Model-Driven and low-code approaches to have the best of both worlds. Low-code and Model-Driven Engineering are not the same and use different technologies and philosophies. In both approaches, software engineering is improved by increasing abstraction and hiding details regarding the implementation of the software. In model-driven approaches, it is not always the goal to reduce the amount of user code. And low-code does not mean that something is model-driven. These are areas 4 and 5 in Figure 1. Examples are software packages such as WordPress and Shopify (in area 4). When using WordPress and Shopify users can build websites and webshops, respectively without writing a significant amount of code (Lethbridge, 2021). But unlike model-driven approaches, they do not use models conforming to meta models to build these websites. Instead, these services solely provide deployment and lifecycle development of the websites that they create. There are also low-code approaches which do not use models and don't have built in functionality for

deployment and lifecycle management. These are found in area 5 of aforementioned figure. Di Ruscio et al. mention database-schema-driven generators as examples for these approaches. Bock and Frank (2021) however argue that the components of LCDPs are no innovations. LCDPs reuse components from older technologies but do a lot of effort to integrate these components into one environment. This reduces the amount of time users must spend on routine tasks. However, this can only be achieved if the used framework can be used for developing that kind of application and fulfills other technical and economical conditions. In the future, these conditions can be carefully assessed in research. They see the potential for the revival of conceptual modeling, which is one of the central LCDP components. Cabot (2020) shares the ideas of Bock and Frank. He says that challenges in model-driven can be transferred to low-code as the technologies these types of approach used are the same. Low-code however has the potential for bringing modeling to more domains and attracting more experts in the domain.
Both studies have their concerns about the generality of LCDPs. They are often too general and do not cater to the needs of a specific industry. Only general business and communication concepts are provided by these platforms as reusable artifacts. Di Ruscio et al. (2022) think that domain specific LCDPs could be developed to cater to these requirements from the industry. In there they could for example provide more specific data reference models for that domain. The study by Bock and Frank (2021) sees this as a welcome addition to LCDPs because it can improve the reusability of artifacts.
Both studies are also in agreement that there is not enough reusability, import, and interpretation of artifacts possible between different platforms. If there are no good approaches to transfer artifacts, this results in a vendor lock-in for developers of those platforms. This thought is shared by Sahay et al. (2020). They also think that not enough effort is taken to support the reusability of already developed low-code artifacts. In addition, more generic mechanisms should be developed that can enable interoperability between LCDPs. Di Ruscio et al. (2022) mention that the MDE community supports this by providing standards. For example they have standards for modelling languages (UML, BPMN) and meta modelling languages (MOF, Ecore). They mention that more research should be done if using a set of standards would improve interoperability and therefore prevent vendor lock-in.

Thoughts on the MDE vs LCDP discussion As mentioned in the previous section, various authors voiced their ideas on the differences and similarities between the terms MDE and LCDP. Authors such as Di Ruscio et al. (2021) think that LCDPs are something else and should be treated as such. While Bock and Frank (2021) see both terms as the same concepts.
We personally think that these terms can not be interchangeably used due to the technical differences. While many concepts within LCDPs are certainly model-driven. There are still differences in terms of abstraction. This makes it difficult to generalize MDE solutions to the LCDP domain. When looking at available information on mature LCDPs (Mendix and Outsystems) we noticed that the analysis and validations of models were much more advanced than were seen within currently available approaches (Mendix, 2021; OutSystems, n.d.). As we do not have knowledge of the underlying architecture and supporting technology, we still need to find a way to connect the available MDE CIA approaches to LCDPs. Therefore, it is important to closely look at currently available MDE CIA approaches and look whether

they have potential. Additionally performing case studies with mature LCDPs could validate existing thoughts and ideas on applicable approaches for this purpose.

3.2 Change Impact Analysis

According to Bohner and Arnold (1996), Change Impact Analysis (CIA) is “the process of identifying the potential consequences of a change or estimate what needs to be modified to accomplish a change”. Rolfsnes et al. (2016) specify this by mentioning that CIA aims to find the artifacts that are potentially affected by a change. With the knowledge gained, direct feedback can be given to the developer, tests can be performed, and prioritization can be done. As Low-Code Development Platforms (LCDPs) make use of various models, CIA should be applicable in this domain. A common way to do CIA is to perform static or dynamic dependence analysis (Rolfsnes et al., 2016). This identifies methods that call a changed method. It is a relatively safe approach that finds potentially affected artifacts when a change happens.
Historically, the effect of changes happening in one part of the system, resulting in problems in the other parts of the system, has been called the “ripple effect” (Yau et al., 1978). This effect has been researched during various projects. As a result of these projects, tools have been developed such as the “Ripple Effect and Stability Tool” to compute this effect for programs written in C (Black, 2001). The probability that this effect occurs can be measured with the Ripple Effect Measure (Arvanitou et al., 2015).
CIA is sometimes also referred to as *Change Propagation* (Hassan and Holt, 2004). Although this term seems to be more common within engineering communities such as mechanical design. Hassan and Holt explain Change Propagation as the process of “Ensuring other entities in a system are updated to be consistent when modifications are done to a system”.
Several reviews of Software Change Impact Analysis have been done (Lehnert, 2011a; Li et al., 2013). These reviews were focused on Change Impact Analysis in traditional software development. The review by Lehnert (2011a) summarized 150 approaches and classified them. The found approaches were classified using a set of criteria. 65% of the approaches found appear to focus on code changes. 13% of the approaches combine artifacts to do analysis. A taxonomy (Lehnert, 2011b) for these approaches was also introduced. With this taxonomy, a reviewer can improve their classification and comparison of Change Impact Analysis approaches. Several criteria found in the literature were included in the taxonomy. Some of these criteria are *Tool support*, *Granularity of entities*, *Utilized techniques*, and *Style of Analysis*.
Li et al. (2013) found 23 different code-based CIA techniques and evaluated them. They identified four perspectives in which these 23 techniques could be grouped. These perspectives are (1) Software repositories mining (2) Coupling measurement (3) Execution information collection and (4) Traditional dependency analysis. The different techniques could also be grouped into the way their code is analyzed and whether the analysis would give a list of priority. Another categorization that is mentioned is the difference between static and dynamic CIA. Static CIA techniques look at possible behavior and inputs of the application. This results in less precision. Dynamic CIA techniques look at specific inputs and do analysis on information collected during the execution of the program. Examples of information that can be analysed after execution of the program are execution traces, coverage information,

and execution relation information. When doing this, the impact set can be calculated.
In LCDP there are many layers that are affected when a change is made in another layer. This is not explicitly found in the works of Li et al. (2013) and Lehnert (2011a). This makes these reviews less applicable in the domain of low-code development and related technologies. They can however be used as a source of inspiration. Some of the criteria that were used in these works can be transfered to the model-driven and low-code domain.
There have been studies that researched Change Impact Analysis within Model-Driven Development Systems (Tekinerdogan and Er, 2009). This study makes use of *Evolution Scenario Templates* which can be used to describe evolution scenarios of models. With the templates, concrete evolution scenarios can be presented using evolution transformation patterns. They define what the impact required for this scenario is. It could therefore be useful to use these templates.
A master thesis project was conducted on Change Impact Analysis in Model-Driven Software Engineering Ecosystems (Jongeling, 2016). In this project, a tool for impact analysis in MDSE ecosystems was implemented and its performance was evaluated. By using this tool, the author achieved to correctly identify implicit and explicit relations between metamodels and model transformations (and their included elements). The tool can also predict that additional changes are caused by a metamodel change. This is unfortunately not very precise and it's only applicable within a limited part of this ecosystem.
In Overeem and Jansen (2021) the authors mention several model-driven CIA approaches that have been used in an industry LCDP. One of those techniques was to use traceability. Galvao and Goknil (2007) present a survey with traceability approaches in Model-Driven Engineering (MDE). 12 approaches were found, and they were compared based on the given criteria for traceability. One of the criteria was providing Change Impact Analysis mechanisms. Out of these approaches, Five can determine the effect of a change on the entire system. They individual approaches had different ways of implementing CIA in their traceability approach.
Another approach that is mentioned to improve CIA is to make use of recommender systems (Overeem and Jansen, 2021). A recommender system could use rules to analyze change results. By doing this, alternative solutions for the model can be provided that have lower impact on the run-time. This however requires that the system can link model changes to run-time model changes. Almonte et al. (2021) looked at various recommender systems that can be used for different parts of MDE solutions. Examples are metamodels, models and transformations. These are important artifacts that could be affected after a change. The approaches that were found were used for model-driven development. However, the authors see many opportunities for these approaches when applied in the low-code domain. Recommender systems make use of information from best practices and provides users the recommendations that can be given based on these practices. LCDPs make use of repositories that store data of the models. With this information, recommender systems have many sources to improve their effectiveness. Therefore, the authors see much potential in the use of recommender systems in LCDPs.

Beneficiaries of Low-Code Development Platforms In their project, Overeem and Jansen (2021) identified multiple professionals who are stakeholders when improving CIA in LCDPs. There were *Citizen developers* who made use of the platform to build applications. These developers generally lack traditional software development skills but do know the do-

main for which they are developing an application. This could be, for example, accounting and business applications. *Citizen developers* make changes to a model using a model designer. These changes can affect customer data. Therefore, the *citizen developers* need to make sure that no unintended operations take place when making changes that impact customer data. For them, it would be a valuable addition to their development experience when they receive feedback on their models. *Platform developers* are involved in building the platform that citizen developers use to build and change their models. For this, they develop and maintain multiple subsystems within the platform. With proper feedback, they can improve the development and maintenance experience for citizen developers and operations engineers, respectively. By analyzing changes in the model, platform developers can identify popular and neglected features in the platform. This will allow them to prioritize preferred additions to the metamodel and designer application for the citizen developer. Analysis of the metamodel will give them an overview of how the model designer is impacted after changes. Better decisions on changing the model designer can be made with this information. The last group of professionals involved in the development and operation of applications that are developed using LCDP are the *operations engineers*. According to the authors, there are two groups of these *operations engineers*. The first group of operations engineers are in charge of the upkeep of the model designer, the model transformations, and the runtime of the platform. They are employed within the company that provides an LCDP. With impact observations, these engineers can help them plan upgrades to the platform. There are also operations engineers for companies that make use of LCDPs to develop applications. Their responsibilities are different, as they have to guarantee the upkeep and performance of the applications that are developed with the platform. For them the impact observations can assist them with planning upgrades of the application.
In this paper, the authors did not indicate which of the presented groups profit most from doing CIA. The impression is now that citizen developers who interact with models that impact customer data have the greatest benefit from better CIA in the platform that they are using to build their applications.

3.3 Definitions within research on CIA for LCDPs

Software evolution and maintenance *Software evolution* is a term that has been researched for a long period. Software evolution and maintenance are often seen as something similar. But there are differences between both concepts. Rajlich (2014) sees software lifespan as a staged model which starts with the *initial development* of an application. In this stage the first version of the application is delivered. Following this first *evolution* will take place. The developers will add new features and correct their previous mistakes to make the product more suitable towards the needs of the users. This is the phase most LCDPs are in as they keep adding new features to improve their software. After this stage the *maintenance* stage takes place in which developers do not make significant changes in the software but focus on repairing the software to keep it usable for end-users. Software in this stage is also known as “legacy software”. After these stages, software can be *phased-out* which means service is withdrawn from the developers and finally the software can be *closed-down*.
The International Organisation for Standardization thinks of software maintenance as “the totality of activities required to provide cost-effective support to a software system.” (IEEE,

2006) which is different from the ideas Rajlich (2014) stated about software evolution being a stage of the software lifespan.
Chapin et al. (2001) have more comprehensive definitions of *software evolution* and *software maintenance*. They define *software evolution* as "the application of software maintenance activities and processes that generate a new operational software version with a changed customer-experienced functionality or properties from a prior operational version, where the time period between versions may last from less than a minute to decades, together with the associated quality assurance activities and processes, and with the management of the activities and processes; sometimes used narrowly as a synonym for software maintenance, and sometimes used broadly as the sequence of states and the transitions between them of software from its initial creation to its eventual retirement or abandonment". They define *Software maintenance* as "the deliberate application of activities and processes, whether or not completed, to existing software that modify either the way the software directs hardware of the system, or the way the system (of which the software is a part) contributes to the business of the system's stakeholders, together with the associated quality assurance activities and processes, and with the management of the activities and processes, and often done in the context of software evolution".
The main difference between these two concepts is that maintenance is a deliberate choice, and evolution happens in the course of making changes to the software (for example during maintenance). In the context of this research project, we use the definition by Chapin et al. (2001).
Evolution is a topic that is widely researched within software development. Figueiredo et al. (2008) look at the evolution of Software Product Lines using several aspects. They mention that the evolution brings concerns to software engineers due to the frequent changes that happen during evolution. Examples are the introduction and removal of features, but also the transformation of mandatory features. Mens et al. (2005) mention that co-evolution between different types of software artifacts will be an important challenge in the upcoming years. This is also one of the problems CIA tries to tackle.

LCDP Change There are different locations in which professionals can make LCDP-changes. Overeem and Jansen (2021) mention multiple changes such as model (primary artifacts in LCDPs), metamodel, transformation, run-time model, and platform changes. There are specific CIA approaches for the aforementioned changes that can be made while interacting with an LCDP. Lehnert (2011b) mentions these in his taxonomy as the scope an LCDP approach can focus on. As previously mentioned, for LCDP-Evolution, the *software evolution* definition by Chapin et al. (2001) will be used. As the activity of making changes is highly comparable to *software maintenance*, we will use their definition for the concept of *LCDP-Change* within the metamodel.

Analysis methods Lehnert (2011a) did a review of Software *Change Impact Analysis* activities. Within this project a definition was constructed from the works of Bohner and Arnold (1996). Lehnert defines it as "Identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change". The concept of *Analysis Methods* within the metamodel represents the methods that are concerned with the definition by Lehnert. They are supplemented with concepts from his taxonomy (Lehnert, 2011b)

Impact Within the project by Overeem and Jansen (2021), the concept of *Impact* is being used to define the effects that the changes made by professionals have on the performance of the LCDP. Arnold and Bohner (1993) found in earlier literature that impact has also been defined as "The effect or result of making a change to a system or its software". They define it themselves as "a part determined to be affected, and therefore worthy of inspection".

Feedback In Overeem and Jansen (2021) the authors mention that *impact analysis methods* provide *feedback*. This feedback can assist the involved professionals in the LCDP with an analysis of how the changes they make have an impact on other parts in the system. The professionals can use this feedback to make well supported engineering decisions to improve the platform.

3.4 Systematic Literature Reviews in Software Engineering

In the past twenty years researchers have been discussing the quality of Systematic Literature Review in the information systems domain (Brereton et al., 2007). One of the major criticisms of authors in this field is that researchers did not take inspiration from methods and experience in other domains with more experience in doing SLR. Therefore various researchers aimed to combine best practices from other domains and applied them in the information systems domain. Three attempts and results from doing SLR following these procedures are discussed in this subsection.
Kitchenham (2004) introduced her *Procedures for Performing Systematic Reviews.* This has become one of the major sets of guidelines for conducting a systematic literature review in software engineering research. In her guidelines, she used three existing guidelines for doing research in the medical field and adapted them to apply to the problems in software engineering research.
The various procedures to conduct Systematic Literature Review Kitchenham (2004) have been used in various studies in the domain of software engineering. This can lead to different types of contributions to the scientific community. A common output of an SLR is to provide an overview of the relevant literature found (Hall et al., 2011; Costa et al., 2022). Researchers can combine answers to their research questions into a set of requirements or guidelines (Azadani and Boukerche, 2021; Alhirabi et al., 2021). Another possible artifact created by doing SLR is creating models that comprise the findings in the literature. An example is a maturity model that can be used to find the current stage a software producing business is in (Overeem et al., 2022).
Wohlin (2014) presented the snowball approach. In this approach, a starting set is used from which the reviewer searches forward and backward with the set of references that are included in the starting set papers. By applying inclusion and exclusion criteria, the final set of included papers will be applicable to answer the research question. This approach is suitable when using keywords that yield too many results to manually examine due to noise from general keywords. This can happen in a widely covered research area. With a set of picked starting set papers that fit the topic, snowballing can result in a better set of papers suited to the topic.
The snowballing approach was used in various studies. For example, in the paper by Overeem and Jansen (2017) on generative and interpretive Model-Driven Development. Their system-

atic review of the literature resulted in an overview of which paper favored generation or interpretation for characteristics for software quality.
Okoli (2015) saw other problems concerning SLR in information systems research. Guidelines that were previously published on doing SLR in this domain were generally incomplete. Only several parts of the entire process were explained, while other important aspects were not mentioned in these SLR protocols. Therefore Okoli combined best practices from multiple guides to develop a rigorous standardized methodology for systematic literature review in this domain. For this, he used guides from multiple disciplines such as health sciences and adapted them to the information systems domain. The guide can be used for the entire process of doing SLR, and provides many examples of guides that give proper explanation of an aspect of each step of doing SLR.
To improve the field of software engineering research Ralph et al. (2021) affiliated with ACM SIGSOFT released their set of Empirical Standards. With these standards, researchers can design better studies, fix peer reviews, and educate graduate students. The authors released these standards for different types of research and show what kinds of attributes should be and what other desirable attributes could be included when doing research. Based on previous expert experience, specific standards were developed for research methods such as conducting case studies, questionnaires, and systematic reviews. Throughout this project, the systemic reviews empirical standards, as well as the supplements on Inter-Rater Reliability and Agreement are consulted while working on the proposal, searching the literature, and reporting the results.
In this master thesis research project, the output consists of an overview of the available approaches for Change Impact Analysis in Low-Code Development Platforms which will be complemented with recommendations on which techniques can be used for challenges that exist in the domain for which CIA can provide a solution. Similar guides on how to deal with specific situations have also previously been an outcome of SLR (Marinho et al., 2014; Sancar Gozukara et al., 2022; do Carmo Machado et al., 2014). Problems are elicited and studies mentioning solutions are presented for specific problems and challenges.

3.5 Tool Support for Systematic Literature Reviews

To conduct Systematic Literature Review, a reviewer can make use of a wide range of tools to streamline their reviewing process. Some tools can help manage references and generate citations, coding information, and more tasks that are part of conducting Systematic Literature Review. There have been academic and various commercial initiatives to develop these kinds of tools. The academically developed tools are generally open-source and those developed by commercial software vendors often require subscriptions to make use of the features included in the tools.
During a study, desirable features for SLR tooling were discussed and prioritized (Hassler et al., 2016). It appears that currently available tools do not provide enough functionality for *integrated searching* of papers. In addition, the *collaboration* functionalities that allow SLR teams to work together on one project are not well supported. For integrated tool sets, several features must be provided: (1) A tool should make it possible to collaborate with other authors in a team. (2) Tasks and processes must be automated. (3) It must include data-sharing features. (4) Imported and exported data need to be preserved to access previous

projects. (5) Forward and backward traceability must be present to link goals, actions, and results. This improves the standardization, verification and validation of the project.

Reference management Managing reference is an intensive task when doing it manually. Several reference management tools can be used to track found resources and easily use these references in SLRs. Over the years, these tools have been adapted to work in browsers, where they can easily import details from studies and include them in the list of references. These references can be presented in various reference styles such as APA, MLA, and Chicago. There are many citation management tools available (Ivey and Crum, 2018). Most of them have a similar feature set, as they can be used with browsers and provide integration with text processors such as Microsoft Word and Google Docs. There are however reasons to do or do not choose for a specific tool. Some platforms provide collaborative functionalities while some don't. Of the four tools that were analyzed, three of them required a fee or subscription to access the tool. This could be a reason for reviewers to use a free alternative such as the mentioned tool Zotero that offers similar functionalities.

Tool	**Advantages**	**Disadvantages**	**Collaboration**	**Costs**
EndNote	A large range of output and citation formats are supported.	The costs for using this tool are high.	Library sharing to up to 100 others.	One time fee.
Zotero	All references are stored online and are accessible from different operating systems and browsers. A large range of output and citation formats are supported.	Some features aren't provided out-of-the-box and require external plugins.	Libraries can be shared between authors.	Free (open source).
Citavi	A large range of output and citation formats are supported. Especially for several LaTeX editors.	The costs for using this tool are high.	Team projects can be made in order to collaborate.	Subscription-based access.
Excel and comparable tools	These tools are versatile, and can be used for various actions depending on the set up of the workbook. Can be accessed online.	Requires time and effort to create workbook with the required features that are needed for a project. Several actions need to be done manually. Such as downloading references and placing them in the workbook.	Collaboration is possible using shared workbooks.	Paid or free to use.

Table 2: Tool support for reference management.

EndNote[1] can be used to manage bibliographies and references. Users can store their found references, and the tool can find PDF files for the found references, making it easier to access, read, review, and annotate these papers. The tool works from the cloud and makes it easy for reviewers to continue their work from anywhere. Reviewers can make shared libraries with their peers and track each other's changes. This increases collaboration between authors. The references found can be exported to various output reference styles. To use EndNote, users need to buy the software with a one-time purchase.
Zotero[2] can also be used to manage references. Like EndNote, it can be used to store and

[1]https://endnote.com/
[2]https://www.zotero.org/

organize references. There are integrations for various text editors (including Google Docs) that make it easy to include citations in the text when reporting results. All of the supported data can be synced in the cloud which allows users to switch between devices when needed. The reference libraries created can be shared among multiple users. Zotero is an open-source initiative. Therefore, no subscription or product fees must be paid to use this tool.
Citavi[3] is another tool to manage references when performing SLR. The tool can help a reviewer with database searches from a wide range of sources. Papers in PDF file extension can be annotated inside the software, and integration with text processors such as Microsoft Word and LaTeX is provided to the user.
Excel and similar spreadsheet solutions by other vendors are versatile tools that can be used for various business tasks. It can also be tweaked to be used to perform tasks that support conducting Systematic Literature Review. Excel is especially useful for tracking literature that is collected. Using Excel, a reviewer can assign labels to references and then find them using filters. This makes it easy to keep track of different references that are collected throughout the project. Unfortunately, Excel lacks the functionalities to import references from databases. This can however be done manually or with help from other tools such as Harzing's Publish or Perish[4]. As there are various vendors with cloud-based spreadsheets with support for multiple users contributing to the same workbook, it is possible to easily collaborate with other reviewers by simply sharing a link to a workbook. With a well-designed workbook, references can easily be recorded, and relevant codes can be given to relevant papers. These codes can be found using user-defined filters.

Screening A feature that has been included in several tools is the screening. During the screening activity, a reviewer looks at the search results of their query and rates the titles and abstracts of the found references to include or exclude a paper from the set of papers that are used in the SLR. As SLRs often use database searches with keywords, this can yield many results; reviewers must spend a long time checking these references and indicating whether they match the topic of interest well enough. This process can be optimized to take less time and be more efficient. Therefore, tools have been developed to improve this. When performing keyword searches on "SLR Tooling" AND "Snowballing"", it appears that there are no available tools that simplify the snowballing process. However, it is possible to manually insert reference lists from the paper into some of the screening tools mentioned in this paragraph.
A set of these tools were included in a study that looked at the screening procedure for doing SLR in the healthcare domain (Harrison et al., 2020). Fifteen tools were analyzed based on their features. Since not all desired features were found in these tools, only six tools were included in the final survey which was answered by healthcare researchers. The tools Covidence and Rayyan were recommended as the best solutions for researchers to use. The authors found that these tools aligned well with the user requirements for these types of tools. Other researched tools during the project might lack in usability but compensate for that by including specialist features which could be a valuable addition during specific projects.

[3]https://www.citavi.com/en
[4]https://harzing.com/resources/publish-or-perish

Several industry tools were inspected to see features. The main characteristics, information about collaboration support, and pricing can be found in Table 3.

Screening Tool	**Special features**	**Collaboration**	**Costs**
SLR tool	CrossRef API support. Analysis with PowerBI.	No support for collaboration with other reviewers in a team.	Free
ASReview	Makes use of machine learning to automatically screen references based on a sample set of relevant references.	No support for collaboration with other reviewers in a team.	Free
CADIMA	Uses steps from Kitchenham to systematically do the screening process of found papers	There is support for collaboration with other reviewers.	Free
Covidence	Support for screening abstracts, titles, and full text. Highlighting of keywords. Can use metrics on inclusion and exclusion to generate PRISMA diagrams.	Reviewers can work together on one project. There is also a feature that can help with achieving consensus when doing practical screen	Paid

Table 3: Tool support for screening

Hinderks et al. (2020) present a tool for SLR. The tool assists in creating and managing SLR projects. Features are supported to import search results and manage search results by including and excluding papers. Another interesting feature is that the CrossRef API can be called to update the references to be more complete. The tool makes it easy to screen references based on the title, keywords, and abstract. The reviewer can then decide whether to include or exclude it. All of the choices that were mede by the authors are documented for a good overview of the number of papers that are included or excluded based on specific criteria. These metrics can be used to create a PRISMA diagram.
Researchers of Utrecht University developed a free open-source tool that can assist in systematic reviewing (van de Schoot et al., 2021). Using RefWorks files, this tool **ASReview**[5] can import large sets of references. With these references, reviewers can filter titles and abstracts. The tool supports machine learning to select relevant and irrelevant papers based on these segments of a paper. A reviewer can first manually review a small set of papers and afterward let the software use an active learning algorithm to perform this task with high-quality performance. Unfortunately, the tool does not support users to collaborate with others when sharing a set of papers. The tool can be installed locally on a device but can also run on a server.
CADIMA[6] is a freely accessible webtool that can be used for various steps in conducting a systematic review of the literature. Its main purpose is to improve the screening step while performing SLR. Unfortunately, CADIMA does not provide search functionality. To insert references into CADIMA, a user needs to import a RIS file. The webtool was originally developed for the agriculture domain. The tool can facilitate the conduct of systematic

[5]https://asreview.nl/
[6]https://www.cadima.info/index.php

reviews and maps on agricultural and environmental issues. The tool can however also be used in other domains. CADIMA supports the collaboration with other reviewers. This means that multiple users can be appointed to one project which can speed up the screening process.
Another tool that is used for screening is **Covidence**[7]. This is another paid tool that promises to import citations, screen abstracts and titles, screen full text, extract data, and export data. Unfortunately, the tool lacks integration with search engines. Users need to manually search using search engines and export the results to one of the supported reference storage extensions such as PubMed and RIS text formats. In the screening environment, a reviewer can easily indicate if they want to include or exclude a paper. Keyword-highlighting features can also be found to make this process easier. As papers are included or excluded for a specific reason, they are automatically collected by the PRISMA diagram generator. This allows us to easily trace back how many papers were removed during each step of the process.

[7]https://www.covidence.org/

4 Research Questions

In this project we aim to find out what kind of best practices and open challenges exist for Change Impact Analysis in Low-Code Development Platforms. To systematically find an answer to this problem, a main question and four sub questions are proposed to achieve the objectives and goals of the research project. These questions will be answered by conducting systematic literature review which will be mentioned in section 5.
As mentioned in section , there are many similarities between Model-Driven Engineering and Low-Code Development. Therefore we assume that approaches for MDE are also applicable within Low-Code. The findings for MDE are therefore generalizable for LCDPs.

- **Research Question 1:** What are the current best practices for Change Impact Analysis in Low-Code Development Platforms?

The best practices in this domain have not yet been collected and synthesized. The outcome of doing this will benefits two groups. As current best practices are identified, companies that are developing or planning to develop a Low-Code Development Platform can receive advice and guidelines on what to use on their platforms.

- **Sub Question 1.1:** Which challenges and solutions for Change Impact Analysis in Low-Code Development Platforms exist?

Using the available knowledge of what is known about CIA in this domain and the solutions that are being used, an overview of the challenges and approaches can be given for this purpose. It must provide distinctive categories of these challenges and applicable solutions.

- **Sub Question 1.2:** What kind of technology-agnostic processes and tools exist for Change Impact Analysis in Low-Code Development Platforms?

This sub question will be answered to find out what kind of processes and tools are known in research or currently applied in the industry for CIA in LCDPs. Since the question revolves around technology-agnostic processes and tools, many solutions will be included. They don't have to be restricted to Low-Code Development Platforms but can also have their grounding in other similar approaches such as Model-Driven technologies.

- **Sub Question 1.3:** What are the open challenges for Change Impact Analysis in Low-Code Development Platforms?

In this sub question we will find out what kind of open challenges can be identified in the domain of CIA for LCDPs. Opposed to sub question 1.1, this research question aims to present open challenges for which no solutions exist yet. An example is using countermeasures to tackle negative impact as a result from evolution. An overview of these open challenges can serve as a starting point for a roadmap for low-code CIA research.

5 Research Approach

To gather information on current best practices and open challenges in the domain of Change Impact Analysis of Low-Code Development Platform, research needs to be done. There are various ways to conduct research on this topic. Interviews with domain experts who work on these platforms could be a potential research method. This however requires enough knowledge of the domain, in order to have interviews that will result in interesting answers that can give enough information to contribute to the body of knowledge (Hadar et al., 2014). As the prerequisite knowledge is not available yet in the current literature, we first need to create a body of knowledge covering the topics that will be discussed in a potential interview. To do this, we propose conducting Systematic Literature Review. In the future, the authors can decide to use the SLR results to conduct interviews with experts or to conduct further research on the SLR results by conducting case studies with industry professionals.
Performing Systematic Literature Review is one of the main ways to synthesize evidence and come to a joint understanding of the status of research on a topic (Wohlin, 2014). Kitchenham (2004) defines systematic reviews as "a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. Systematic reviews aim to present an evaluation of a research topic using a reliable, rigorous, and auditable methodology". When doing Systematic Literature Review, the author should ensure that their research can contribute to the scientific community. A set of attributes that contribute to a good systematic review was presented by ACM SIGSOFT and will be used to validate whether essential and desirable attributes are included while performing research (Ralph et al., 2021).
For this project, the Standalone Systematic Literature Review method by Okoli (2015) will be used as an inspiration. This method is used because this method was developed specifically for information systems research. Research into these information systems has different needs and Okoli tailored his method toward this. As this is one of the more recent methods, Okoli combined the best practices of different authors of SLR methods. In this method, Okoli describes the relevant steps that should be taken to do a comprehensive Systematic Literature Review. The steps were used as inspiration and transformed into the steps that will be taken during this project. These steps can be found in Figure 2. Not all of these steps are followed according to Okoli's guidelines. Some activities have been done prior to choosing this method and are therefore not needed to be performed again. Also, the "Practical Screen" and "Search for Literature" steps are combined, as they are intertwined due to the use of the snowballing approach (Wohlin, 2014). In the following sections, the activities and how they are included or excluded during this project are presented.

5.1 Identify purpose

The first step Okoli (2015) introduces is to identify the purpose of conducting a systematic review of the literature. A researcher needs to show that a Systematic Literature Review is an appropriate means to do their research. Additionally, the intended goals of the project are elicited.
For this project, the purpose of conducting Systematic Literature Review is to get a better understanding of what kind of best practices there are for Change Impact Analysis in

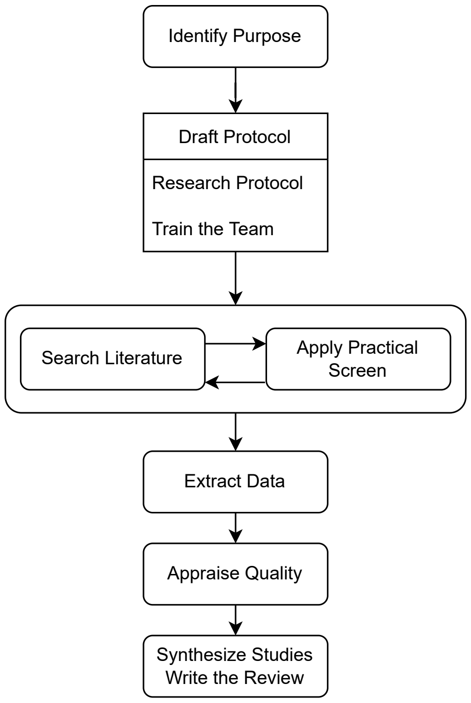

Figure 2: The steps that will be followed in this research project. Inspired by Okoli (2015)

Low-Code Development Platforms. As there has not been much prior research into this combination of topics, searching for academic sources is required to create an overview of the current state-of-the-art. Therefore, performing a systematic review of the literature is a suitable way to gather the literature and analyze the information that can be found in this literature. With the knowledge gained by doing this, the reader will have a better overview of what kinds of approaches are currently applicable in this domain. In addition, the open challenges in this domain are presented. These challenges can show what research in the future might interest the research community. The results of this SLR will therefore be useful to both researchers and practitioners in the LCDP industry. The scientific community can use the conclusions of this SLR to form new research directions. The results of this project can be used as a starting point for more specialized research on one of the aspects of Change Impact Analysis of Low-Code Development Platforms. Platform providers will also benefit from the output of performing SLR. The output will be an overview of the best practices and the available solutions (such as tools) that correspond to these practices. With the resulting recommendations for using solutions for challenges, platform providers can make better choices on what to implement in their platform to improve its capabilities for Change Impact Analysis. This will lead to an improvement in the development experience of stakeholders who interact with LCDPs.

5.2 Draft Protocol

5.2.1 Research Questions and Research Protocol

Following Okoli's SLR protocol (Okoli, 2015), the research questions are defined in this step. As research questions have previously been decided on before choosing an SLR protocol, this step is skipped in this project. The main question and its subquestions are stated in Section 4.

The next step is to write a research protocol and train the research team. As this project has a significantly large workload, the main author will be assisted by others during the literature search. Therefore, instruction on how the assisting researchers have to perform their tasks will be included.

During the development of a research protocol, a strategy is chosen to carry out the literature search. To mitigate sampling and publication bias, we will use a combination of search methods. The main search method is snowballing (Wohlin, 2014). This is an appropriate literature search method for this project because it is currently unknown which synonyms are being used to express Change Impact Analysis in the Low-Code domain. There could be multiple terms used for similar approaches and techniques. An additional advantage of snowballing is that while doing backward snowballing, we can find more papers from obscure journals that cannot be found when completely depending on search engines. During forward snowballing, the reviewers will use search engines as they provide functionality for showing which papers refer to a prior published study.

There are several steps to perform the snowball approach (Wohlin, 2014). A visual description of the steps of this approach is found in Figure 3.

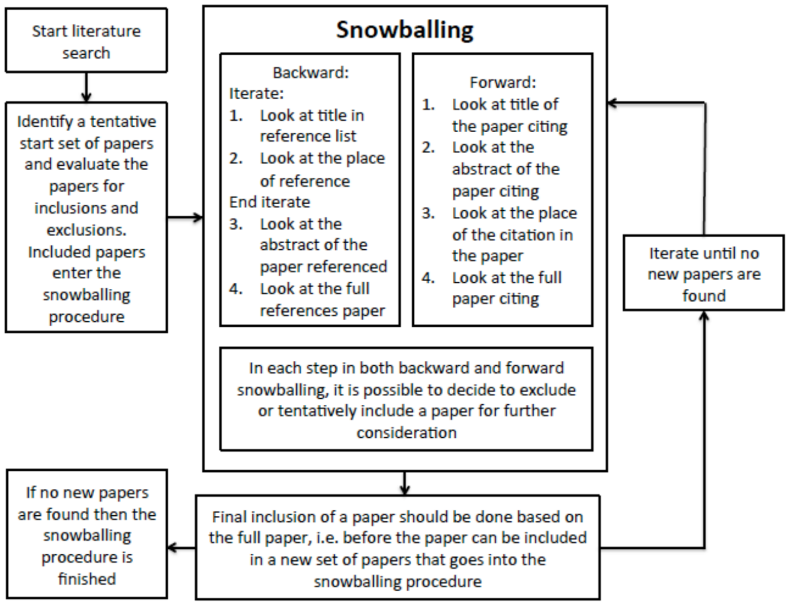
Start literature search
Identify a tentative start set of papers and evaluate the papers for inclusions and exclusions. Included papers enter the snowballing procedure
Snowballing
Backward:
Iterate:
1. Look at title in reference list
2. Look at the place of reference
End iterate
3. Look at the abstract of the paper referenced
4. Look at the full references paper
Forward:
1. Look at title of the paper citing
2. Look at the abstract of the paper citing
3. Look at the place of the citation in the paper
4. Look at the full paper citing
In each step in both backward and forward snowballing, it is possible to decide to exclude or tentatively include a paper for further consideration
Iterate until no new papers are found
Final inclusion of a paper should be done based on the full paper, i.e. before the paper can be included in a new set of papers that goes into the snowballing procedure
If no new papers are found then the snowballing procedure is finished

Figure 3: The snowballing procedure by Wohlin (2014)

In this method, the first step is to find starting set of papers. This is done by doing a keyword search on scientific search engines. To reduce bias and create a diverse set, Wohlin (2014) says that this set of papers should:

- Come from different communities. Using starting set papers in the same research community might result in clusters that mention each other.
- Not be too small. It should depend on the breadth of the studied area.
- In case of many papers found with the search terms used, identify several relevant and highly cited papers to use as an alternative.
- Include publications of different publishers. The papers should be published in different years and written by different authors. This will improve the diversity of found papers.
- Use the keywords of the research question to search for starter set papers. Make use of different synonyms that could be used.

Using the starting set, backward and forward snowballing are applied. A set of criteria is set up to see what will be included from the references. With *backward snowballing* we look at a paper's list of references. From this list, we remove the papers that don't meet the basic criteria which are the language, publication year, and whether the paper was peer-reviewed. Next, we remove duplicates already included in the list of included papers. For forward snowballing, we use papers that were referenced in more recent publications. Using Google Scholar we generate the citations of the paper. We apply the inclusion and exclusion criteria to this list and remove duplicates.
Next, the remaining references are then reviewed using another set of criteria. These are related to the title, topic, and authors. If the paper meets these criteria, we can examine the context in which the paper was referenced in the source document. We can then proceed to find the paper itself using a scientific research engine.
When the paper is collected we proceed to read the paper's abstract and conclusion. On the basis of this information, we can decide to include them in the collection of papers. These can be found in Section 5.3.
After choosing papers as inclusion candidates, we start again by examining references of another paper in the starting set and repeating the process. After the starter set has been checked for references. We iterate to the set of found papers and repeat the process at this stage. This continues until we can't find any more papers that apply to our inclusion and exclusion criteria (as mentioned in Section 5.3). The papers that are decided to include are later extracted to find relevant passages that can be reported.
As the papers should be selected with care, the author has the assistance of other researchers who will also do snowballing with the starting set papers and their iterations. After snowballing on one set, they can compare which papers they include and which they do not. This will make sure that everyone is on the same page and that the quality of chosen papers is assured before doing further iterations of snowballing. This will improve the quality of the set of articles included. This is called Inter-rater Reliability (Ralph et al., 2021).

Searching Literature In the literature research protocol, a researcher should explicitly describe how they will find the literature that they will use to answer their research questions (Okoli, 2015). They also have to explain and justify how they ensured the search's comprehensiveness.

Before choosing a starter set, preliminary research was done by reading relevant papers on the topics of Low-Code Development Platforms and Change Impact Analysis. By doing this, criteria were developed for the starting set. For the starting set, we aimed to use papers published after 2018 and before the moment of starting the search for this set (February 2022). The starting papers must come from leading journals in the areas that are relevant to the research topics. The starting set papers are related to Low-Code Development Platforms or related technologies such as Model-Driven Development and Model-Driven Engineering. Google Scholar was used to find the papers. Google Scholar is a freely available scientific search engine. It indexes papers from various sources such as academic journals and conferences. Most of the publications from leading publishers are included in Google Scholar, making it a suitable library for this project.
We have found the starting set papers by entering these search terms on Google Scholar:

- "Model-Driven" AND "Change Impact Analysis"
- "Low-Code" AND "Change Impact Analysis"

These search terms were decided after doing initial research on the topic. This resulted in 456 papers for the first pair of keywords. And 26 for the second pair. By exporting the search results using an application [8], a list of papers was created in a spreadsheet application. On this list, an initial screening was performed. First, the entries which had titles unrelated to our keywords were removed from the list. An unrelated title would be a paper with a topic that is an entirely different domain than the domain of information systems. Some keywords led to unrelated results. For example, "Change" and "Model" led to several papers focused on developing climate change models or risk analysis of climate change. They were, therefore, unsuitable to include in the starting set for this research project. Next, the papers whose titles were written in another language than English were removed. At last, we removed entries containing theses, Ph.D. dissertations, websites, and complete books. For the remaining list, we looked at the title of the paper. If the title was highly relevant to the research topic, the reviewer reviewed the abstract and the conclusion of the paper. By doing this, the reviewer was able to give each paper a score and an explanation of why the paper is suitable or unsuitable as a starting paper. The highest five ranking papers from this exploration are used as a starting set to do snowballing.
The papers in the starting set are:

1. The paper by Overeem and Jansen (2021) proposes a framework to discuss Change Impact Analysis in Low-Code Development Platforms. The paper discusses an LCDP that is currently used in the industry. It shows what kind of techniques are embedded to perform CIA on this platform.

2. Tröls et al. (2019) present an approach to check the consistency of collaborative engineering artifacts. As the models in these platforms are used by various developers. Achieving better consistency between these artifacts during the development process would improve analyzing the impact this has.

[8]https://harzing.com/resources/publish-or-perish

3. (Cho et al. (2011) introduces model-driven impact analysis that combines domain-specific modeling, constraint-based analysis, and software testing. These techniques are used to make traceable relations between artifacts that are used in software product lines.

4. Keller et al. (2009) present an approach that supports designers of applications to which extent their changes impact the software they are designing. The approach they introduce is a decision support tool that can help software designers to resolve inconsistencies between models and metamodels.

5. Butting et al. (2018) shows a method that can determine the impact of model evolution on generated and handcrafted code. Using a language workbench, it enables users of this method to identify the effect of model changes.

With this selection of papers, there is a broad search area that can be narrowed down using inclusion and exclusion criteria to find a suitable group of papers that can be used to extract data. This is mentioned in section 5.3. Unfortunately not all of the found papers fulfilled the year of publication criterion. Therefore one paper from 2009 and one from 2011 were included in the starting set.

Reviewer training As the snowball approach can be very time-consuming for one researcher, multiple reviewers are involved in this process. To make it easier to read the list of references that are found during the starter set and future iteration analysis, the workload is shared.
The involved reviewers receive an explanation on how to review the reference list. Reviewers must follow the inclusion and exclusion criteria, as mentioned in the following subsection. When reviewers follow the guidelines established, the references and the papers found when snowballing should strongly overlap each other. Comparing the found papers will keep researchers on the same page. By discussing which papers should and should not be included, changes can be made to the inclusion and exclusion criteria. ACM SIGSOFT developed empirical standards for conducting Systematic Literature Review, here they mention the necessity of Inter-Rater Reliability and Agreement (Ralph et al., 2021). Inter-Rater Reliability and Agreement are especially useful when there are cases of controversial content, practicality, and philosophy. In this project, there could be a controversial choice of relevance. Specifically, in deciding which papers should and should not be included. Therefore, comparing sets of found papers is highly recommended during this project.
After the first stage of the snowballing procedure (snowballing using the starting set), the reviewers will meet and discuss whether the sets of articles found to correspond to each other. They will also discuss whether important papers have been overlooked in this process. In this stage, the reviewers will essentially do double work. This is justified as this will lead the main reviewer in the right direction which will save time and effort later on in the project due to having a better overview.

5.3 Literature Search and Apply Practical Screen

The next steps Okoli (2015) integrates into his method are to apply criteria to decide which studies are relevant enough to be included in the review. This is called ”practical screen”. Following this step, the reviewer needs to search for literature. As the snowballing approach is an interaction between searching the literature and applying practical screen, we will treat this as one step instead of two separate steps during this research project.
During the practical screen, this criteria check is only done on a small number of criteria to dispose of unrelated papers. During the snowballing procedure, many papers will appear. Not all of these papers will be relevant. To perform this activity, several tools are available that improve this process. But in this project, it will be done manually and recorded in Google Sheets as the reviewer prefers this. This set of papers can be narrowed down using the inclusion and exclusion criteria to find a suitable group of papers that can be used to extract data. This means that papers that do not meet these criteria will not be included in the final set.
When doing snowballing with the starting set of papers, the reviewer will first remove the papers from the reference list of the paper that is already included in the collected set of papers. The reviewer will then remove papers that do not meet the basic criteria. Wohlin (2014) provided a set of criteria for this. These criteria were adapted for this project. The criteria used are as follows:

- Title: Is the paper's title related to the keywords ”Change Impact Analysis” AND ”Model-Driven” OR ”Low-Code”?
- Publication venue: Is the paper published in a location where relevant other papers on the research topics are published? Examples are journals and conferences concerned with Model-Driven software development.
- Authors: Are the authors of the paper familiar with this research area and have they previously published on these topics?

  In addition three extra basic criteria improve the quality of the set of inclusion candidates.
- Moment of publication: Was the paper published after 2005?
- Language: Is the paper written in English?
- Quality: Was the paper peer-reviewed to be published?

By answering these questions, papers can be excluded from the list based on their titles. This will give a list of candidates for inclusion. For the remaining list, more strict inclusion and exclusion criteria can be applied to have a definitive set. The reviewer will skim through the paper's title and abstract to make the decision.
One criterium was used to include or exclude the remaining papers based on their title and abstract. This criterium is:

- The paper presents an approach for Change Impact Analysis in Low-Code Development and related technologies.

There are also exclusion criteria in this step. We do this because some papers go beyond the scope of this project. These papers could be interesting in further research where these techniques, tools, and methods can be applied. These exclusion criteria are:

- The paper explicitly discusses an approach only applicable for code-based development.
- The paper discusses the implementation of a tool, technique, or method on a level that is beyond the scope of this research project.

The papers that meet these criteria are collected. In these papers, the relevant sections (such as the results and conclusion) are read to find out whether the papers consist of relevant information for this project. When a paper cannot be accessed using academic search engines with an academic account, it will be excluded from the search. The content of these papers should be applied within the domain of Change Impact Analysis in Low-Code Development Platforms. The papers that meet these criteria will be part of the final set.
The included papers are entered into a Google Spreadsheets document where the reviewers can easily find and trace back which papers are included, where they were found, and why they are relevant for this project. This allows for better collaboration between reviewers.
When no other papers that meet these criteria can be found through references the snowballing procedure ends.
The next step described by Okoli (2015) is to perform the actual search. In the previous steps, we explained that we are applying the snowballing technique by Wohlin (2014) and include the practical screen while applying this technique. The literature search is visualized as a simplified PRISMA flow diagram. It can be found in Figure 4. We started with a set of 456 papers with our first query and an additional 26 for the second query. Using this set of papers, the papers most relevant to the research topic were chosen as starting set papers. This set of five starting papers will be included in the final review. This set of included papers will be supplemented with additional papers that will be found with backward and forwards snowballing. The collection of the papers using snowballing will be documented by following the PRISMA 2020 checklist 2021. As a result, a flow chart will be presented that shows how many papers are included and excluded at each stage of the literature search. When snowballing is applied, we increase our set of papers and decrease them based on our search criteria. After each iteration (first, the starting set and then using the papers that were found using the starting set), we see if more relevant papers are found based on our criteria. If no valuable additions are found the search process ends. Throughout this process, the literature found is collected in an Excel spreadsheet. A list of tools was collected and synthesized on the basis of their characteristics. This is found in Section 3.5. In the end, for this project, Excel and similar worksheet tools such as Google Sheets were chosen. These tools are very versatile and can be used to store the necessary data in an organized manner. When changes need to be made to codes, research questions, and/or collected information, these worksheets are easily adaptable. As the literature and the application of the Practical Screen are intertwined, the process looks a bit different than it would normally look when applying the steps by Okoli (2015).

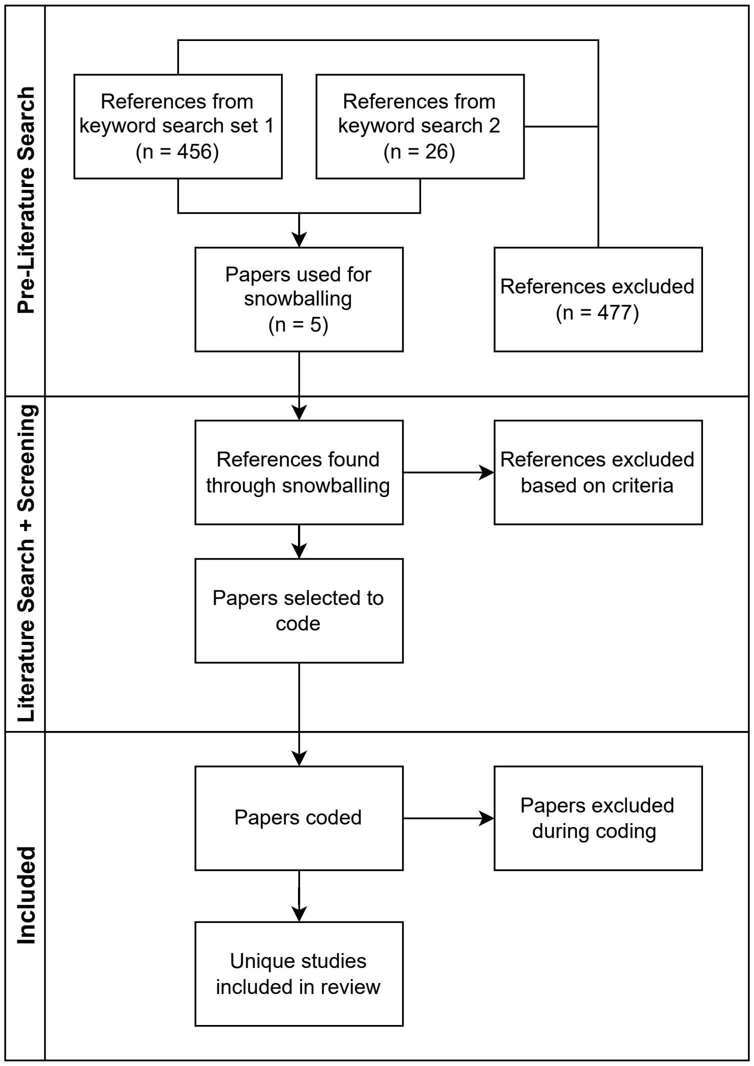
Pre-Literature Search
References from keyword search set 1 (n = 456)
References from keyword search 2 (n = 26)
Papers used for snowballing (n = 5)
References excluded (n = 477)
Literature Search + Screening
References found through snowballing
References excluded based on criteria
Papers selected to code
Included
Papers coded
Papers excluded during coding
Unique studies included in review

Figure 4: Simplified PRISMA 2000 chart (Page et al., 2021) of the literature search, and synthesis.

5.4 Extract Data

When the full set of relevant papers is collected during the Search for Literature step, we continue to the Extract Data step. During the Extract Data step, a reviewer systematically takes information from the selected papers that can be used to synthesize this information. Okoli (2015) advises reviewers to make use of a form of data extraction that supports storing details and commenting on segments. NVivo is a tool that supports this and can be used for this purpose. Wolfswinkel et al. (2013) developed a *grounded-theory method* to review the literature. As previously described, there are already specific search terms, research fields, and inclusion/exclusion criteria. Therefore only the *Analyze* and *Present* stages in their method are applicable in this project.
During the Analyze step in the method by Wolfwinkel et al., the reviewer will apply codes to the studies that were decided to be included in the review. Three coding activities are

done; *Open* coding, *Axial* coding, and *Selective* coding. During open coding, the hidden aspects that were missed during the prior search are conceptualized. These segments can be incorporated into concepts and insights for which labels/codes can be made. During the axial coding step, the interrelations between categories and subcategories are defined. When creating categories and derived subcategories can become the main themes in the review. Finally, selective coding is done to find the relations between the main categories. The categories are then integrated and refined to be more complete and applicable for the segments in the studies. This will lead to an overview of the selected categories and how they can be presented in the review.

5.5 Appraise Quality

During the appraise quality step, Okoli (2015) suggests researchers to evaluate the quality of the extracted information during the previous step. If the papers and the knowledge found inside those papers end up being of low quality, they can be removed from the set of papers. Due to the prior application of the inclusion and exclusion criteria during the "practical screen" step, we only have to do quality appraisals based on the type of studies within the papers that could be included. In the Qualitative appraisal, studies that have been implemented in real-world situations are preferred over those in controlled environments. In this project studies that are tightly connected to Low-Code Development Platforms are prioritized over those in related technologies such as Model-Driven Engineering. As in (Di Ruscio et al., 2022; Bock and Frank, 2021), there can be an overlap of low-code and model-driven techniques within the approaches. Therefore, many Model-Driven approaches are applicable in LCDPs and therefore inclusion candidates. Studies related to traditional software development will be included when alternatives are not available in the aforementioned context.

5.6 Synthesize Studies and Writing the Review

The last two steps included in Okoli (2015) are to synthesize the studies and to write a review of the literature. With the codes that are made and placed into various categories and subcategories (Wolfswinkel et al., 2013), it is possible to synthesize the found information to answer the research question. These categories provide a structure in which the relevant passages can be placed. These can be written down as such in the literature review. The (sub-)categories and their contents can be used as input for data visualizations such as tables, graphs, and images. They can also be used to convey knowledge to the reader. An example would be a tree structure using the papers that are included in the review based on the snowballing approach. This can show the relations between papers and what the process has been during the literature search. An example can be found in the paper by (Overeem and Jansen, 2017).
For this project, a metamodel for the to-be-developed artifacts is proposed. In this metamodel 5, there are several aspects that play a role in Change Impact Analysis approaches for LCDPs. The blue rectangles are the concepts for which tables are going to be developed.

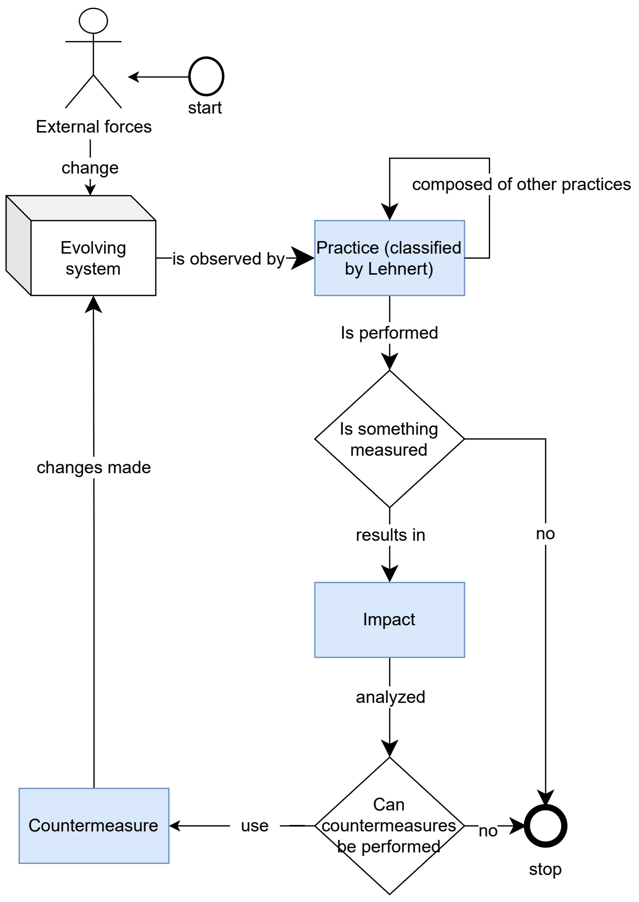

Figure 5: Metamodel of the collected information in relation to CIA for LCDPs

LCDPs are continuously evolving systems. Due to deliberate and accidental changes, new versions of a platform are created. There are internal and external forces that must be taken into account. To find out what changes are made and how they influence aspects of the system or the system as a whole, CIA practices can be implemented. There are many practices/approaches available for this purpose. They can be categorized using the taxonomy by Lehnert (2011b) which offers various criteria to compare and classify approaches. As this taxonomy is not developed especially for MDE/LCDP approaches, some adaptations might have to be made in order to be compliant with aspects that play a more significant role in these development methods that have some different characteristics compared to code-based development. These practices can work independently or could be dependent on other practices in order to perform their intended activities to analyze what is impacted within an evolving system.

There are also circumstances in which no negative impact is detected. This ends the process untill external forces make new changes. If an impact can be detected, some practices will provide countermeasures. These countermeasures could be restoring the changes made in order to make a system function as intended, or suggesting alternative changes in order to get to the intended state of the system. This leads to the system evolving again and restarting the entire process.

Within this project, the LCDP CIA practices are collected using the taxonomy by Lehnert

(2011b). Next to this, we also collect the challenges the authors describe and the future work that is proposed. This will make it possible to give a better overview of the current state of the art within this domain. Coding papers based on predetermined codes is called *deductive coding*. When other interesting segments in the papers appear that do not match one of the predefined codes, inductive coding is used. New codes are created to give these text segment a place.

5.7 Metamodel tables

To structurally synthesize the findings in the found approaches, a metamodel was developed that shows which artifacts can be made to answer our research questions. The main artifact will be a table containing the found relevant approaches (Table 4). The found approaches have characteristics that can be collected. The characteristics used, come from the taxonomy of Lehnert (2011b). This taxonomy was developed to classify CIA approaches. Using this taxonomy can classify and compare the approaches found. As the taxonomy is used for a multitude of CIA approaches, and not specifically for LCDP/MDD CIA approaches, there might be a need to adapt the taxonomy and the collected characteristics in order to comply with specific aspects of these approaches.
In their taxonomy (Lehnert, 2011b), the author collects a multitude of characteristics an approach can have. In this table, not all of these characteristics will be presented. Some of the characteristics will be excluded because they are less relevant to the goals of this project. The collected characteristics in the table are:

- **Description** A short description of the approach and whether it is being used. This is not mentioned in the taxonomy by Lehnert (2011b).

- **Scope of analysis**: Does the approach work in a specific part of the information system. In our case, it will be focused on LCDPs and mainly on the model and metamodel. Lehnert puts these under the *Models - Requirements* and *Models - Architecture* criteria. Lehnert presents also code-based criteria which are relevant for the interpreter side of the LCDPs that are analyzed using the approaches. There are three distinct types of source code analysis scopes. These are *Static*, *Dynamic*, and *Online* approaches.

- **Technique**: Lehnert identified ten different techniques to which his found approaches could be compared. Examples are *Program Slicing*, *Message Dependency Graphs*, and *Tracability*: Some of these ten techniques were found during initial snowballing research.

- **Style**: Lehnert defines three styles for CIA. (**1**) *Global Analysis*: which looks at the whole system. Independently of the current task that is performed. (**2**) *Search based*: when the strategy operates on-demand to find more about a specific change request (**3**) *Exploratory*: when an approach step-by-step goes to the system and presents the user possibly impacted elements in the application.

- **Granularity**: Which level of granularity does this approach support. This is shown for the artifacts, the changes, and the results (which are reported by making use of the approach).

- **Independent**: Is the approach independent, or does it make us of other approaches to perform its activities. This is not found within Lehnert's taxonomy.

ID	**Description**	**Scope**	**Technique**	**Style**	**Granularity**			**Independent**
					Artifacts	**Results**	**Changes**	

Table 4: Collected characteristics of found approaches

In the final step, the findings of the literature review are highlighted to present the contribution to the research domain. This way, the readers will be able to extract what was written in the previous chapters. A structure based on the metamodel and filled-in tables will be used to present the found results and artifacts created by conducting SLR. Additionally, the steps that are taken to conduct the research are presented so that other researchers will be able to replicate the results when following the same steps.

6 Papers found

By executing the protocol, a set of 23 papers was collected. The papers within this set present approaches for Change Impact Analysis that are applicable in systems that make use of Model-Driven Engineering and related techniques.
The process of the literature search and an overview of the number of selected papers is visualized in a simplified Prisma diagram (Page et al., 2021). This can be found in Figure 6. Throughout the process, more than 1500 references were examined and 23 papers were selected to be included in the literature review after coding.

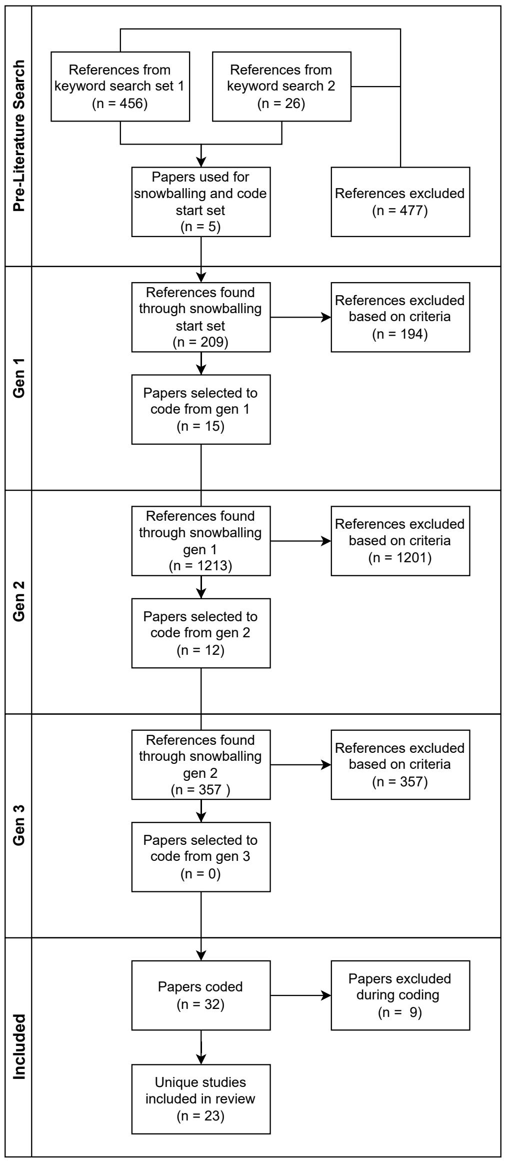
Pre-Literature Search
References from keyword search set 1 (n = 456)
References from keyword search 2 (n = 26)
Papers used for snowballing and code start set (n = 5)
References excluded (n = 477)
Gen 1
References found through snowballing start set (n = 209)
References excluded based on criteria (n = 194)
Papers selected to code from gen 1 (n = 15)
Gen 2
References found through snowballing gen 1 (n = 1213)
References excluded based on criteria (n = 1201)
Papers selected to code from gen 2 (n = 12)
Gen 3
References found through snowballing gen 2 (n = 357)
References excluded based on criteria (n = 357)
Papers selected to code from gen 3 (n = 0)
Included
Papers coded (n = 32)
Papers excluded during coding (n = 9)
Unique studies included in review (n = 23)

Figure 6: Simplified Prisma diagram of the literature search (Page et al., 2021)

The papers and their sources can be found in Figure 7. Each oval represents a paper, and each line is a reference from a paper to a referenced paper. The starting set of papers is highlighted in yellow.

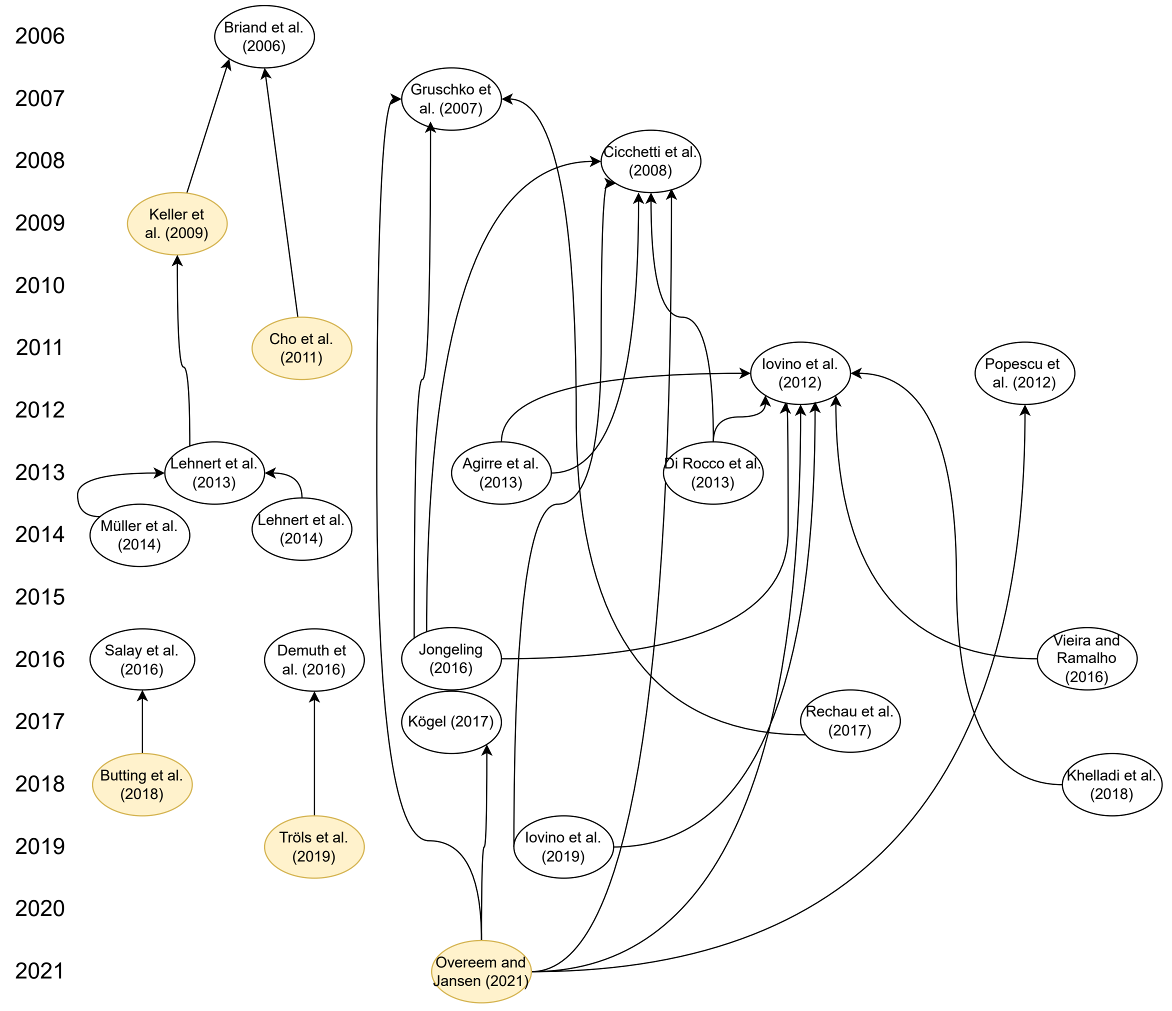

Figure 7: Graph of papers found through snowballing

It was noticeable that the paper by Overeem and Jansen (2021) led to many interesting backward sources, as this project was mainly inspired by this paper. The papers found led to more interesting sources as can be seen in Cicchetti et al. (2008) and Iovino et al. (2012). And these papers both referred to many other interesting sources that were highly relevant. The starting set papers of Tröls et al. (2019), Cho et al. (2011), Keller et al. (2009), and Butting et al. (2018) only led to a small number of papers when doing backward and forward snowballing. It appears that these papers lacked some relevancy to this topic and referred to other papers that were in other domains than the papers found by snowballing with the paper by Overeem and Jansen (2021). In this paper, the authors mention several approaches that are already known to be effective and were therefore an easy and trustworthy source to use in the start set. It might have been a better choice to only use this paper in the starting set. This would have saved time that was now spent on finding other relevant papers that ended up not giving the results that we wanted to achieve with this set. Limiting ourselves to only include papers published within the time frame of 2018 and 2022 might have been a limiting factor as this domain of research is not very active.
During the execution of our practical screen using snowballing, we came across some papers

that were relevant to the topic of CIA within LCDPs and other MDE techniques were excluded because they did not satisfy the criteria. The systematic review by Alam et al. (2015) is a good example of this. Even though no approach is presented by the authors, the paper presents more background information to discuss other found approaches. It is therefore excluded from the found approaches in the main list of papers but can be included in an extra set of papers.

Some of the approaches found were removed from the set of included papers at a later stage of the process. During the coding of the papers, several of them appeared to not present the information that we were looking for. An example of such a paper is Di Rocco et al. (2018). This paper did not present an approach but gave an overview of the impact certain metamodel changes have in a specific Eclipse editor (Sirius[9]) that supports graphic modeling. These papers could have been omitted at an earlier stage. But due to a lack of domain knowledge and a relevant title, we kept this a candidate until the later stage of the project where we finally decided on excluding the paper.

As the papers Gruschko et al. (2007), Cicchetti et al. (2008), and Iovino et al. (2012) were very fruitful in leading to more papers, we tried to find out whether more papers could be found that were initially missed. Using the website Litmaps[10] we found that these three papers were referenced by mutual other authors. The paper by Hebig et al. (2016) presents approaches to the co-evolution of metamodels and models that can be used as background information to describe other approaches that were found by snowballing. Using the Researchrabbit tool[11] a paper from Kessentini and Alizadeh (2022) was found that did not yet appear when snowballing. This paper introduced a semi-automated metamodel/model co-evolution approach. Upon examination of the paper, it appears that it would have been included if it had been found during the process. Both papers can be used to support statements that will be made in the following sections. As snowballing is a time-consuming and mentally exhausting task, some people tend to overlook things. When there is limited manpower and experience in doing the snowballing task, users might have a less complete set of selected papers.

The following sub-section presents the included papers, where they were published, and a brief description of the approach.

6.1 Approaches

Overeem and Jansen (2021) *MODELS-C, 2021* This paper proposes an LCDP CIA framework. The authors also described how CIA was done within an industry LCDP. The analysis is done on the source code and models. The approach uses some sort of explicit rules, traceability, message dependency graphs, and differencing. The analysis is done on various levels. For example, the OEM diff and evolution operations are done on all available levels, while the mergelog that provides the co-evolution of model and customer data operates on the attribute level. The change analysis results are represented as diffs. And also message dependency graphs are used to represent the run-time model of the application. The approach is used explicitly for the platform AFAS developed and therefore not easily transferable to

[9]https://www.eclipse.org/sirius/
[10]www.litmaps.com
[11]www.researchrabbit.ai

other applications. The mentioned techniques were implemented within their platform and the authors presented their framework to discuss CIA for other platforms.

Tröls et al. (2019) *MODELS-C, 2019* The approach by Tröls et al. is multifaceted as many different types of artifacts can be analyzed. It focuses on code and various other artifacts such as requirements and models. It makes use of traceability to point from the code to a generated uniform artifact representation that is linked to an artifact such as a UML model. A consistency checker using explicit rules is afterward used to analyze the impact of the artifacts change. As explicit rules are used, the analysis style is search-based. The evaluation result is stored within the Consistency Rule Evaluation Artifact that presents relevant engineers concerned with these artifacts to take action when necessary. The approach works with several artifact types such as UML models, Excel files, and Eplan P8 electric models. The approach is built into an Eclipse platform application.

Cho et al. (2011) *Model-Driven Domain Analysis and Software Development: Architectures and Functions, 2011* This approach is concerned with CIA of Domain Specific Models of Software Product Lines. Traceability is established and the models are transformed into an augmented constraint network (ACN) which encompasses the assumptions and constraints in heterogeneous software artifacts. A framework then uses the ACN to calculate metrics about design candidates. The metrics can help engineers decide which design is least volatile to changes. Case studies were conducted but no tool was presented by the authors.

Keller et al. (2009) *Emerging Technologies for the Evolution and Maintenance of Software Models* The authors developed an algorithm that can predict the impact of inconsistency resolutions. The approach makes use of explicit rules to check which model elements are impacted. As a result of executing the algorithm, the inconsistency instances are presented and the violated rules are presented. No tool was presented.

Butting et al. (2018) *MOD-WS, 2018* A method is introduced for shepherding model evolution for Model-Driven Development tool chains. The method works with MontiArc, which is an architecture description language. The technique that is used to find the impacted artifacts due to evolution is differencing. As a result, the developers receive the impacted artifacts when the model evolves. The approach does not look at the performance but solely on the syntactic changes of models. Developers can use the impacted artifact to manually find out what the impact is on the performance.

Kögel (2017) *ESEC/FSE, 2017* Kögel proposes an approach using history mining to give recommendations for Model Driven Software Development. Ecore metamodels were used to learn about the common model transformation. The prototype recommender system can analyze the change history of a single model. For these transformations, recommendations are generated and presented to the developer. The author has not yet completed the tool as more data has to be gathered and more case studies need to be done to validate the performance of the approach.

Popescu et al. (2012) *DEBS, 2012* An approach for impact analysis of distributed event-based systems is described. Using the source code of a system, message dependency graphs (MDGs) can be created that capture inter-component and intra-component dependencies. When comparing versions of an MDG before and after a change, the differences can be compared which can provide developers with opportunities to check components that require attention. These graphs can be generated for Java, C++, and C# code. The authors called their technique Helios (Popescu, 2010).

Gruschko et al. (2007) *Proc. Int. Workshop on Model-Driven Software Evolution held with the ECSMR, 2007* Gruschko et al. state that metamodel evolution can pose a threat to the applicability of Model-Driven Development to large-scale projects. When metamodels evolve, models do not conform to the metamodel anymore. The authors introduce an approach that can find and synchronize these metamodels and models. The approach is concerned with the architecture of the system and makes use of traceability and explicit rules. The model transformations are analyzed and the relevant set of changes is collected. The approach supports the transformations of Ecore-based M2 models. No complete tool has been presented in the paper, only a prototype was made.

Cicchetti et al. (2008) *12th International IEEE Enterprise Distributed Object Computing Conference, 2008* The approach by Cicchetti et al. (2008) attempts to automate co-evolution within MDE. They focus on metamodels and models. Explicit rules and Differencing are used to facilitate this. The input of this approach are the difference models that were gathered by using Ecore and MOF metamodels. By performing the approach, exported traces can be used to instantiate changes to the model-based changes in the corresponding metamodel. Unfortunately, this only supports structural features at the moment.

Iovino et al. (2012) *Journal of Object Technology, 2012* Iovino et al. proposes an approach that allows developers to establish relationships between the metamodel and its related artifacts. It also automatically identifies the elements within these related artifacts that will be affected by metamodel changes. This analysis is done on modeling elements that are impacted during metamodel evolution. The approach supports several types of artifacts that can interact with the megamodel that is used as input. These are Petri net metamodels and ATL transformations.

Demuth et al. (2016) *IEEE ICSME, 2016* is an approach that presents an example of Tröls et al. (2019). They also use a uniform data representation so multiple artifact types can be supported. The analysis is done using Explicit rules and Traceability. The result of the analysis is an overview of affected artifacts after a change and the responsible engineer. A tool was presented as the output of the paper.

Briand et al. (2006) *Journal of Systems and Software, 2006* The author presents an approach for the analysis of UML diagrams. The approach makes use of Explicit rules to detect changes in components of UML models. A measure of distance is calculated that presents the distance between the changed model elements and the impacted elements. The distance is calculated using the number of impact analysis rules (explicit rules) that had to be invoked to identify the impacted element. As a result, a prototype tool called iACMTool was presented by the authors.

Lehnert et al. (2013) *Proceedings of the Euromicro Conference on Software Maintenance and Reengineering, 2013* Lehnert presents a rule-based analysis of heterogenous software artifacts. Explicit rules and traceability are used within the approach. Similar to Tröls et al. (2019) and Demuth et al. (2016) they decided on using a uniform representation of the artifacts so it supports various modeling languages. When executing the analysis, the user is presented with the impacted elements, and also how they are impacted. So that developers will know how they should be changed. The authors presented a tool as their main output.

Salay et al. (2016) *Software and Systems Modeling, 2016* Megamodel slicing is used within this approach. Megamodels can represent collections of interrelated models. The approach makes use of traceability relations to assess the change impact. The analysis within this approach is used on a megamodel of a UML diagram. As a result, the approach presents the fragments of the megamodel that are impacted by evolution. The authors are developing a tool for the MMINT framework (Sandro et al., 2015).

Jongeling (2016) *TU Eindhoven* In this master thesis, the author presents an approach for CIA within MDE systems. It makes use of information retrieval and Explicit rules. The artifacts that the approach supports are domain-specific languages and model-to-model transformation. As a result, it presents the elements that are likely to co-change once the metamodel of the system changes. A tool was presented by the author.

Rechau et al. (2017) *European, Mediterranean, and Middle Eastern Conference on Information Systems, 2017* In this approach model transformations are the main target. Explicit rules are used to analyze the transformations of Enterprise Architecture metamodels and models. As a result, it presents the qualitative assessment of the impacted instances. A tool is presented for this purpose.

Iovino et al. (2019) *45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2019* The main topic of this paper is the proposal of a query-based impact analysis approach for the evolution of metamodels. The approach uses evolution definitions to generate OCL queries that can be executed for models and transformations. As a result, the impacted elements can be obtained. Developers can then decide whether they should proceed with the evolution or not. The approach supports Ecore model and metamodels, and ATL transformations.

Agirre et al. (2013) *AMT@MoDELS, 2013* This approach is focused on the impact analysis of software architecture migration. The tool that is presented by the authors can do an automated analysis of the impact of software architecture changes that are caused by evolution. The tool makes use of explicit rules, and traceability and uses model differencing in order to do the analysis. The result of the analysis is a model of the adaptation goals.

Di Rocco et al. (2013) *Proceedings of the Second Workshop on Graphical Modeling Language Development* To improve the visualization of metamodels, Di Rocco et al. propose an approach. The approach uses traceability information and model differencing for this. Transformation elements in relation to the changed metamodel elements are used as input and as a result, the approach shows a visualization of these traces. For this project, they used ATL and ECore metamodels.

Müller and Rumpe (2014) *Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 65, 2014* In this approach, difference models are generated that are then checked with explicit rules who define the consequences of the changes between UML class diagrams. The approach currently supports UML class diagrams for their artifacts. As a result, it presents checklists with hints about development steps that can help to manage the evolution of the system.

Lehnert et al. (2014) *Modiellierung, 2014* Lehnert et al. present an approach that looks at model dependencies for rule-based regression test selection for business processes. They support BPMN, UML, and UML testing profile models. To achieve this, explicit rules were used. These rules were implemented within a tool called EMFTrace and have been evaluated in this project. After analysis. The tool can present the user with the impacted test elements.

Vieira and Ramalho (2016) *International Journal of Software Engineering and Knowledge Engineering 26* This approach makes use of explicit rules to analyze ATL model transformations. As a result, it presents an impact value of a given change and the impacted elements. A tool for this purpose was presented.

Khelladi et al. (2018) *21th ACM/IEEE International Conference* The final selected paper presents an approach that looks at side effects when a model inconsistency is repaired. The approach the authors presented can automatically detect and track the consequences of repairs. (for example side effects). The analysis is done on transformations and explicit rules are used as the main technique. As a result, the approach presents a ranking of possible repairs that have the least side effects (negative impact) on the system.

7 Results

In the past section (Section 6.1), the found approaches were presented. The relevant information was extracted from the papers using the codes by Lehnert (2011b). The information is also briefly described in the Appendix within Table 9. The yellow highlighted fields show concepts that do not occur often when looking at the other found approaches. These concepts are explicitly mentioned when discussing the techniques, styles and scope.
In the following sub-sections we discuss the characteristics that can be found according to taxonomy by Lehnert (2011b) .

7.1 Scope

Lehnert (2011b) first mentions the scope of the approach within his taxonomy for change impact analysis approaches. The scope of analysis describes what kind of artifacts an approach is concerned with. Lehnert states that approaches can be applied to code, models, miscellaneous artifacts, or a combination of these different artifacts.
Distinction between different source code level approaches were made between *Static*, *Dynamic*, and *Online* analysis. Static is based on syntactic and lexical analysis of source code, in dynamic analysis, analysis is performed after data has been collected from program execution, and online analysis is done while the program is being executed in real-time.
Lehnert also makes a distinction between analysis that is done on models. Analysis can be done on requirement specifications, but also on the architecture and the abstract design of the software. The scope that was mostly used within the found approaches was the architecture of models and their abstract design. In Lehnert (2011a), the author mentions that architectural models such as UML component diagrams can enable the assessment of architectural changes on a more abstract level than source code. This is especially useful when doing impact analysis during earlier stages of development and within "model-based development" which can be seen as MDE. This scope brings different levels of granularity such as systems, sub-systems, components, and classes. This classification (Models - Architecture) is also concerned with transformations between models, and are therefore the mainly found scopes within this analysis.
As this project did not focus on code-based development, the analysis on the code level was only found in four of the papers, which were Overeem and Jansen (2021) and Popescu et al. (2012) for static analysis (before the code was executed). Wheras Tröls et al. (2019), and Demuth et al. (2016) made use of analysis of code after the compiler in an IDE was executed. The requirement specification scope was never used as no papers were included with approaches for the impact analysis of requirement specifications. The scopes that were used for the papers can be found back in Table 5

7.2 Techniques

In his taxonomy, Lehnert shows 10 different techniques that he identified while doing research on CIA approaches. These 10 techniques were identified within a research project where Lehnert combined existing taxonomies of CIA to make a more comprehensive taxonomy (Lehnert, 2011b). These techniques were used often within the found approaches that showed

	Models - Architecture	**Code - Dynamic**	**Code - Static**
Overeem and Jansen (2021)	X		X
Tröls et al. (2019)	X	X	
Cho et al. (2011)	X		
Keller et al. (2009)	X		
Butting et al. (2018)	X		
Kögel (2017) X			
Popescu et al. (2012)			X
Gruschko et al. (2007)	X		
Cicchetti et al. (2008)	X		
Iovino et al. (2012)	X	X	
Demuth et al. (2016)	X		
Briand et al. (2006)	X		
Lehnert et al. (2013)	X		
Salay et al. (2016)	X		
Jongeling (2016)	X		
Rechau et al. (2017)	X		
Iovino et al. (2019)	X		
Agirre et al. (2013)	X		
Di Rocco et al. (2013)	X		
Müller and Rumpe (2014)	X		
Lehnert et al. (2014)	X		
Vieira and Ramalho (2016)	X		
Khelladi et al. (2018)	X		
Total	22	2	2

Table 5: The scope of the found approaches as described within Lehnert (2011b)

up in his project. Lehnert mentions that these techniques could be refined in the future. Several of these techniques were applied within the found approaches in this project. Next to these techniques, we found that *differencing* is an additional technique that facilitates CIA for MDE. The papers and their corresponding techniques can be found in Table 6

	Explicit rules	**Traceability**	**Differencing**	**Message Dependency Graph**	**Information Retrieval**	**History Mining**	**Program Slicing**	**Total**
Overeem and Jansen (2021)	X	X	X	X				4
Tröls et al. (2019)	X	X						2
Cho et al. (2011)	X	X						2
Keller et al. (2009)	X							1
Butting et al. (2018)			X					1
Kögel (2017)						X		1
Popescu et al. (2012)				X				1
Gruschko et al. (2007)	X	X						2
Cicchetti et al. (2008)	X		X					2
Iovino et al. (2012)		X						1
Demuth et al. (2016)	X	X						2
Briand et al. (2006)	X							1
Lehnert et al. (2013)		X	X					2
Salay et al. (2016)							X	1
Jongeling (2016)	X				X			2
Rechau et al. (2017)	X							1
Iovino et al. (2019)	X							1
Agirre et al. (2013)	X	X	X					3
Di Rocco et al. (2013)	X		X					2
Müller and Rumpe (2014)	X		X					2
Lehnert et al. (2014)	X							1
Vieira and Ramalho (2016)	X							1
Khelladi et al. (2018)	X							1
Total out of 23 papers	18 (78%)	8 (35%)	7 (30%)	2 (9%)	1 (4%)	1 (4%)	1 (4%)	

Table 6: All of the mentioned techniques as described by Lehnert (2011b) for each approach. *Differencing* was added as this supplemented Lehnerts list

Explicit rules As can be found in the main table of the characteristics of the found papers and Table 6, most (78%) of the papers are making use of explicit rules. Briand et al. (2006) mentions that impact analysis rules can be used to determine which model elements are directly or indirectly impacted by changes. Within Lehnert (2011b) the author mentions that these rules are based on design and domain knowledge. Design, domain, and expert knowledge are used to form strict impact rules. They can determine which entities should change when a certain entity changes Lehnert (2011a). Something similar to these explicit rules is found in Neto et al. (2013) in which the authors develop a language to specify design rules. The design rules can be described in a declarative manner which makes it easier to develop automatic verification mechanisms of these rules in the written code.
Müller and Rumpe (2014) mention that the main motivation for explicitly specifying impact rules is that they allow leveraging the known dependencies and characteristics of a product. Explicit impact rules can embody the conditions that have to be fulfilled by UML class diagram changes in order to have an impact. But they can also contain what the actual impact would look like. Within Overeem and Jansen (2021) the developers used explicit rules for the co-evolution of model and customer data within their platform.
Many other papers mention these or similar rules (migration rules etc.) as a key technique that powers their approach. In total, 17 out of 23 papers mention these rules. This makes up 73% of the found approaches. These approaches can be found in Table 9 and 6.

Traceability IEEE (1990) defines traceability as

- The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor–successor or master–subordinate relationship to one another.

- The degree to which each element in a software development product establishes its reason for existing.

This is especially useful within CIA to find out what happens with certain elements when the operations are executed. Lehnert (2011b) say that traceability links are able to capture the relationships between entities of different levels of abstraction. When aggregating these links, they can be treated as a graph. With this information, the maintainer of the software can understand the relationships and dependencies among artifacts. With this knowledge, they can find out what part of the design, models, and code needs to be maintained in order to work as intended (De Lucia et al., 2008). Within Overeem and Jansen (2021), the authors use traceability links in the transformation system. Traceability links are used to link elements in the run-time model to transformation components. Tröls et al. (2019) and Demuth et al. (2016) use traceability links in a special way. They generate a uniform artifact representation and place traceability links to connect the code to the artifacts with the uniform artifact representation in the middle.

Differencing Differencing is a technique that is not mentioned within Lehnert (2011b). Using a differencing technique it is possible to find out what the differences are between two or more artifacts (Kolovos et al., 2009). In their analysis of approaches to model migration,

Rose et al. (2009) mention *differencing* as an activity during the *metamodel matching* strategy to migrate models.
Agirre et al. (2013) uses difference models generated based on metamodel evolution. In their approach, they use differential models of the models that represent the code in order to establish adaptations that must be made in transformation rules. Di Rocco et al. (2013) used differencing in their approach to make traceability visible for metamodel change impact detection.
Within Overeem and Jansen (2021) these results of the difference between models is called a *diff*. These diffs can later be analyzed with other techniques. Within this paper, diffs are represented in a text-based way using JSON.
This evokes a short discussion as the taxonomy by Lehnert (2011b) does not mention differencing as a technique for CIA. We see this technique as a supporting technique that makes it possible to execute the analysis with the found approaches. A difference model needs to be calculated in order to be used by an approach. Lehnert (2011a) applied his taxonomy within a research project where he reviewed 150 approaches. In this paper, he mentions that differencing is used as an activity that is necessary in order to create message dependency graphs. They also mention that papers by a set of authors make use of *diff* tools to support the differencing of hierarchical models. With these tools hidden evolutionary dependencies can be extracted in the future to predict future changes.

Message Dependency Graphs As mentioned by Lehnert (2011b), Message Dependency graphs can present communication between systems and can propagate changes between them. An example by Popescu et al. (2012) shows how message dependency graphs can be created. These graphs are made with various levels of analysis. Within their technique, control flow, state-based dependency analysis, and structural analysis were used in order to create these graphs. This technique has also been used within Overeem and Jansen (2021) in which they made use of message dependency graphs to visualize the run-time model. The created graph shows the event-based communication between the different micro-services involved within the platform. When a difference model is made of two versions of the run-time model's message dependency graphs, various observations can be made. The authors mention that with these difference results, the developers have an overview of the nodes and edges that are added, removed, and changes between versions.

Information Retrieval In this technique similarities between attributes of classes, methods, and other parts of the source code are used to identify impacted elements (Lehnert, 2011b). Lehnert (2011a) mention that information retrieval (IR) is a textual retrieval method as they are concerned with natural language in order to infer a relation between similar documents. IR was only vaguely mentioned by Jongeling (2016) as one of the techniques that he applied for the analysis of a system. The specifics of why he did that are therefore unknown. We can however imagine a role for IR in order to find similarities between names of entities within models and source code that are being used throughout an LCDP. Using these similarities and combining these with history mining, better suggestions can be given to developers.

History Mining By using history mining, the entities that are often changed together can be identified. They are likely to change again during a similar action. With this knowledge, it is also possible to make recommendations for better alternatives. Kögel (2017) makes use of history mining for their approach. Different sources of data can be combined to create a list of relevant model changes in real-time. This can support users in automating repetitive tasks and prevent errors that are caused by incorrect changes. In their approach, the recommendations were generated based on the change history of the model.

Program Slicing Salay et al. (2016) mentions Program Slicing as the technique that is used in their metamodel slicing for the model evolution approach. Program Slicing can be described as returning the parts of the source code that are affected by a change (Lehnert, 2011b). This subset can then be closely observed. Most research on slicing within MDE the context is focused on particular model types and are therefore not technology agnostic.

7.2.1 Unmentioned techniques

Call Graphs A call graph visualizes the method and function calls that are extracted from the code. When methods or procedures are changed, they can affect other parts of the source code Lehnert (2011a,b). When the call behavior of these methods is analyzed, the impact of method/procedure change can be assessed. The method calls are then extracted and stored in for example a matrix or graph. This can allow developers to estimate the propagation of an observed change.

Execution Traces Approaches that use execution traces look at methods or functions that were called during the execution of a program (Lehnert, 2011b). By monitoring the running software, traces are collected. With these traces, it is possible to see which methods were called after something changed. Execution traces are an improvement over static slicing which is expensive and uses call graphs that are seen as imprecise Lehnert (2011a).

Program Dependency Graphs Program Dependency Graphs (PDGs) are graphs that show communication between program entities that are found in the source code of a program Lehnert (2011b). The changes that are made within the source code can then be propagated through the entire graph of the program. During the literature search, there were no approaches to make use of this technique. As LCDPs make use of multiple sub-systems that communicate with each other, Message Dependency Graphs seems to be a better technique to use instead of PDGs. PDGs are more suitable for monolith systems.

Probabilistic Models Within this technique, mathematical models and theorems are used to model the propagation of change. The probability of an entity being impacted is calculated Lehnert (2011a). These models make use of Markow Chains, Bayesian Belief Networks or similar models to calculate this probabilities. To improve these calculations, they can be combined with greedy algorithms or linear and nonlinear systems of equations.
This technique has not appeared in the approaches that were discovered in the literature review. However, Breuker (2014) presents how MDE can facilitate the development of machine learning using probabilistic models.

7.3 Style of analysis

Several distinct styles are mentioned by Lehnert (2011b). By style, Lehnert means how the analysis is done. This can be done by looking at the full system (global), or by looking at specific change operations (search-based) The most commonly found style within the set of approaches was **Search-based**. There were also mentions of **Global** and **Exploratory analysis**

Search-based *20/23 approaches* Almost all of the found approaches make use of Explicit rules. A big part of the analysis comes from checking different elements that are defined within these rules. The rules that are configured only look for these items, and therefore only they are analyzed. This is a cheaper option in terms of resources, as generally only these rules and their violations are checked. And no other parts of the system that have no relation to the targeted elements have to be analyzed.

Global analysis *2/23 approaches* Only two of the approaches, the one by Overeem and Jansen (2021) and the approach by Popescu et al. (2012) (which is mentioned within the former) describe how the analysis is done on the whole system independently of the current task that is executed. This is more resource intensive because it has to do analysis on more parts of the system that have a low chance of being impacted. This unfortunately often consumes more energy (Pinto and Castor, 2017) than the more directed (search-based) analysis style.

Exploratory *1/23 approaches* Vieira and Ramalho (2016) presents an exploratory analysis approach. In their paper, they call the style exploratory because it provides developers with methods to obtain the impacted elements and the impact value.

7.4 Granularity

Artifacts A distinction can be made between approaches that do analysis on difference models, models and those that do analysis on transformations. The approach by Butting et al. (2018) is concerned with artifact and model differencing. There the approach takes a starting point from which the project will be further developed. The current state is then captured. Often this can be automated. When the data is extracted, the evolution to a new version can start. Then the generator has to be executed to generate modified models. The artifact data of these two versions can be compared. In their approach, the abstract syntax of models is compared in order to identify model elements that have been added, modified, or deleted.
An example of an approach that focuses on transformations is the one by Rechau et al. (2017).

Changes Most of the found approaches focus on *atomic changes*. These contain the creation, modification, and deletion of model elements. Presenting this within Table 9 would not be interesting. This column is excluded.

Lehnert (2011b) also mentions *unstructured changes* and *compound changes*. The former focuses on changes within logs and change records and the latter looks at a remove and add operation. In a later publication, another approach (Lehnert et al., 2014) is presented which supports atomic and composite changes. Khelladi et al. (2018) mentions atomic and complex changes. By the term "complex changes" the authors mean a sequence of atomic changes. Butting et al. (2018) mentions syntactic changes as the change type that is supported. However, the addition, modification, and removal of model elements were discussed, and therefore, we think this paper focuses on atomic changes.

Results The results of the impact analysis presented in these approaches can vary. The results from using the approaches can be used by developers to make different engineering decisions that will improve the platform. Message dependency graphs of the difference models are a specific type of results (Overeem and Jansen, 2021; Popescu et al., 2012). By analyzing the difference in these graphs, developers can observe how many components were added, removed, changed, or new between two versions of a system
Another commonly found type of result is to present metrics on how much impact a change will have and whether it's a good candidate for a new version (Cho et al., 2011).
Other approaches such as Butting et al. (2018) and Demuth et al. (2016) present a list of artifacts that may have been changed.
The approach by Iovino et al. (2012) presents a list of model elements that are impacted by the changes in a metamodel. Jongeling (2016) and Lehnert et al. (2014) do something similar.

7.5 Supported languages

The most commonly supported language is UML. In various approaches, UML models can be used as an input for which difference models can be created using a tool. These difference models are then analyzed. Several other languages are supported. Examples are Domain Specific Languages for a specific system, ECore metamodels, and ATL transformations (Khelladi et al., 2018),(Vieira and Ramalho, 2016). The approach as described by Overeem and Jansen (2021) makes use of specific models that are different from standards such as UML. These artifacts are developed specifically for the AFAS Focus platform. It is expected that other LCDPs such as Mendix and Outsystems make use of platform-specific artifacts as well. Therefore, technology-agnostic approaches are an interesting research direction.

7.6 Tool support

Within the papers, various approaches were introduced. Some of the approaches presented tools or plugins which can support the CIA activity. The following paragraphs talk about these tools.

Briand et al. (2006) In their paper, the authors present a prototype. Which is the IACM tool. This tool identifies change propagation rules for the change types to manage the change between class, sequence, and state chart diagrams. A measure of distance between changed elements and potentially impacted elements is then presented to let the designer prioritize their actions.

Tröls et al. (2019) Tröls et al. made a plugin that can evaluate using consistency rules. The plugin can work in real-time to provide live synchronization of the artifact in engineering tools. The tool was built and developed based on evaluations during case studies (Demuth et al., 2016).

Jongeling (2016) The author made a tool that can be integrated with Eclipse to provide suggestions to developers. The tool can find parts of metamodels and model transformation that are likely to co-change based on a metamodel that has been changed.

Agirre et al. (2013) To automate the analysis of difference models and transformations, a JAVA and EMF Tool was developed. The tool is independent of the metamodel of the design of the system. They are however only applicable models compliant with the EMF meta-metamodel.

Di Rocco et al. (2013) A toolchain was presented that works together with TraceVis [12]. The generated artifacts through that toolchain can be used with TraceVis. TraceVis itself is a tool that can be used to visualize interactions between classes in Java programs.

Lehnert et al. (2014) The authors developed a tool to detect dependency between different types of software artifacts. The tool is extended with rule-based impact analysis.

Vieira and Ramalho (2016) A tool support module was developed using a static analyzer, tool support, and metrics component in order to present developers with an impact value of an element that is impacted after a change is made.

7.7 Future work

The papers that mention an approach, always mention future work. There are various plans to improve their approaches. As several of the approaches were only tested on a small scale, they want to validate their approach in different case studies or do more experiments. In various other papers, the authors mention that they want to provide users with better assistance for the developer in order to resolve problems in an easier way (Iovino et al., 2019; Tröls et al., 2019; Rechau et al., 2017; Müller and Rumpe, 2014). The authors also mention that they want to investigate how their current results can be combined to give a better overview. None of the found approaches mention how their approach could be implemented within LCDPs.
As can be found in Table 8 in the appendix, there are various future work directions for this domain. The directions of future work can be divided into two distinct categories, which also have sub-categories.

[12]https://www.win.tue.nl/ wstahw/tracevis/

	Empiric grounding				**Synthesizing results**			
	Literature	Case Studies	Experiments	Unspecified grounding	Present relevant CIA results	Integrate multiple CIA results	Improve correctness and completeness	Total
Overeem and Jansen (2021)	X	X						2
Tröls et al. (2019)					X			1
Cho et al. (2011)						X		1
Keller et al. (2009)			X					1
Butting et al. (2018)							X	1
Kögel (2017)		X	X					2
Popescu et al. (2012)		X						1
Gruschko et al. (2007)							X	1
Cicchetti et al. (2008)				X				1
Iovino et al. (2012)				X	X		X	3
Demuth et al. (2016)						X		1
Briand et al. (2006)		X			X			2
Lehnert et al. (2013)							X	1
Salay et al. (2016)		X						1
Jongeling (2016)					X		X	2
Rechau et al. (2017)					X			1
Iovino et al. (2019)							X	1
Agirre et al. (2013)		X					X	2
Di Rocco et al. (2013)					X			1
Müller and Rumpe (2014)					X			1
Lehnert et al. (2014)							X	1
Vieira and Ramalho (2016)						X	X	2
Khelladi et al. (2018)						X		1
Total	1 (4%)	6 (26%)	2 (9%)	2 (9%)	7 (30%)	4 (17%)	9 (39%)	

Table 7: Research directions that were identified within the selected papers. *Unspecified* means that the authors did not explicitly mentioned how they intended to ground their approach

- Empiric grounding
  - Literature
  - Case studies
  - Experiments
  - Unspecified effort into empiric grounding
- Synthesizing results
  - Present relevant results
  - Integrate multiple CIA results
  - Improve correctness and completeness

The papers and their corresponding research directions can be found in Table 7. In the following section

Empiric grounding 39% of the papers mention that more grounding needs to be done in order to validate their approach. This can be done by doing a literature review, case studies, and/or experiments. By doing these validations, the authors will know whether they are going in the right direction and they will have a better understanding of what should be changed in the future. In case studies and experiments, new scenarios can occur that will trigger the developers to improve their approach. Some of the papers do mention the need of empiric grounding, but do not suggest whether they want to do this using literature review, case studies, or experiments. We categorize this as *unspecified effort into empiric grounding.* As mentioned in Overeem and Jansen (2021), the proposed future work is to execute a systematic literature review in order to improve the framework that was proposed. This needs to be supplemented using case studies. Having stronger grounded literature sources and applying the proposed framework with other LCDPs will make it possible to validate the framework. This will allow researchers to make improvements and add detail to the framework.
Examples of papers that propose to do more literature research are Popescu et al. (2012) which want to gather more empirical data on their approaches performance during case studies. Kögel (2017) wants to evaluate his tool in various case studies and experiments. And Agirre et al. (2013) who want to apply their tool to more MDSD systems during case studies.

Synthesizing results 74% of the papers want to synthesize the results of their paper. There are three distinct categories.
The first category is **Present relevant CIA results**. An example is an approach by Tröls et al. (2019) for which the authors mention that they want to come up with a better way to present the results to the engineers. This feedback should be relevant and only appear to the relevant engineer. Briand et al. (2006) wants to improve the ranking of the impacted elements. This way the developers get to see an ordered list with the most impacted elements after a change.

The second category that can be defined within synthesizing results is **Integrate multiple CIA results**. Cho et al. (2011) for example wants to integrate the connection between design and source code metrics. This will improve the transition between software lifecycle phases when doing predictive change analysis. Vieira and Ramalho (2016) wants to incorporate more metrics that can be calculated. This can then be integrated within Eclipse plugins to improve the analysis of ATL transformations.

At last, a third category was identified for future work. This category is **Improve correctness and completeness**. Various of the found approaches only proposed a proof of concept or a prototype. Researchers aim to improve their approaches to support more types of artifacts within a wider range of systems, and also would like to make the analysis better. Butting et al. (2018) mentions that in the future, the approach should support syntactic changes that do not modify the behavior of the system. While Lehnert et al. (2013) wants to refine and add additional dependency detection results that will make the analysis more complete.

8 Analysis and Recommendations

Throughout the discussion of the findings from coding the papers we found a number of techniques being used within the approaches and some not (see Table 6). Most commonly Explicit Rules, Traceability, and Differencing are mentioned within these papers. One observation is that 6 of the 23 papers mention the use of both explicit rules and traceability as techniques that support their approach. Within the found approaches only 7 distinct techniques were found. In the following list, the techniques that were found and the number of times they were found are shown.

- Explicit rules (18)
- Traceability (8)
- Differencing (7)
- Message Dependency Graphs (2)
- Information Retrieval (1)
- History Mining (1)
- Program Slicing (1)

There were however 10 techniques mentioned in Lehnert (2011b), which are supplemented with *differencing*. Techniques that can be found in the taxonomy by Lehnert, but were not used in the approaches are:

- Call Graphs
- Execution Traces
- Program Dependency Graphs
- Probabilistic Models

Call Graphs appears to be less relevant in the MDE and LCDP domain as opposed to code-based development. This is because they consist of method and function calls that are extracted from code. MDE and LCDP are more concerned with models and less about code (unless code generated from models is the target of the call graph). **Execution traces** are another technique. This technique contains the methods and functions that have been called while the program was executed. This is again too much focused on code and less on how MDE software development works and therefore less applicable in this domain. Within **Program Dependency Graphs** a similar trend can be detected. The use of source code that is used to extract information about communication within a system makes this technique unsuitable for MDE systems.
Opposed to the three previously mentioned techniques that were unsuitable for MDE CIA, **Probabilistic Models** might be a future technique that can be used within an MDE CIA approach. By using these models, the system can detect the probability of an entity being

impacted due to the propagation of a change. Lehnert (2011a) mention that researchers managed to develop models that take into account code ownership and the change history of code. This can provide more accurate data. When adapting these probabilistic modeling techniques to (architecture) models instead of code, they might be able to develop more accurate metrics on change propagation that can be used within MDE systems and LCDPs. Overeem and Jansen (2021) mentioned the use of four different techniques. As the studied LCDP is more mature and provides more functionality, we think that it is feasible to use more CIA techniques alongside each other. And it is recommended for other LCDPs to do so as well to get more insight into the possible effects of change and evolution.
When starting out with an LCDP. It is recommended to start with **Explicit Rules** as the first technique. This technique is less resource-intensive and can be used to describe which entities are affected when a certain change is made. For new LCDPs that are often focused on a specific domain, developers and other stakeholders with domain knowledge can design these rules with their domain knowledge. Once Explicit Rules are in place, these platforms can decide to include **Traceability** as a technique to provide impact analysis. Using Traceability, different artifacts can be connected to each other in order to analyze the impact relations between them. **Differencing** was often paired with Explicit Rules. We also think that Differencing is a technique that can be included in an earlier stage. By using Differencing it is possible to find out the differences between two or more artifacts. For example two artifacts, one before and one after a change. The generated difference model can be linked to an evolving metamodel. The metamodel can afterward be connected with the artifacts (Di Rocco et al., 2013). This can provide more information.
Message Dependency Graphs can also be included if the system makes use of communication between distributed systems. They can especially play a role within distributed event-based systems. As they are highly decoupled (Popescu et al., 2012). As Overeem and Jansen (2021) have shown, these graphs can provide a lot of insight into the communication that takes place in such a system. Component graphs that can be created based on Message Dependency Graphs. And those can be used to find bugs in components and messages that do not match each other.
Information Retrieval, **History Mining** and **Program Slicing** were found as well within the found approaches. **Information Retrieval** was only mentioned in one paper and was not clearly explained. However, we think that this technique can play an interesting role as soon as an MDE system or LCDP increases in size. There are then more occasions of natural language use that can be analyzed in order to infer relationships between similar documents. These documents (models) are likely to have similar behavior which could be the same during a change. As mentioned within Almonte et al. (2021), recommender systems can become important in the future for MDE and LCDPs. When one wants to use these recommender systems, they are likely to make use of **History Mining**. Using this technique, the system looks at the history of changes and gives a developer suggestions based on past best practices. This can only be used once enough quality data is available to base these recommendations on. At last **Program Slicing** could be a technique that is possible to add. This technique presents the subset of the system that is affected by a change. In the approach that was found in this project (Salay et al., 2016). A slicing approach for heterogeneous model collections was developed which is not specific to one type of model. Although the technique does not provide much insight on how severe the effect of a change will be. It could be

useful at some point. It is not a priority to include this technique. As previously mentioned. **Probabilistic Models** might be an interesting technique to use during later stages of an LCDP as they can provide more accurate metrics when they will be applied to architecture models at some point.
The thoughts on techniques that are currently found in these approaches, and whether they should be included within an LCDP are presented within Figure 8. We used the MoSCoW method (Clegg and Barker, 1994) for this.

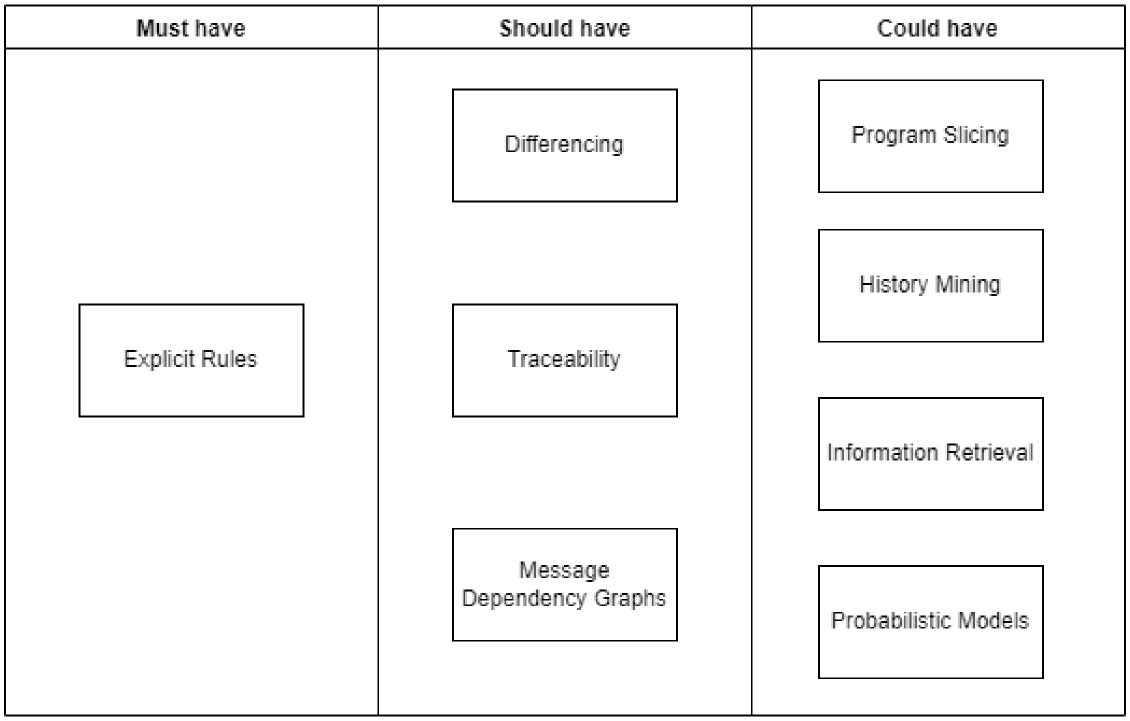

Figure 8: A diagram showing which techniques must, should and could be used within LCDPs to provide CIA

9 Discussion

The most interesting finding in our research is that explicit rules are a technique that is very important within CIA for LCDPs and other MDE systems. There are also various other techniques that are less prevalent in this domain. Various tools are available, although some are specific to a domain, or only support specific languages. This limits the reusability of these tools. There has currently not been enough research to extend this research to the LCDP domain. We however give recommendations on techniques that can be used during the early stages of the development of an LCDP and provide suggestions of techniques that are interesting additions when evolving to a larger-scale platform. There is plenty of research going on. And there are various research directions that can be taken in order to add new knowledge to this domain. This can be found in Section 7.7 and Table 8.

9.1 Limitations

Throughout the execution of this research project, we noticed several problems with our research approach to studying this domain.
There are only several authors and research groups focusing on these topics. There appear to be several leading universities that are working on model-driven engineering and its applications. Therefore, many publications are collaborations between authors with similar ideas. Notable authors were referred to several times. These were mainly Di Ruscio and Iovino which can be seen back in Figure 7. They are both in the same department at the University of L'Aquila and focus on MDE, model transformation, and evolution. It was therefore very likely to see these authors mentioned several times during the literature search.
As starting set, five papers were chosen. As our domain knowledge was limited and many papers were found on inconsistency resolution. We decided on including various papers on this topic in the starting set. Using some of these less relevant papers, not many interesting snowballing papers were found. Therefore it would be advised to select the starting papers more carefully and do exploratory research.
Snowballing might not be ideal when doing a project of this scale with only one researcher. In the future when a similar project would be done, then it is recommended to make use of tools such as Research Rabbit and Litmaps to have a better overview of what papers might have mutual references and will lead in a certain direction when executing a snowball literature search procedure. Using these tools can also give a researcher similar results as doing a manual search using reference lists within papers and managing relations between papers. A downside is that some of these tools do not have every publication indexed within their library. Also, some tools require premium memberships in order to create these graphs. Additionally, using the snowballing technique brings problems when having only one researcher. It is advised to have another researcher to check the papers for the inclusion and exclusion criteria. At some stages some mistakes might have been made which could have related in a lower quality set of found papers.
The goal was to generalize MDE approaches and find out how applicable they are within Low-Code Development Platforms. Making the comparison between how both types are built can be difficult. The found approaches do not mention LCDPs and are often using modeling languages that are not used within platforms such as AFAS Focus, Mendix, and

Outsystems. These platforms have their own way of doing this. Finding technology-agnostic approaches that can be used within these platforms was therefore a hard task.

9.2 Opportunities

One big challenge was to connect and extend the knowledge of MDE CIA to LCDP CIA. This still has some challenges. As can be seen in Table 9 and Section 7.2, many of the approaches make use of explicit rules in order to facilitate the CIA. Many of these approaches that were presented have been tested in small case studies where it was possible to define a number of these explicit rules. It is shown that these can be applied within their chosen domain and that they are sufficient. For small-scale systems with limited operations and available actions, it is good enough. Many small-scale Model Driven Engineering (MDE) systems are catered to a specific domain. For this domain, often domain-specific languages (DSLs) are developed (Whittle et al., 2013). Developers can then develop a set of explicit rules that are limited to all possible operations within this DSL. LCDPs support more functionality and provide developers with more creativity to build solutions than small-scale MDE applications. This unfortunately gives users more opportunities to do actions that can impact the platform. In Overeem and Jansen (2021) the authors describe an industry LCDP and an application developed with that LCDP. They are both far more complex and in order to facilitate CIA within this platform, various techniques are used throughout the platform. These are explicit rules, traceability links, message dependency graphs, and differencing. Other LCDPs such as Mendix make use of some kind of history mining in order to compare users' solutions against their best practices and pinpoint anti-patterns that are being used (Mendix, n.d). However, this requires these systems to support these techniques. For example, when using history mining in order to present recommendations, a system will require a dataset with the relevant source material which can be used when utilizing the technique. When working with a specific (meta)model or transformation that does not conform to standard modeling languages, a large dataset needs to be sourced in order to get useful results from history mining. It would be interesting to find out how currently used platforms deal with this. Conducting case studies with these platforms would be a good way to get an overview of how this transition can be made.

10 Conclusion

To answer the main question and the subquestions, a literature review was conducted. By researching the literature and extracting information from the papers we were able to answer our sub-questions, leading to an answer to our main question.

SQ1.1: Which challenges and solutions for Change Impact Analysis in Low-Code Development Platforms exist?

Currently, there are only a small number of papers that explicitly look into Change Impact Analysis for Low-Code Development Platforms. As LCDPs are being used as a proxy for Model-Driven Engineering systems, CIA in MDE approaches was explored as well. No synthesis of the available literature on this topic can be found using the scientific search engine Google Scholar. We can conclude that the research domain requires a more directed literature review in combination with case studies should be done to collect the available approaches that can target these applications to improve their capabilities for supporting CIA within LCDPs.

SQ1.2: What kind of technology-agnostic processes and tools exist for Change Impact Analysis in Low-Code Development Platforms? 23 approaches were found by doing a literature review. The approaches were then classified using an adapted taxonomy by Lehnert (2011b). This resulted in the commonalities and differences between the approaches. Several of the approaches presented a tool. Some tools are able to be used individually and some are plugins for general-purpose modeling environments such as Eclipse. The proposed tools that can facilitate CIA in this domain were presented within section 7.6 (Tool support). Most of the approaches that were found are dependent on a small number of CIA techniques. Mainly *Explicit rules* have been used to formalize how entities are affected when changes are made. These rules are developed by engineers who have design and domain knowledge of the application. Additionally, *Traceability* and *Differencing* were found to be common within MDE systems. Also, other techniques were found. *Message Dependency Graphs*, *History Mining*, and *Slicing* are mentioned in papers presenting an approach. These techniques are more appropriate for more mature systems and should be studied in the future to find out how and where they are most suitable.

SQ1.3: What are the open challenges for Change Impact Analysis in Low-Code Development Platforms? As a result of the literature review and coding the relevant segments into tables, the future work (Table 7 and 8) from the found papers were identified. This research agenda consists of two categories with both three sub-categories. These are

- **Empiric grounding:** The authors need to do more research to find out how applicable their approach is. This can be done by conducting **(1) Literature review**, **(2) Experiments** or by doing **(3) Case studies**. This will validate current approaches and find out what can be improved to provide a better analysis of MDE systems and LCDPs.

- **Synthesizing results:** Within this category we present the future work that is concerned with the current approach. This category has subcategories such as **(1) Present relevant CIA results to stakeholders**, in which the researcher aims to change how and what kind of feedback is shown to those who are making a change or evolving the system. **(2) Integrate multiple CIA results**, which is concerned with the integration of results and metrics in order to give users more insight into the effects of their changes. And at last **(3) Improve correctness and completeness** in which the developer wants to improve their approach by adding more functionality and supporting new artifacts and new systems.

With the answers to the sub-questions we can answer our main research question **RQ1: What are the current best practices for Change Impact Analysis in Low-Code Development Platforms?** There are several approaches available. Most of these approaches make use of explicit rules, some form of model differencing, and apply traceability to establish relations between artifacts. These techniques seem to be a golden standard and should be used within every LCDP. In later stages of an LCDP, other techniques and related approaches utilizing these techniques can be used. In the found papers techniques such as message dependency graphs, information retrieval, history mining and slicing were found. Unfortunately most of the found approaches were not applied within LCDP context. It is therefore hard to tell whether they are applicable within these platforms. We therefore think that more research needs to be done within this domain. The domain has overlap with MDE but performs on a different scale that can not be found back in many of the found approaches. Therefore the first step that should be taken is finding out how CIA is performed within a multitude of mature LCDPs such as OutSystems, Mendix, and Pega. With this information, researchers can do more directed research on the approaches that are already in use within these platforms.

For future work on this project, it would be useful to have case studies with several LCDP suppliers as well as some of the researchers that proposed the approaches mentioned within this research project. With directed questions on the architecture and utilized techniques, the difference between MDE and LCDP in terms of applicable techniques, tools, and approaches can be elicited. This literature review can serve as inspiration in order to develop these questions.

Within the research domain, researchers are encouraged to look at how results from CIA can be used. The feedback that is presented to users could be improved so that more effective engineering decisions can be made. They can also research currently used metrics that can be enhanced and combined in order to provide users with even more feedback. At last, the approaches that currently exist can be improved by increasing their completeness and correctness.

References

J. A. Agirre, L. Etxeberria, and G. Sagardui. Automatic impact analysis of software architecture migration on model driven software development. In *AMT@ MoDELS*, 2013.

K. A. Alam, R. Ahmad, A. Akhunzada, M. H. N. M. Nasir, and S. U. Khan. Impact analysis and change propagation in service-oriented enterprises: A systematic review. *Information Systems*, 54:43–73, 2015.

N. Alhirabi, O. Rana, and C. Perera. Security and privacy requirements for the internet of things: A survey. *ACM Transactions on Internet of Things*, 2(1):1–37, 2021.

L. Almonte, E. Guerra, I. Cantador, and J. De Lara. Recommender systems in model-driven engineering. *Software and Systems Modeling*, pages 1–32, 2021.

R. S. Arnold and S. A. Bohner. Impact analysis-towards a framework for comparison. In *1993 Conference on Software Maintenance*, pages 292–301. IEEE, 1993.

E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou. Introducing a ripple effect measure: a theoretical and empirical validation. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. IEEE, 2015.

C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE software*, 20(5):36–41, 2003.

M. N. Azadani and A. Boukerche. Driving behavior analysis guidelines for intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

S. Black. Computing ripple effect for software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(4):263–279, 2001.

A. C. Bock and U. Frank. Low-code platform. *Business & Information Systems Engineering*, 63(6):733–740, 2021.

S. A. Bohner and R. S. Arnold. *Software Change Impact Analysis.* Wiley-IEEE Computer Society Pr, 1996.

P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583, 2007.

D. Breuker. Towards model-driven engineering for big data analytics–an exploratory analysis of domain-specific languages for machine learning. In *2014 47th Hawaii International Conference on System Sciences*, pages 758–767. IEEE, 2014.

L. C. Briand, Y. Labiche, L. O'Sullivan, and M. M. Sówka. Automated impact analysis of UML models. *Journal of Systems and Software*, 79(3):339–352, 2006.

A. Butting, S. Hillemacher, B. Rumpe, D. Schmalzing, and A. Wortmann. Shepherding model evolution in model-driven development. In *Modellierung (Workshops)*, pages 67–77, 2018.

J. Cabot. Positioning of the low-code movement within the field of model-driven engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–3, 2020.

N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, 13(1):3–30, 2001.

H. Cho, J. Gray, Y. Cai, S. Wong, and T. Xie. Model-driven impact analysis of software product lines. In *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, pages 275–303. IGI Global, 2011.

A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 222–231. IEEE, 2008.

D. Clegg and R. Barker. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

I. Committee. IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). Los Alamitos. *CA IEEE Comput. Soc*, 1990.

B. Costa, J. Bachiega Jr, L. R. de Carvalho, and A. P. Araujo. Orchestration in fog computing: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 55(2):1–34, 2022.

A. De Lucia, F. Fasano, and R. Oliveto. Traceability management for impact analysis. In *2008 Frontiers of Software Maintenance*, pages 21–30. IEEE, 2008.

A. Demuth, R. Kretschmer, A. Egyed, and D. Maes. Introducing traceability and consistency checking for change impact analysis across engineering tools in an automation solution company: an experience report. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 529–538. IEEE, 2016.

J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio. Traceability visualization in metamodel change impact detection. In *Proceedings of the Second Workshop on Graphical Modeling Language Development*, pages 51–62, 2013.

J. Di Rocco, D. Di Ruscio, H. Narayanankutty, and A. Pierantonio. Resilience in Sirius Editors: Understanding the Impact of Metamodel Changes. In *MoDELS (Workshops)*, pages 620–630, 2018.

D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 2021.

D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, pages 1–10, 2022.

I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. De Almeida. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, 56(10):1183–1199, 2014.

G. Dushnitsky and B. K. Stroube. Low-code entrepreneurship: Shopify and the alternative path to growth. *Journal of Business Venturing Insights*, 16:e00251, 2021.

S. Farshidi, S. Jansen, and S. Fortuin. Model-driven development platform selection: four industry case studies. *Software and Systems Modeling*, 20(5):1525–1551, 2021.

A. M. Fernández-Sáez, M. R. Chaudron, and M. Genero. An industrial case study on the use of uml in software maintenance and its perceived benefits and hurdles. *Empirical Software Engineering*, 23(6):3281–3345, 2018.

E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho, et al. Evolving software product lines with aspects. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 261–270. IEEE, 2008.

U. Frank, P. Maier, and A. Bock. Low code platforms: promises, concepts and prospects. a comparative study of ten systems. Technical report, ICB-Research Report, 2021.

I. Galvao and A. Goknil. Survey of traceability approaches in model-driven engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 313–313. IEEE, 2007.

B. Gruschko, D. Kolovos, and R. Paige. Towards synchronizing models with evolving metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution*, page 3. Citeseer, 2007.

I. Hadar, P. Soffer, and K. Kenzi. The role of domain knowledge in requirements elicitation via interviews: an exploratory study. *Requirements Engineering*, 19(2):143–159, 2014.

T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2011.

H. Harrison, S. J. Griffin, I. Kuhn, and J. A. Usher-Smith. Software tools to support title and abstract screening for systematic reviews in healthcare: an evaluation. *BMC medical research methodology*, 20(1):1–12, 2020.

A. E. Hassan and R. C. Holt. Predicting change propagation in software systems. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 284–293. IEEE, 2004.

E. Hassler, J. C. Carver, D. Hale, and A. Al-Zubidy. Identification of slr tool needs–results of a community workshop. *Information and Software Technology*, 70:122–129, 2016.

R. Hebig, D. E. Khelladi, and R. Bendraou. Approaches to co-evolution of metamodels and models: A survey. *IEEE Transactions on Software Engineering*, 43(5):396–414, 2016.

A. Hinderks, F. José, D. Mayo, J. Thomaschewski, and M. J. Escalona. An slr-tool: Search process in practice: A tool to conduct and manage systematic literature review (slr). In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 81–84. IEEE, 2020.

IEEE. ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, pages 1–58, 2006. doi: 10.1109/IEEESTD.2006.235774.

F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, and A. Pierantonio. Low-code engineering for internet of things: a state of research. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–8, 2020.

L. Iovino, A. Pierantonio, and I. Malavolta. On the impact significance of metamodel evolution in mde. *J. Object Technol.*, 11(3):3–1, 2012.

L. Iovino, A. Rutle, A. Pierantonio, and J. Di Rocco. Query-based impact analysis of metamodel evolutions. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 458–465. IEEE, 2019.

C. Ivey and J. Crum. Choosing the right citation management tool: Endnote, mendeley, refworks, or zotero. *Journal of the Medical Library Association: JMLA*, 106(3):399, 2018.

R. Jongeling. *Change Impact Analysis in Model Driven Software Engineering Ecosystems.* PhD thesis, Master's thesis, Eindhoven University of Technology, the Netherlands, 2016.

A. Keller, H. Schippers, and S. Demeyer. Supporting inconsistency resolution through predictive change impact analysis. In *Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*, pages 1–10, 2009.

W. Kessentini and V. Alizadeh. Semi-automated metamodel/model co-evolution: a multi-level interactive approach. *Software and Systems Modeling*, pages 1–24, 2022.

D. E. Khelladi, R. Kretschmer, and A. Egyed. Change propagation-based and composition-based co-evolution of transformations with evolving metamodels. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 404–414, 2018.

B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

S. Kögel. Recommender system for model driven software development. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 1026–1029, 2017.

D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige. Different models for model matching: An analysis of approaches to support model differencing. In *2009 ICSE Workshop on Comparison and Versioning of Software Models*, pages 1–6. IEEE, 2009.

S. Lehnert. *A review of software change impact analysis*. Citeseer, 2011a.

S. Lehnert. A taxonomy for software change impact analysis. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, pages 41–50, 2011b.

S. Lehnert, M. Riebisch, et al. Rule-based impact analysis for heterogeneous software artifacts. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 209–218. IEEE, 2013.

S. Lehnert, M. Riebisch, et al. Analyzing model dependencies for rule-based regression test selection. *Modellierung 2014*, 2014.

T. C. Lethbridge. Low-code is often high-code, so we must design low-code platforms to enable proper software engineering. In *International Symposium on Leveraging Applications of Formal Methods*, pages 202–212. Springer, 2021.

B. Li, X. Sun, H. Leung, and S. Zhang. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, 23(8):613–646, 2013.

M. L. Marinho, S. Sampaio, T. Lima, and H. P. Moura. A guide to deal with uncertainties in software project management. *arXiv preprint arXiv:1411.1920*, 2014.

Mendix. Model consistency, Jul 2021. URL `https://www.mendix.com/evaluation-guide/app-lifecycle/model-consistency/`.

Mendix. MxAssist Performance bot, n.d. URL `https://docs.mendix.com/refguide/mx-assist-performance-bot/`.

T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in software evolution. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pages 13–22. IEEE, 2005.

K. Müller and B. Rumpe. A model-based approach to impact analysis using model differencing. *arXiv preprint arXiv:1406.6834*, 2014.

A. C. Neto, R. Bonifácio, M. Ribeiro, C. E. Pontual, P. Borba, and F. Castor. A design rule language for aspect-oriented programming. *Journal of Systems and Software*, 86(9): 2333–2356, 2013.

C. Okoli. A guide to conducting a standalone systematic literature review. *Communications of the Association for Information Systems*, 37(1):43, 2015.

OutSystems. How outsystems solves the problem: Evaluation guide, n.d. URL `https://www.outsystems.com/evaluation-guide/how-outsystems-solves-the-problem/`.

M. Overeem and S. Jansen. An exploration of the 'it' in 'it depends': Generative versus interpretive model-driven development. In *MODELSWARD*, pages 100–111, 2017.

M. Overeem and S. Jansen. Proposing a framework for impact analysis for low-code development platforms. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 88–97. IEEE, 2021.

M. Overeem, M. Mathijssen, and S. Jansen. Api-m-famm: A focus area maturity model for api management. *Information and Software Technology*, page 106890, 2022.

M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan, et al. The prisma 2020 statement: an updated guideline for reporting systematic reviews. *International Journal of Surgery*, 88:105906, 2021.

G. Pinto and F. Castor. Energy efficiency: a new concern for application software developers. *Communications of the ACM*, 60(12):68–75, 2017.

D. Popescu. Helios: impact analysis for event-based components and systems. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2 (pp. 531-532).*, pages 531–532, 01 2010. doi: 10.1145/1810295.1810466.

D. Popescu, J. Garcia, K. Bierhoff, and N. Medvidovic. Impact analysis for distributed event-based systems. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 241–251, 2012.

V. Rajlich. Software evolution and maintenance. In *Future of Software Engineering Proceedings*, pages 133–144. Association for Computing Machinery, 2014.

P. Ralph et al. Empirical standards for software engineering research. *arXiv preprint arXiv:2010.03525*, 2021.

T. Rechau, N. Silva, M. M. d. Silva, and P. Sousa. A tool for managing the evolution of enterprise architecture meta-model and models. In *European, Mediterranean, and Middle Eastern Conference on Information Systems*, pages 68–81. Springer, 2017.

T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley. Generalizing the analysis of evolutionary coupling for software change impact analysis. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 201–212. IEEE, 2016.

L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack. An analysis of approaches to model migration. In *Proc. Joint MoDSE-MCCM Workshop*, pages 6–15, 2009.

A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE, 2020.

R. Salay, S. Kokaly, M. Chechik, and T. Maibaum. Heterogeneous megamodel slicing for model evolution. In *ME@ MoDELS*, pages 50–59, 2016.

S. Sancar Gozukara, B. Tekinerdogan, and C. Catal. Obstacles of on-premise enterprise resource planning systems and solution directions. *Journal of Computer Information Systems*, 62(1):141–152, 2022.

R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler. Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences*, 10(1):12, 2020.

A. D. Sandro, R. Salay, M. Famelis, S. Kokaly, and M. Chechik. Mmint: A graphical tool for interactive model management. In *P&D@MoDELS*, 2015.

B. Tekinerdogan and E. Er. Change impact analysis of model-driven development systems using evolution scenario templates. *Models and Evolution*, page 111, 2009.

M. A. Tröls, A. Mashkoor, and A. Egyed. Multifaceted consistency checking of collaborative engineering artifacts. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 278–287. IEEE, 2019.

R. van de Schoot, J. de Bruin, R. Schram, P. Zahedi, J. de Boer, F. Weijdema, B. Kramer, M. Huijts, M. Hoogerwerf, G. Ferdinands, et al. An open source machine learning framework for efficient and transparent systematic reviews. *Nature Machine Intelligence*, 3(2): 125–133, 2021.

A. Vieira and F. Ramalho. Towards measuring the change impact in atl model transformations. *International Journal of Software Engineering and Knowledge Engineering*, 26(02): 153–181, 2016.

P. Vincent, K. Iijima, M. Driver, J. Wong, and Y. Natis. Magic quadrant for enterprise low-code application platforms. *Gartner report*, 2019.

R. Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019.

J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2013.

C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, 2014.

J. F. Wolfswinkel, E. Furtmueller, and C. P. Wilderom. Using grounded theory as a method for rigorously reviewing literature. *European journal of information systems*, 22(1):45–55, 2013.

S. S. Yau, J. S. Collofello, and T. MacGregor. Ripple effect analysis of software maintenance. In *The IEEE Computer Society's Second International Computer Software and Applications Conference, 1978. COMPSAC'78.*, pages 60–65. IEEE, 1978.

11 Appendix

Paper	**Mentioned future work**
Overeem and Jansen (2021)	The authors propose doing systematic literature review and to validate framework with other LCDP providers as future research directions.
Tröls et al. (2019)	The authors would like to expand on their mechanism for distributing consistency feedback. This is currently often not interesting to engineers. In the future they only want to provide relevant information relevant to the engineers' focus.
Cho et al. (2011)	The authors want to increase the connection between design metrics and the source code metrics. This can improve the transition between lifecycle phases as related to analysis of predictive change. If lifecycle metrics are combined, then many new areas of research can be investigaed. The authors also want to explore the extention of the traceability relation functionality to provide more query-specific opportunities.
Keller et al. (2009)	They want to refine the algorithm. Improvements could be to assign weights to modification operations. Also the extend of the distance of an impacted element can be examined better. The authors also want their approach to be validated by doing more experiments with different and more inconsistencies.
Butting et al. (2018)	Future work would be to let the approach support syntactic changes that do not modify the behavior. This can be done using semantic differencing.
Kögel (2017)	Developing the tool as it is proposed. Various phases are described for evaluations and integrations.
Popescu et al. (2012)	Gathering more empirical data with different applications to increase the confidence in the approaches utility. Assess the applicability to distributed systems that use varying filtering mechanisms beyond type-based.
Gruschko et al. (2007)	Investigating the possibility of embedding manual migration steps.
Cicchetti et al. (2008)	Their future work includes the implementation of the power model construction the difference refinement depends on. Additionally, more systematic validation of the approach is necessarily to encompass a larger population of models and metamodels. They also plan to investigate how the works related to change impact analysis can be adapted and used in MDE to support the co-evolution of metamodels and corresponding models.
Iovino et al. (2012)	As future work, the authors want to investigate how to increase the degree of automation in the adaptation by using the outcome of the process. They also want to investigate how to parametrize current modelling platforms so that the modeller can have user-defined relationships. At last they want to evaluate the graphical user interface of their approach and visualize traceability links

Demuth et al. (2016)	For future work, the authors plan to investigate possibilities of making the integration of such tools easier so that the effort for establishing traceability and consistency checking technologies can be reduced and the acceptance and application of these technogies increases.
Briand et al. (2006)	Future work includes performing additional case studies. This will also be used in an attempt to associate probabilities with impact analysis rules based on empirical data. This would allow us to further refine our ranking of impacted elements according to their likelihood of actually requiring change.
Lehnert et al. (2013)	The authors want a more systematic investigation of dependency types. They also want to refine and add additional dependency detection rules to elicitate further, yet missing dependencies, as revealed by their case study. At last, they want to extend the dependency analysis of Java source code to the level of program statements, to allow for more fine-grained couplings with dynamic UML models.
Salay et al. (2016)	The authors want to develop tooling for the algorithm using the Model Management INTeractive (MMINT) framework and plan to use it to conduct more extensive case studies to better understand the strengths and weaknesses of the approach.
Jongeling (2016)	As future work, the author proposes supporting other co-evolving artifacts that can be found in graphical and textual editors. As well as model-to-text transformations and M1 models. Other potential future work could be to allow developers to indicate suggestions as correct or incorrect so that the suggestions can be improved.
Rechau et al. (2017)	Being an ongoing research, a final and fully functional version of the tool is currently being developed. The authors consider as future effort the possibility of implementing an interactive help feature to offer the user proper guidance and more visual refinements regarding the change impact analysis.
Iovino et al. (2019)	The proposed future work is to extend the approach by including extra artifact types (next to models and transformations) such as code generators
Agirre et al. (2013)	They want to apply the tool to other MDSD systems. They also want to extend the tool to deal with more difference types. When more types of software evolution and architecture migration situations are analyzed, new refinements operations and transformation rules patterns can be used.
Di Rocco et al. (2013)	The authors plan on live monitoring of change impact. As soon as a change is made, impact information on the related artifacts in the ecosystem shold be displayed. This requires an operation recorder to register model differences.

Müller and Rumpe (2014)	For future work, they plan to investigate for which checklist hints the authors can check automatically whether the user really implemented the particular change. They also want to extend their approach by giving more detailed checklists with additional explanations and a brief checklist. The checklist can also be optimized by supporting ticking off the checklist and connecting it with issue tracking systems such as JIRA.
Lehnert et al. (2014)	Their future work targets an extension of their impact rules to cover the concrete test scripts. Furthermore, they plan to analyze how risk, cost, and fault severity-based approaches can be integrated with their approach for further test prioritization.
Vieira and Ramalho (2016)	Incorporating more metrics in the calcChangeImpactValue() function. Look at CIA in a chain of information. Support more than one atomic change at a time and integrate the approach with currently used Eclipse plugins for ATL transformations.
Khelladi et al. (2018)	As future work, the authors plan to explore further the composition of resolutions and investigate what is the minimum set of resolutions that can be sufficient for a wide range of compositions. They further plan to use meta-heuristic algorithms to search for possible useful compositions of resolutions.

Table 8: A description of the future work as mentioned in the found approaches.

ID	Paper	Scope	Technique	Style	Granularity Artifacts	Granularity Results	Supported languages	Output
1	Overeem and Jansen (2021)	Code - Static, Models - Architecture	Traceability, Differencing, Message Dependency Graphs, Explicit rules	Global	OEM diff all levels, Mergelog attributes, traceability links on generator components, component graph on architectural components	Diffs, message dependency graphs	Built to be specifically for AFAS' metamodel and architecture	Implemented in LCDP
2	Tröls et al. (2019)	Code - Dynamic, Models - Architecture	Traceability, Explicit rules	Search-based	Uniform data representation	Results within Consistency Rule Evaluation Artifact.	UML, Excel, Eclipse, Eplan P8, Creo	Plugin
3	Cho et al. (2011)	Models - Architecture	Explicit rules, Traceability	Search-based	DSL	Design performance metrics	UML, Traceability links (XML)	
4	Keller et al. (2009)	Models - Architecture	Explicit rules	Search-based	Model-elements	Graphs showing results of actions	UML2 metamodel	Algorithm
5	Butting et al. (2018)	Models - Architecture	Differencing	Search-based	Specific models to Models-architectural	A list of generated artifacts that may have been changed	Monticore software architecture models	
6	Kögel (2017)	Models - Architecture	History mining	Search-based	Model transformations	Recommendations	Ecore metamodels	Tool in-progress
7	Popescu et al. (2012)	Code - Static	Message Dependency Graphs	Global	Source code	A message dependence graph	Java, C++ or C#	Technique
8	Gruschko et al. (2007)	Models - Architecture	Traceability, Explicit rules	Search-based	Model transformations	The set of changes	ECore metamodels	Prototype tool
9	Cicchetti et al. (2008)	Models - Architecture	Explicit rules, Differencing	Search-based	Difference models		KM3 metamodels	Prototype tool
10	Iovino et al. (2012)	Models - Architecture	Explicit rules	Search-based	EMFText syntax specifications	Set of elements that may be impacted by changes.	AM3 megamodels	Implementation within AMMA platform
11	Demuth et al. (2016)	Code - Dynamic, Models - Architecture	Explicit rules, Traceability	Search-based	Sheet data, electrical models, source code, and components	Overview of affected artifacts	Excel, EEP8 models, Eclipse with Java	Tool
12	Briand et al. (2006)	Models - Architecture	Explicit rules	Search-based	Components of UML models	Measure of distance	UML	Tool
13	Lehnert et al. (2013)	Models - Architecture	Explicit rules, Traceability	Search-based	Component, Package, Class/Interface, Method, Method-parameter, Attribute, Use case	Impacted elements	UML models, Java source code, JUnit tests. Combined into EMF metamodel	Tool.

14	Salay et al. (2016)	Models - Architecture	Slicing, Differencing, Traceability		Components	Megamodel fragments showing what is impacted	UML Diagram	Algorithm
15	Jongeling (2016)	Models - Architecture	Information retrieval, Explicit rules	Search-based	DSL, Model-to-model transformations	The elements that are likely to co-change based on changed metamodel	DSL	Tool
16	Rechau et al. (2017)	Models - Architecture	Explicit rules	Search-based	Transformations of EA metamodel and models	Quantitative assessment of instances impacted	EA metamodels	Tool
17	Iovino et al. (2019)	Models - Architecture	Explicit rules	Search-based	Models + Transformations	List of impacted instances	Ecore metamodels, ATL transformation language	
18	Agirre et al. (2013)	Models - Architecture	Differencing, Traceability, Explicit Rules	Search-based	Traceability model, output differences model, weaving model, UML design and Simplec model	Adaptation goal model	EMF and ATL	Tool
19	Di Rocco et al. (2013)	Models - Architecture	Traceability, Differencing	Search-based	Transformation elements in relation with changed metamodel elements	Visualization of dependencies	ECore metamodels, ATL	Extended another Tool[13]
20	Müller and Rumpe (2014)	Models - Architecture	Explicit rules, Differencing	Search-based	Difference model of UML class diagrams	Checklists with impacted properties	UML class diagrams	Tool
21	Lehnert et al. (2014)	Models - Architecture	Explicit rules	Search-based	BPMN/UML models on which changes have been applied	Impacted test elements.	BPMN, UML, UML Testing Profile.	Tool
22	Vieira and Ramalho (2016)	Models - Architecture	Explicit rules	Exploratory	ATL model transformation elements	An impact value of a given change and the impacted elements of that specific change	ATL model transformation language	Tool
23	Khelladi et al. (2018)	Models - Architecture	Explicit Rules	Search-based	ETL AST classes	Sorted repair suggestions	Ecore/EMF metamodels, ETL Epsilon transformations	Tool

[13]`https://www.win.tue.nl/~wstahw/tracevis/`

Table 9: The characteristics of the approaches that were found using the snowballing technique (Wohlin, 2014). The characteristics were inspired by the Taxonomy of Lehnert (2011b). The yellow annotations present papers the characteristics that are less commonly found compared to other papers.