Proposing a Framework for Impact Analysis for Low-Code Development Platforms

1st Michiel Overeem
AFAS Software
Leusden, The Netherlands
michiel.overeem@afas.nl
0000-0003-4807-4124

2nd Slinger Jansen
Utrecht University
Utrecht, The Netherlands
slinger.jansen@uu.nl
0000-0003-3752-2868

***Abstract*—Low-code development platforms accelerate software development by facilitating end-user programming. Through higher-level abstractions citizen developers are enabled to develop increasingly complex software systems. While this improves productivity and efficiency it also introduces new challenges in the development process.**

The evolution of the low-code development platform and the applications built on top of it is one of those challenges. Understanding the impact of changes on the software system is crucial for both the maintenance as well as the improvement of running software. Citizen developers can be supported by direct feedback that reflects how their changes impact the system. Professional developers can use the impact analysis to correctly migrate existing data. Finally, the operations engineers that are responsible for the availability of the platform and the applications can plan seamless upgrades of new versions. Impact analysis should be at the foundations of the development of low-code development platforms.

This paper proposes the Impact Analysis for Low-Code Development Platforms framework, a conceptual framework that supports the discussion, research, and implementation of impact analysis. The proposed framework describes the different subsystems and artifacts in a low-code development platform, the different types of professionals involved, and how these professionals can use impact analysis to support their engineering decisions. Through a descriptive case study we discuss the role of impact analysis in an industry low-code development platform. Through the feedback acquired by impact analysis, professionals can stay in control of the evolution of both the applications as well as the low-code development platform itself.

***Index Terms*—Low-Code Development Platform, Co-Development, Citizen/end-user Development, Change Impact Analysis, Evolution**

I. Introduction

A trend that is still growing and gaining traction are low-code development platforms (LCDPs) [1]. These LCDPs facilitate end-user programming for *citizen developers*, people without formal programming education that develop software, through Model-driven Development (MDD). Ultimately, the goal of these platforms is to enable citizen developers to build full-stack software applications [2].

This research was supported by the NWO AMUSE project (628.006.001): a collaboration between Vrije Universiteit Amsterdam, Utrecht University, and AFAS Software in the Netherlands. See amuse-project.org for more information.

While this may have started with relatively simple applications that automated one task, the applications targeted by LCDPs are becoming increasingly complex. From enterprise services [3], Internet of Things [4] to the enablers of digital transformations in manufacturing industry [5]. This growth and evolution of LCDPs into supporting more different kinds of systems and more complex systems also gives rise to new challenges. Challenges that were once the domain of software engineers and operations engineers[1] are now becoming challenges for these citizen developers.

Software evolution is one of these challenges. While it starts with the design, development, and release of an application build with a LCDP, the citizen developer will quickly become aware of the challenge of software evolution. As these applications grow and become more complex, companies will also depend more and more on them. Quality characteristics of these applications become more important, and the impact of changes needs to be predicted before they are made, or released into production. However, software evolution for the professional developer and operations engineer also becomes more challenging. A large part of the applications is developed by a new type of professional, the citizen developer. The professional developers and operations engineers are not always aware of the changes made by the citizen developers as they might be part of a different team, or even a different company.

Impact analysis provides all three types of professionals with the needed feedback. The analysis of how changes impact other parts of the system and the running applications benefits the professionals in making well supported engineering decisions. We propose a framework to support the discussion and research of impact analysis in low-code development platforms. First we describe the three types of professionals (citizen and professional developers and operations) involved, the subsystems of a LCDP, and the involved artifacts. The *Impact Analysis for Low-Code Development Platforms* itself comprises of the execution of change analysis, the collecting of change analysis results, and the deduction of impact

[1]Different titles are used for these roles, such as 'DevOps engineers', 'Platform Engineers', and 'Site Reliability Engineers'. We use the term *operations engineers* to refer to the people and/or teams that are responsible for technical management and maintenance of software systems running in production.

observations that are presented to the involved professionals. Through a case study we show how this framework can be applied, and how impact analysis can result in feedback for those professionals involved in the development of LCDPs and applications. We report on a decade of development of an industry LCDP and application, with 18 months of operational usage. Different forms of impact analysis are used to facilitate control over the evolution of the system and support engineering decisions made by the involved professionals.

The research approach is explained in Section II. In Section III we discuss LCDPs in general, and propose the Impact Analysis for Low-Code Development Platforms framework. The case study is described in Section IV and analysed in Section V. Section VI discusses the case study, the research, and future work. Related work is discussed in Section VII. Our conclusions are stated in Section VIII.

II. Research Approach

In this research the role of impact analysis in a low-code development platform (LCDP) is discussed. A descriptive case study is conducted at AFAS Software, during the development of an industry LCDP. The LCDP is used to develop a new ERP system. Currently the platform itself is used only internally, while the resulting ERP system is offered as a Software-as-a-Service (SaaS) product. The research and development of the LCDP *AFAS Focus* started in 2010, but a separate development team was not created until 2013. From 2013 until 2019 the development team grew from 10 people to 50 people, including citizen developers, testers, and professional developers. End of 2019 the first customers went live in a private beta program, and in the second half of 2021 the product was publicly launched.

The first author is part of the research and development team at AFAS Software since 2011, first as Software Architect, but from 2013 as Lead Software Architect. During the development of the LCDP, the first author was jointly responsible for the development of the LCDP architecture, the data conversion techniques, and the upgrade strategy. The second author is only involved in the project as an independent external researcher since 2015. Our research is based on observations and contributions made by the first author during the development of AFAS Focus. The challenges, the proposed framework, and the results are discussed with the second author since the start of the involvement of the second author. These discussions have bend the inward look from AFAS Focus to LCDPs in general. The research and development of AFAS Focus has resulted in contributions such as a comparison of model execution [6], a framework for data migration [7], and a maturity model for API management that is applied to LCDPs [8]. The *Impact Analysis for Low-Code Development Platforms* framework presented in Section III is based on the acquired knowledge, research contributions, and experience accumulated throughout these years.

III. Impact Analysis for Low-Code Development Platforms

LCDPs accelerate the development of applications by decreasing the amount of hand-coding required [1]. This is accomplished by making software development easier by raising the abstraction level through model-driven development. The higher abstraction label also makes it possible to develop software for people without a formal software development background: the citizen developer. These citizen developers are trained professionals with domain knowledge that are enabled to develop solutions for their domain specific problems.

The *Impact Analysis for Low-Code Development Platforms* framework is depicted in Figure 1. It comprises of the execution of change analysis, the collecting of change analysis results, and the deduction of impact observations that are presented to the involved professionals. We distinguish three types of professionals involved in the impact analysis. First of all the citizen developers, who uses the LCDP to develop application. Secondly the LCDP developers, responsible for the development of the LCDP itself. Finally, the operations engineers who are responsible for keeping the applications and LCDP available and responsive. The general structure of a LCDP consists of three subsystems and three types of artifacts. These subsystems and artifacts are analysed for changes, resulting in *change analysis results*, in the form of diffs. These diffs result in impact observations that inform the professionals and support their engineering decisions. The evolution of the LCDP and the applications can be controlled through the created feedback. The remainder of this Section explains the subsystems and artifacts, and the impact analysis process in more detail.

The first subsystem is the *model designer*. The designer is the Integrated Development Environment (IDE) offered to the citizen developer. It provides an interface for the development of the model(s) and includes help, and feedback. The first type of artifact is the *meta-model*. The meta-model describes the model elements and thus the capabilities of the model. A LCDP can utilize multiple models, and thus multiple meta-models, targeting different aspects of an application, such as the data, the business logic, and the user interface. The designed models are the second type of artifact. These are parsed, processed, and transformed by the *model transformations* subsystem. This system transforms the model into a *run-time model*. The run-time model can take different forms, depending on the LCDP implementation. It could either be an intermediate model specific to the LCDP, or a general purpose model (or programming language). The *platform* subsystem contains all the features and infrastructure necessary to execute the run-time model. It contains service frameworks, data access libraries, and other functionality present in the resulting application that is not dependent on the model. The platform and run-time model are deployed to the *runtime* that executes the two subsystems.

Changes are collected by analysing the subsystems and artifacts in a LCDP. This is done before these subsystems

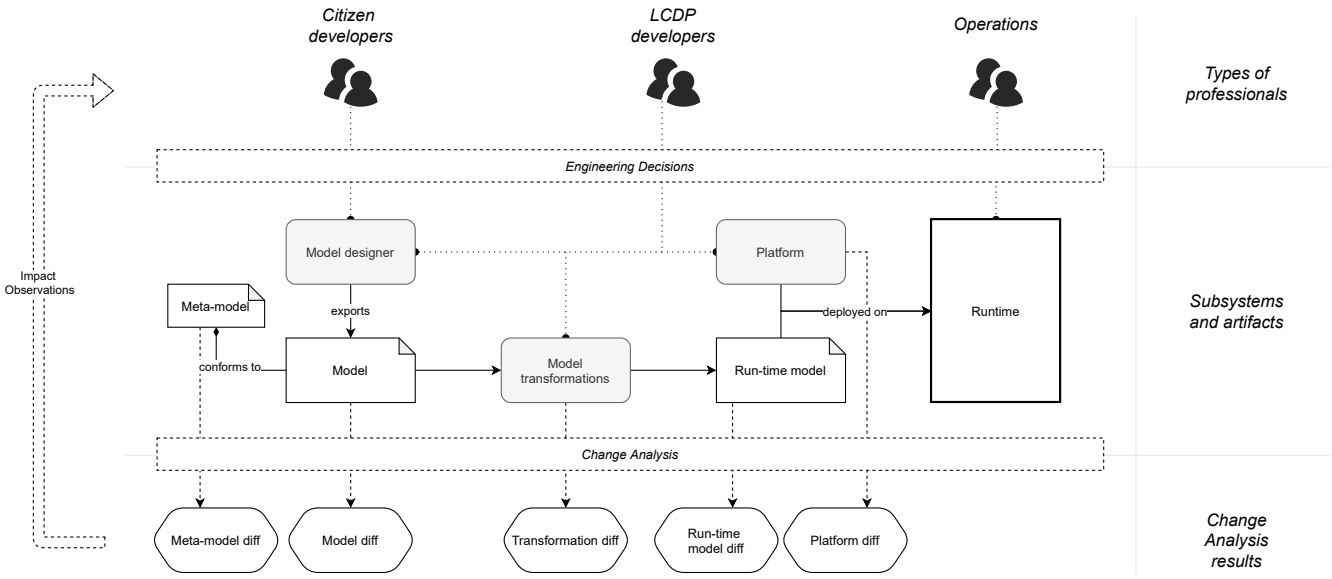

Fig. 1. The *Impact Analysis for Low-Code Development Platforms* framework comprises of the execution of change analysis, the collecting of change analysis results, and the deduction of impact observations. Changes occur in different parts of the system, but are collected through the change analysis. The resulting changes are represented in diffs. These diffs can be analysed for impact, resulting in impact observations. The results of impact analysis are used to inform the professionals. They can use this feedback to improve the platform, to redesign existing solutions, or decide to revert certain changes. Furthermore, team coordinators can use the impact framework to orchestrate the process of LCDP evolution.

and/or artifacts are deployed on the runtime. The results are expressed in *change analysis results*, representing the changes made, and are generally expressed in a *diff*. From these results the impact that changes have on the system can be derived. The impact observations support the professionals in making engineering decisions on the evolution of the LCDP and the applications. These observations can block the release of new versions, or adjust the future roadmap.

The specifics of the change analysis, the representations chosen for the change results, and the impact observations are LCDP specific. Therefore the remainder of this section will give example suggestions to illustrate the process expressed in the Impact Analysis for Low-Code Development Platforms framework. These examples are given by revisiting the three types of professionals that are responsible for the development and operations of the applications developed on the LCDP and the LCDP itself.

First of all the citizen developers who use the LCDP to create solutions. In commercially offered LCDPs these developers are the customer, or work for the customer, of the LCDP. They use the *model designer* as the primary way of interacting with the LCDP. It allows them to create an application by expressing their solution constrained by the meta-model offered by the LCDP. To facilitate rapid application development, technical details will be hidden from them. Citizen developers can, based on the impact observations, receive feedback on the quality of the model(s) that they have developed. An example is given: *A change to the model leads to a far bigger change in the run-time model, because a specific model element represents a complex piece of run-time functionality. There is, however, a different solution possible that solves the problem and leads to a smaller change in the run-time model. This solution has preferable characteristics: a smaller impact on the running application. This impact observation is generated by analysing the model diff and the run-time model diff, and linking the changes made to the model to those in the run-time model.* In general the change results can be analysed by specific rules, maybe a recommender system [9], [10], and suggest alternative solutions. Such a system requires the knowledge to link model changes to run-time model changes and to characterize these changes depending on their impact on the runtime by incorporating Software Operational Knowledge.

Secondly the LCDP developers, they develop and maintain the three subsystems in a LCDP. The LCDP developers can use the feedback to optimize the platform for the both the citizen developers as well as the operations team. Model changes can be analysed to identify often neglected features or popular features. These observations can then be incorporated in the roadmap to optimize the meta-model and the designer. Meta-model changes can be analysed to find the places in the designer impacted by these changes. The run-time model and platform changes serve as the source for the runtime impact analyses. This analysis points to parts of the system that could harm the operational characteristics of the application. These changed parts can then be reverted before releasing the new version.

Finally the operations engineers who are responsible for keeping the applications and LCDP available and responsive. If the LCDP is an internal platform, such as in our case study, there will be a single group of operations engineers. However,

if the LCDP is commercially offered solution there will be probably two groups of operations engineers. The operations engineers that support the platform itself are responsible for the availability of the LCDP: the model designer, the model transformations, and the runtime. The operations engineers that are part of the customer company will focus on the availability of the applications, using the features offered by the LCDP to do so. This team supports the release of new versions, while monitoring the running environments. These operations engineers benefit from the impact observations in planning the upgrades of the platform and/or new applications. The run-time model and platform changes can tell them if they require more runtime resources.

The proposed framework describes the process of impact analysis for LCDPs in generic terms. LCDP providers and consumers should instantiate this framework for their own specific case. However, the *taxonomy for software change impact analysis* [11] can help. The taxonomy lists eight criteria to classify impact analysis approaches.

- *The scope of the analysis*: does the impact analysis operates on code, models, or other artifacts. The framework focuses on static analysis of code and models, and does not incorporate dynamic aspects collected from a running system.
- *The granularity of the analysis*: what level of detail is analyzed and reported. Collected changes can be aggregated, or only collected on a certain level. This determines the impact observations that can be made.
- *The utilized technique*: examples of techniques are call graphs, execution traces, or message dependency graphs. The best fitting technique depends on the kind of LCDP and its architecture.
- *The style of the analysis:* is the analysis global, search based, or exploratory.
- *Tool support*: which tools support the chosen approach.
- *Supported languages*: which programming or modelling languages are supported by an approach. While the model in a LCDP is custom developed, providers can benefit from standard tooling for language and model engineering. The run-time model could also be a standard programming language or intermediate model that is supported by available tools.
- *Scalability*: how scalable is the impact analysis approach.
- *Experimental results*: is the approach tested and shown to be successful.

These criteria list the variability that is present in the Impact Analysis for Low-Code Development Platforms framework. LCDP providers and consumers need to chose an existing approach or design their own approach for impact analysis when applying the framework.

IV. Case Study

We conduct a case study on how impact analysis is applied on an industry LCDP at AFAS Software. This case study describes the process in which impact analysis is applied and from which we derived the Impact Analysis for Low-Code Development Platforms framework. AFAS Software is a Dutch vendor of ERP software based in Leusden, the Netherlands (with additional offices in Belgium, Curaçao, and Aruba). The privately held company currently employs over 500 people and generated €191 million of revenue in last year (2020). AFAS' main software product is called *Profit*, which is an ERP system consisting of different modules such as Taxes, Finance, HRM, Order Management, Payroll, and CRM. This product has over 2 million users across 11.000 organizations of all sizes, ranging from companies with a single employee to companies with thousands of employees.

After 25 years of development, AFAS launched a new version of its ERP system, which is called *SB+*. This new system is based on an internally developed LCDP, called *AFAS Focus*, using an ontological enterprise model [12] (the platform was formerly called NEXT). The system is cloud-based and its architecture applies event-sourcing and CQRS [13] to satisfy quality characteristics such as availability and responsibility. The research and development of AFAS Focus started ten years ago. Approximately 60 companies currently use *SB+* for their day-to-day accounting. Figure 2 shows the instantiated Impact Analysis for Low-Code Development Platforms framework for AFAS Focus.

A. Involved Professionals

The AFAS Focus LCDP is developed and utilized by AFAS only, the three types of professionals described in Section III are all employees of AFAS. The citizen developers are a team of professionals who formerly served in roles such as business analyst, software tester, or support engineer. They have multiple years of experience in the domain of ERP software, but have no formal training in software development. Through internal training and knowledge sharing sessions they are trained in the usage of the LCDP. The LCDP developers consist of four teams that are responsible for the development of the model designer, model transformations, and platform. Finally, a team of operations engineers are responsible for maintaining the runtime and deploying upgrades of AFAS Focus. AFAS Focus uses a four-weekly release schedule: every four weeks a new release is developed, tested, and deployed. During the four weeks that a release is in production, smaller releases (called *hotfixes*) are deployed to solve blocking issues. In the first 18 months (January 2020 to June 2021) that AFAS Focus was used by companies for their day-to-day accounting 212 releases (18 regular releases and 194 hotfixes) were deployed.

B. Subsystems and Artifacts

AFAS Focus is developed using C# on the .NET platform and TypeScript. As mentioned, an ontological enterprise model [12] is used to develop the ERP system on top of this platform. The LCDP developers are responsible for the model designer, called *Studio* and the meta-model, called *OEM*. Required features are designed and developed in collaboration with the citizen developers, who are the 'customers' of these components. The model is created through the combination

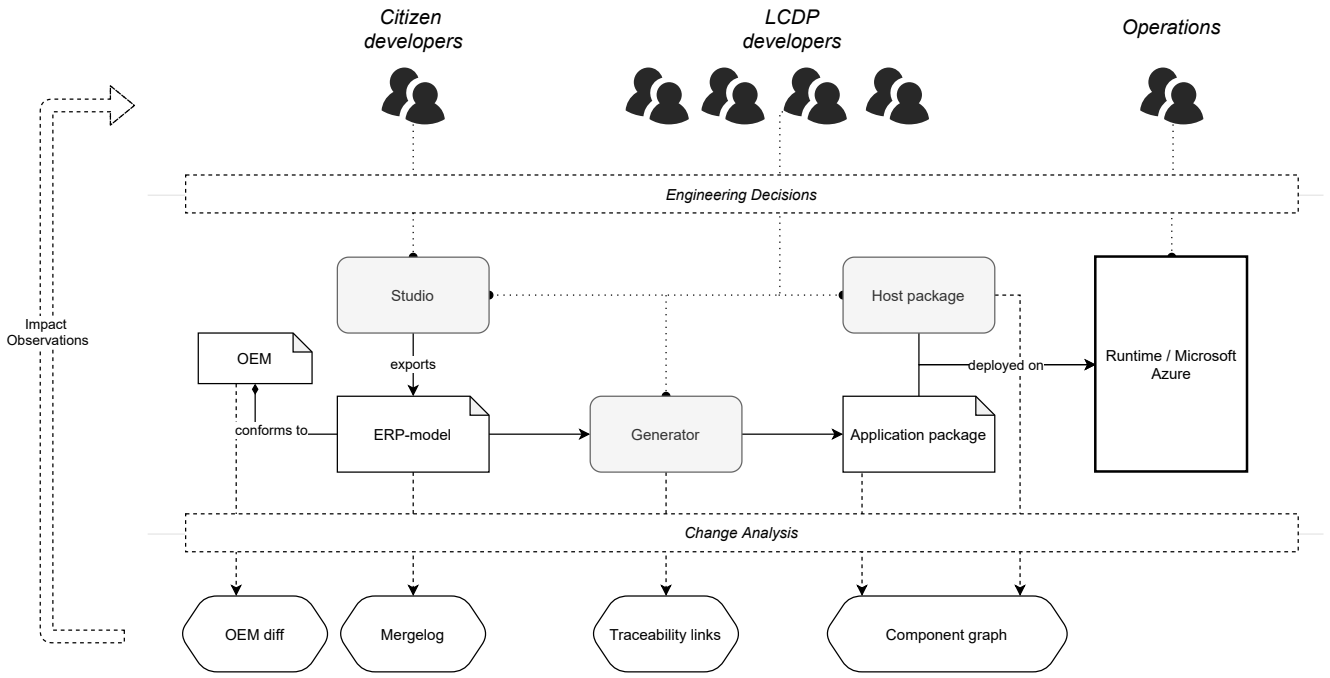

Fig. 2. The *Impact Analysis for Low-Code Development Platforms* framework as instantiated for AFAS Focus.

of a graphical designer and a text-based designer. Through a suite of model transformations (the *Generator*), this model is transformed into a run-time model called the *application package*. To optimize the model execution approach AFAS Focus adopted a custom run-time model that is interpreted [6]. The run-time model expresses the component types present in an event-sourced system, such as *aggregate roots*, *events*, and *projectors*. Together with a *host package*, containing the platform, the application package is deployed on the runtime which is running in *Microsoft Azure*. The host package contains both the interpreters for the run-time model, as well as features that are not dependent on the model and are not developed by the citizen developers. Examples of these features are generic import functionality and user management. This features are developed using 'traditional' software development methods and contained in the host package.

C. Change Analysis Results

From the start impact analysis is applied in the development of AFAS Focus to facilitate the co-evolution between meta-model and model, and model and runtime. We discuss the different applied impact analysis and the context within AFAS Focus.

1) Meta-model Diff: While currently only one single model is developed (the model representing the new ERP system), many more models exist for testing and exploration purposes. To maintain compatibility between *Studio* and the model, the *OEM* is versioned and every model contains the version to which it conforms. When a new version of the *OEM* is introduced, a manual evolution step is developed to facilitate the upgrade of existing models, based on the *OEM diff*. *Studio* itself is evolved manually. Whenever a model is loaded that conforms to an older *OEM* version, the developed evolution steps are executed to automatically upgrade the model. These evolution steps are developed in such a way that only the minimal changes are made to a model to make it conform to the new *OEM*. These evolution steps are developed and tested by the LCDP developers whenever they make a meta-model change. Downgrading a model to an older *OEM* is not supported and facilitated. The citizen developers are briefed and educated on the new *OEM* elements, but are not bothered by details of the evolution steps.

2) Model Diff: Co-evolution of the system and the customer data is one of the biggest challenges faced in the development of AFAS Focus. For an accounting system it is crucial that customer data remains accessible and available after an upgrade. Changes that originate from the model were a big unknown in that challenge, because it was outside of the direct influence of the LCDP developers.

The co-evolution of the model and customer data is solved through a combination of *manual specification* and *operator-based co-evolution* [14], called the *Mergelog*. The most frequent evolution steps in the model are mapped and formally supported in *Studio*. citizen developers can select one of the operators to perform co-evolution with the customer data. The manual specification option serves as fall-back for non-supported operations. Together these make sure that the model and the customer data co-evolves.

The model itself is versioned in a general purpose versioning system (*git*) using a text-based representation. To not ob-

scure the versioning history, the evolution steps automatically executed in *Studio* caused by *OEM*-evolution are confined to a minimum. The LCDP developers review the automated evolution steps in *Studio* to make sure that the model history is not polluted by *Studio*.

3) Transformation Diff: AFAS Focus uses a general purpose programming language (C#) for the model transformations. Initially the transformations were designed in a single multi-phase transformation system. This monolithic transformation system, together with the fact that a general purpose programming language is used made it hard to perform change impact analysis.

Therefore, the single transformation system was redesigned into a component-based transformation system. This not only improves the development process for multiple development teams, it also makes it easier to analyse the transformation system. Currently AFAS implements basic traceability [15] in the transformation system that links elements in the run-time model to the specific transformation component. These traceability links are used in the engineering process by the LCDP developers to find the specific transformation components and model elements that cause a run-time model element.

4) Run-time Model Diff: The run-time model of AFAS Focus consists of a small number of component types that exist in an event sourced architecture [13]. To analyse and observe the impact, the run-time model is represented in a message dependency graph [16]. This component graph contains the different micro-services and their event-based communication. A graph of the current release of AFAS Focus consists of around 25.000 nodes and 35.000 edges. These nodes represent the components in an event-sourced system: 5.000 nodes are components containing logic, 15.000 nodes represent messages, and the remaining nodes are data objects. The edges represent the usage patterns between these components. The graph can be explored in a visual representation. The size of the AFAS Focus component graph is useless to visualize in one image, however, by enabling developers to explore the graph many useful observations can be made.

To analyse the impact of changes a diff between two of these graphs is created. This diff reflects the changes made to different component types. An example summary of such a diff is shown in Table I. The numbers in Table I show a 0.1% total size increase. It shows per component type the number of added, removed, and changed elements, together with totals off the nodes and edges. Note that the edges can only be added or removed, not changed, because they do not contain further information. This summary information acts as a starting point to browse the diff information and drill-down to the lowest possible level: a diff of a specific model element (as shown in Figure 3). These diffs can be used to spot specific changes that need the attention of the development team. Example changes that require manual verification are changes that require custom data evolution steps (such as a property that has become mandatory, or a property type that has changes), or changes that could result in data loss (such as an event type that has been removed by error).

TABLE I
EXAMPLE DIFF SUMMARY BETWEEN VERSION 1.19 AND 1.20 FROM THE COMBINED RUN-TIME MODEL AND PLATFORM IMPACT ANALYSIS. THE CHANGES PER COMPONENT TYPE AS WELL AS AGGREGATED TOTALS ARE SHOWN.

Component Type	Original	Added	Removed	Changed	New
Command	3679	34	7	51	3706
Event	11005	76	84	570	10997
QueryModelObject	4797	79	91	409	4785
StreamItem	1395	12	6	273	1401
StreamItemRouter	1379	16	103	142	1292
QueryProjector	578	11	6	49	583
StreamProjector	565	0	4	230	561
Nodes	23398	228	301	1724	23325
Edges	35898	474	506		35866

Using the component graph the LCDP developers have found bugs, such as

- Components receiving messages for which no component exists that send those messages.
- Messages that are received and send, but for which no contract description exist.

An example of a less obvious observation is the identification of a part of the system that has a high level of complexity (measured in terms of many different messages and many different components involved), but offers little functionality in return. Parts with those characteristics are discussed in architecture meetings and optimizations are planned accordingly.

5) Platform Diff: The platform part of AFAS Focus contains the interpreters for the run-time model elements as well as features that are not modelled by the citizen developers and do not depend on the model. Similar to the transformation subsystem the platform too is implemented in C#. However, unlike the transformation system the platform does not have the same challenges for doing change impact analysis. Features that are developed in the platform subsystem use the same component and message types. Therefore, the platform system can be represented through reverse engineering in the same component graph as the run-time model.

6) Run-time Diff: The run-time model and platform diff are combined into a single run-time diff. This diff represents the whole impact of the new release on the runtime.

The run-time diff is used by the LCDP developers to analyse the impact of a new release and plan the upgrade procedure accordingly. The data upgrade steps are verified using the diff, making sure that all necessary upgrades are specified. Depending on the impact of the release a separate upgrade strategy is used.

- **move** - When a release only contains business logic or user interface changes a straightforward move upgrade can be performed. During this upgrade a new application process is launched that connects with the same data storages. When this new process is verified to run correctly all incoming connections are transferred to the new process. This is the fastest upgrade process.

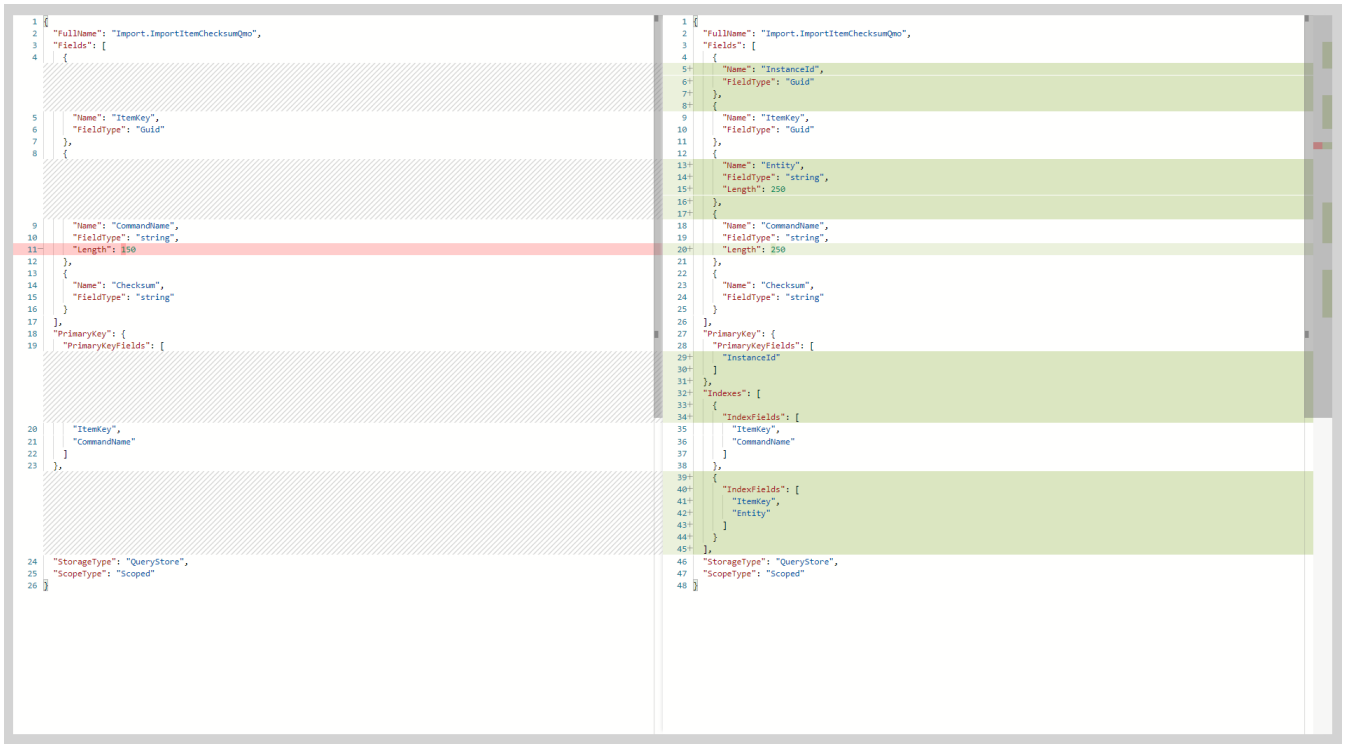

Fig. 3. The diff of a single data storage element from the generated component graph diff. These elements are represented in JSON, the diff shows a plain text-based diff. The diff shows two added properties, a changed property length, and two new indexes. Especially the changed property length is important: a decrease in length requires a data evolution step to make sure existing data conforms to the new schema. This feedback could be presented in *Studio* to warn the citizen developer.

- **minor** - When a release only contains business logic, user interface, or volatile storage changes, a minor upgrade can be performed. During this upgrade a new application process is launched, the data storages are copied, and the data schema changes are executed on the copy. When the new application process is verified to run correctly all incoming connections are transferred to the new process.
- **major** - Whenever the event messages have changed a major upgrade is required. During this upgrade the changed events are rewritten and saved into a copy of the data storage. When the new application process is verified to run correctly all incoming connections are transferred to the new process.

Depending on the complexity of the changes when a **major** upgrade is required, the operation engineers can decide to allocate more resources for the system during the upgrade. Specific upgrade challenges are identified and reported to the operations team.

One metric that can be used to measure the size of the ERP system *SB+* is the total number of components and relations between the components. Table II shows an overview of numbers of the last nine versions. These numbers give a sense of the magnitude of the ERP system, however, similar to source code lines the number of components does not relate to productivity. The decrease of 8% between version 1.15 and 1.16 for instance can be attributed to an optimization in the transformation system. These numbers can be used by the LCDP developers and operation engineers to plan the available resources on the runtime platform: more data storage objects require more data resources, while a larger number of components with logic require more memory resources.

The LCDP developers also analyse the run-time diff to identify required optimizations in platform and model transformations. Together with the citizen developers the diff is analyzed to identify optimizations in the model by applying different meta-model elements, or by introducing new meta-model elements.

7) Taxonomy criteria: Following the eight criteria of the taxonomy for software change impact analysis the approach can be described as follows:

- *The scope*: analysis is done on code, and models
- *The granularity*: this differs for the different analyses, the *OEM diff* and the required evolution operations are done on all levels, the *mergelog* is recorded on the level of attributes, the *traceability links* are recorded on the level of generator components, and the *component graph* is created on the level of the architectural components.
- *Utilized techniques*: analysis is done by traceability links, message dependency graphs, and model diffs.
- *Style of analysis*: the impact analysis in AFAS Focus is done globally.
- *Tool support*: no generic or open source tools are used.
- *Supported languages*: the implemented approaches are specific to the meta-model and architecture of AFAS

TABLE II
THE LAST NINE RELEASES OF AFAS SB+ WITH TOTAL NUMBERS OF NODES AND EDGES OF THE COMPONENT GRAPH, ENRICHED WITH THE PERCENTAGE OF CHANGE WITH RESPECT TO THE PREVIOUS VERSION. DUE TO PROBLEMS WITH THE RELEASE OF VERSION 1.17, VERSION 1.18 WAS NEVER RELEASED, THEREFORE IT IS ABSENT IN THIS TABLE.

Version	Nodes		Edges	
	Total	% change	Total	% change
1.11	18104		26202	
1.12	18499	+2%	26935	+2%
1.13	19431	+5%	28205	+4%
1.14	20638	+6%	29955	+6%
1.15	21814	+5%	32034	+6%
1.16	20132	-8%	29711	-8%
1.17	23819	+18%	36428	+22%
1.19	23398	-2%	35898	-2%
1.20	23325	-1%	35866	-1%

Focus.

- *Scalability*: scalability is no real concern, because there is a single model and the analysis is executed on demand.
- *Experimental results*: these are discussed in this research.

V. ANALYSIS

The previous section described how impact analysis is embedded in the software development process of AFAS Focus. However, we also observe possible improvements.

The **meta-model diff** is used to manually create model evolution steps. This creation could be automated by analysing the meta-model diff. The meta-model diff could also be used to automate the evolution of the model designers. These two improvements would make the process of meta-model evolution more efficient.

The chosen solution for co-evolution of model and customer data, based on the **model diff**, serves its intended purpose, but also has a number of drawbacks. First of all, the chosen approach remains laborious and complicated. Citizen developers are required to explicitly specify evolution operators. However, they are only aware and capable to specify their own evolution. Changes of several developers are combined into a single release, but how the different evolution operators influence and even conflict with each other is not obvious and cannot be specified. Second, the operator-based co-evolution is only able to express model evolution. Changes in the meta-model, the model transformations, or in the platform can not be expressed through these operators. Each of those cases needs to be expressed through manual specification, making the process error prone and laborious. Third, the co-evolution does not support the teams in improving the solutions by providing feedback. It does not facilitate evaluation of the chosen model changes, which might not be optimal.

While the **transformation system** already generates traceability between the transformation components and the run-time model, it misses a link between model element and run-time model element. Such a link would facilitate the translation of run-time errors into model errors for the citizen developers.

The run-time diff, the combination of **platform diff** and **run-time model diff**, greatly improves the impact analysis of AFAS Focus. However, this artifact is also not yet utilized to its full potential. It could replace the model diff and serve as a basis for the data evolution steps. As the run-time diff represents the full evolution step it would allow for a more complete and more automated generation of the required data evolution steps. From the 209 releases that AFAS Focus had, 37 required a *major* upgrade. Most of these major upgrades required a data evolution step that could not be automatically generated with the current operation based co-evolution. Example manual specifications are property type transformation, specific calculations for introduced mandatory properties, and renames of event types. Another improvement would be a better suited representation of the generated component graph for the citizen developers to analyse. The traceability between run-time model and model elements could support such a representation. Applying change patterns [17] could be another solution to improve the representation by summarizing smaller changes in higher-level change patterns. An important missing feature is safety guards against public APIs. Earlier research [8] showed the importance of API management for LCDPs. Safety checks on these published APIs supports the citizen developer in these tasks. Finally, the diff should be used to generate the required data evolution in a semi-automated way. Certain semantics of the model evolution will be lost through the indirection of the impact analysis, because it is done on the run-time model. This could either be mitigated by also analysing the model impact, or by manual specification.

VI. DISCUSSION

LCDPs enable citizen developers to develop increasingly complex software without formal software development training. While this improves productivity and efficiency it also introduces new challenges in the development process. The evolution of the LCDP and the applications built on top of it is one of those challenges. Impact analysis can play an important role in the mitigation of this challenge.

As we have observed and experienced in the development of AFAS Focus, impact analysis supports the professionals in the planning and orchestration of software evolution. The presented Impact Analysis for Low-Code Development Platforms framework offers a conceptual structure to reason about impact analysis for LCDPs. At AFAS Software the framework proved its use in the design of the different subsystems and artifacts, and the implementation of impact analysis. The development process benefits from the different analysis results, even with the identified improvements.

Current research in model-driven development and low-code development platforms offer a lot of the lower-level techniques and approaches to perform impact analysis. However, an overall framework to structure the plan-do-act process of engineering teams is missing. Our research is a proposal for such a framework but requires more grounding and evaluation.

While we believe that it can be generalized to other contexts, this should be proven by further research.

For future work we plan to execute a systematic literature review to ground the framework in existing research on model-driven development, low-code development platforms, and change impact analysis. The review should result in a comprehensive overview and concept definitions that would bring together the different research areas.

Second, the framework itself will be validated with other LCDP providers. Case studies at other LCDP providers are necessary to evaluate the framework and correct it from any biases. To prevent the framework from following biases of a single provider, multiple providers should share and combine their knowledge. We plan to conduct multiple case studies in the near future.

The results of the literature review and the multiple case studies will be used to add more detail to the framework: specific guidance and useful techniques that can be applied. After these improvements the framework can be evaluated on completeness and usefulness through expert interviews.

VII. Related Work

Co-evolution in model-driven development platforms is a well researched topic. An overview of the different approaches is given in [14], while [18] describes an discusses coupled transformations. An approach for creating model diffs is presented by [19]. In [20], [21], and [22] approaches for (semi-)automated co-evolution of meta-model and models are described. [20] categorizes the meta-model changes in *non-breaking changes*, *breaking and resolvable changes*, and *breaking and unresolvable changes*. By using high-order transformation rules, the second category can be used to automatically adapt models to new meta-model versions. The research of [23] is similar and focuses also on the automated adaption of models to meta-models. A dynamically adapting system is proposed by [24]. Impact analysis to support the incremental execution of model transformations is another application [25]. A megamodeling approach is presented in [26], which supports the capturing of change impact in a model. By doing this, change impact becomes a model itself, which allows the application of model-driven tools to the challenge of change impact.

The problem of representation of change impact is researched in the area of *change patterns*. Change patterns express changes to a process on a higher level of abstraction, making them easier to comprehend. This notion is introduced by [27], [28] for Process-Aware Information Systems (PAIS). The authors apply these change patterns to asses the level of change support in different PAIS. The patterns form a language that allow an easy and comprehensible comparison of the different systems.

Distributed event-based systems pose another challenge in impact analysis. Components in event-based systems are intrinsic loose coupled which makes them hard to evolve and analyse. [29] use the notion of change patterns to analyse event-based systems. Their research shows that change patterns are an efficient language to capture the evolution of an event-based system. [16] proposes a static analysis that analyses distributed event-based systems. Analysis of traditional software systems depends on explicit invocation to create a dependency graph. Their proposed method analyses the message-oriented middleware that these systems are based on and creates a dependency graph from those results.

VIII. Conclusion

Low-code development platforms (LCDPs) accelerate the development of software through new abstractions that remove as much of the technical details as possible. However, challenges such as software evolution remain. Citizen developers, LCDP developers, and operations engineers need tools and processes to solve these challenges. Together these professionals, regardless if they are from the same company or not, are responsible for the success of the application of the LCDP. Evolution plays an important role in that success and this requires that these professionals collect feedback that informs and supports there engineering decisions. We believe that impact analysis helps and supports these teams in reaching their goals.

An overall framework to structure the implementation of impact analysis for LCDPs is missing. In Section III we propose the *Impact Analysis for Low-Code Development Platforms* framework that conceptualizes the process of impact analysis for LCDPs. It describes the different subsystems and artifacts, together with the process of impact analysis. Using the taxonomy of Lehnert [11] we discussed the variability in the framework and how providers can implement this.

Through a case study of an industry LCDP we explore the framework in more depth. Although case studies at other LCDP providers are necessary to evaluate the framework and correct it from any biases, we believe that impact analysis within LCDPs can improve both the applications developed on top of the LCDP as well as the platform itself. Impact analysis should be at the foundations of the LCDP development process.

REFERENCES

[1] Jordi Cabot. Positioning of the low-code movement within the field of model-driven engineering. *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, pages 535–537, 2020.

[2] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, pages 171–178, 2020.

[3] Christoforos Zolotas, Kyriakos C. Chatzidimitriou, and Andreas L. Symeonidis. RESTsec: a low-code platform for generating secure by design enterprise services. *Enterprise Information Systems*, 12(8-9):1007–1033, 10 2018.

[4] Silviu George Pantelimon, Tudor Rogojanu, Andreea Braileanu, Valeriu Daniel Stanciu, and Ciprian Dobre. Towards a seamless integration of iot devices with iot platforms using a low-code approach. *IEEE 5th World Forum on Internet of Things, WF-IoT 2019 - Conference Proceedings*, pages 566–571, 2019.

[5] Raquel Sanchis, Oscar García-Perales, Francisco Fraile, and Raul Poler. Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences (Switzerland)*, 10(1), 2020.

[6] Michiel Overeem, Slinger Jansen, and Sven Fortuin. Generative versus interpretive model-driven development: Moving past 'It depends'. In L. Pires, S. Hammoudi, and B. Selic, editors, *Model-Driven Engineering and Software Development. MODELSWARD 2017. Comm. in Comp. and Inf. Science*, volume 880, pages 222–246. Springer International Publishing, Cham, 2018.

[7] Michiel Overeem, Marten Spoor, and Slinger Jansen. The Dark Side of Event Sourcing: Managing Data Conversion. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 193–204, 2017.

[8] Michiel Overeem, Slinger Jansen, and Max Mathijssen. API Management Maturity of Low-Code Development Platforms. In *Enterprise, Business-Process and Information Systems Modeling*, pages 380–394, Melbourne, 2021. Springer International Publishing.

[9] Lissette Almonte, Esther Guerra, Iván Cantador, and Juan De Lara. Recommender Systems in Model-Driven Engineering A Systematic Mapping Review. *Software and Systems Modeling*, 2021.

[10] Stefan Kögel. Recommender system for model driven software development. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, volume Part F130154, pages 1026–1029. Association for Computing Machinery, 8 2017.

[11] Steffen Lehnert. A taxonomy for software change impact analysis. In *IWPSE-EVOL'11 - Proceedings of the 12th International Workshop on Principles on Software Evolution*, pages 41–50, 2011.

[12] Henk Van Der Schuur, Erik Van De Ven, Rolf De Jong, Dennis Schunselaar, Hajo A. Reijers, Michiel Overeem, Machiel De Graaf, Slinger Jansen, and Sjaak Brinkkemper. NEXT: Generating tailored ERP applications from ontological enterprise models. In *IFIP Working Conference on The Practice of Enterprise Modeling*, volume 305, pages 283–298. Springer, 2017.

[13] Michiel Overeem, Marten Spoor, Slinger Jansen, and Sjaak Brinkkemper. An Empirical Characterization of Event Sourced Systems and Their Schema Evolution - Lessons from Industry. *Journal of Systems and Software*, 178(110970), 2021.

[14] Louis Rose, Richard Paige, Dimitrios Kolovos, and Fiona Polack. An analysis of approaches to model migration. In *Models and Evolution (MoDSE-MCCM) Workshop*, pages 6–15, 2009.

[15] Ismênia Galvão and Arda Goknil. Survey of Traceability Approaches in Model-Driven Engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference*, pages 313–313, 2007.

[16] Daniel Popescu, Joshua Garcia, Kevin Bierhoff, and Nenad Medvidovic. Impact analysis for distributed event-based systems. *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS'12*, pages 241–251, 2012.

[17] Barbara Weber, Stefanie Rinderle-Ma, and Manfred Reichert. Change Support in Process-Aware Information Systems-A Pattern-Based Analysis. *Data Knowledge Eng*, 66(3):438–466, 2007.

[18] Ralf Lämmel. Coupled Software Transformations - Revisited. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering - SLE 2016*, pages 239–252, 2016.

[19] Antoine Toulmé. Presentation of EMF Compare Utility. In *Eclipse Modeling Symposium*, 2006.

[20] Boris Gruschko, Dimitrios S Kolovos, and Richard F Paige. Towards Synchronizing Models with Evolving Metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution*, pages 3–3, 2007.

[21] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, Alfonso Pierantonio, Dipartimento Informatica, and I L Aquila. Automating Co-evolution in Model-Driven Engineering. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE (pp. 222-231)*, 2008.

[22] Davide Di Ruscio, Ralf Lämmel, and Alfonso Pierantonio. Automated co-evolution of GMF editor models. In *International Conference on Software Language Engineering*, pages 143–162, 6 2010.

[23] Guido Wachsmuth. Metamodel adaptation and model co-adaptation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4609 LNCS:600–624, 2007.

[24] Hugo Sereno Ferreira, Filipe Figueiredo Correia, and Leon Welicki. Patterns for data and metadata evolution in adaptive object-models. *Proceedings of the 15th Conference on Pattern Languages of Programs PLoP 08*, page 1, 2008.

[25] David Hearnden, Michael Lawley, and Kerry Raymond. Incremental Model Transformation for the Evolution of Model-Driven Systems. In *9th International Conference, MoDELS 2006*, 2006.

[26] Ludovico Iovino, Alfonso Pierantonio, and Ivano Malavolta. On the impact significance of metamodel evolution in MDE. *Journal of Object Technology*, 11(3), 2012.

[27] Barbara Weber, Stefanie Rinderle, and Manfred Reichert. Change patterns and change support features in process-aware information systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4495 LNCS:574–588, 2007.

[28] Stefanie Rinderle-Ma, Manfred Reichert, and Barbara Weber. On the formal semantics of change patterns in process-aware information systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5231 LNCS:279–293, 2008.

[29] Simon Tragatschnig, Srdjan Stevanetic, and Uwe Zdun. Supporting the evolution of event-driven service-oriented architectures using change patterns. *Information and Software Technology*, 100:133–146, 2018.