VISDOM

Visual software diagnostics

D1.1.2 VISDOM input to ITEA Living Roadmap

Programme	ITEA3		
Challenge	Smart Engineering		
Project number	17038		
Project name	Visual diagnosis for DevOps software development		
Project duration	1st October 2018 – 30st June 2022		
Project website			
Project WP	WP*1* - Pre-studies and requirements		
Project Task	Task *1.1 – Update of state-of-art and state-of-the practice analysis for visualization in software projects in DevOps context*		
Deliverable type	X	Doc	Textual deliverable
		SW	Software deliverable
Version	V19		
Delivered	10/02/2020		
Access	x	Public	
		Abstracts are public	
		Confidential	

Document Contributors		
Partber	**Author**	**Role**
EXPERIS	Ester Sancho	editor
EXPERIS	Miriam Moreno	writer
GRO	Paris Avgeriou	writer
INVENCO	Mika Koivuluoma	writer
OCE	Lou Somers	writer/reviewer
OULU	Markus Kelanti	writer
TAU	Kari Systä	writer/reviewer
TAU	Outi Sievi-Korte	writer/reviewer
TIOBE	Paul Jansen	writer
TIOBE	Marvin Wener	writer
UPC	Lidia López	writer
UPC	Xavier Franch	writer
VINCIT	Veli-Pekka Eloranta	writer
BEIA	George Suciu	writer/reviewer

Document History			
Date	**Version**	**Editors**	**Status**
18/06/2019	ToC	EXPERIS	Table of Content
10/07/2019	V01	TAU	Draft
30/09/2019	V02	OCE/GRO	Draft
17/10/2019	V06	TIOBE	Draft
22/10/2019	V08	OULU	Draft
12/11/2019	V09	VINCIT	1ST Final Draft
20/11/2019	V10	OCE	Reviewed version
26/11/2019	V12	UPC/EXPERIS/TAU	2nd Final Draft
10/12/2019	V16	EXPERIS	Final Version
16/12/2019	V16.01	TAU	Peer Review
18/12/2019	V17	EXPERIS	Submission
07/02/2020	V18	BEIA	Reviewed version
10/02/2020	V19	EXPERIS	Re-submission to ITEA STG

Table of Contents

Executive Summary 6
1. Introduction to VISDOM 8
2. State-of-Art of Visualizations IN Software PROJECTS 10
2.1. State-of-the-art of Visualizations in Quality Aspects 10
2.2. State-of-the-art of Visualizations in International SaaS-business 13
2.3. State-of-the-art of Visualizations in Teaching 13
3. State-of-the-art on the DevOps & Supporting Tools 19
3.1. DevOps Supporting Tools 21
4. Competence analysis on software analytics and visualization 25
4.1. Software Analytics based on Commercial Approaches 25
4.2. Software Analytics Community-oriented Approaches 41
5. Visdom State of Practice 46
5.1. State of Practice at OCE 46
5.2. State of Practice at EXPERIS / UPC 47
5.3. State of Practice at QENTINEL 51
5.4. State of Practice at TAU 52
5.5. State of Practice at INVENCO 54
5.6. State of Practice at VINCIT 56
6. Conclusions 58
References 59
Appendix A: 64

Table of Figures & Tables

Figure 1: Example of very simple heat map implemented with vue-heatmap.js (Vue-Heatmap, 2019) 13
Figure 2: RESCON Gantt charts and resource profiles (Deblaere et al., 2011) 14
Figure 3: PpcProject Gantt chart and resource visualization (Morera et al., 2013) 15
Figure 4: Sprint burndown chart (Igaki et al., 2014) 15
Figure 5: A second-level diamod (Espinosa-Curiel et al., 2010) 16
Figure 6: A third-level diamod (Espinosa-Curiel et al., 2010) 17
Figure 7: Radar chart provided by ScrumLint (Matthies et al., 2016) 18
Figure 8: Continuous deployment pipeline 19
Figure 9: Example of tools used in DataOps Value Pipeline 20
Figure 10: CD logical pipeline example, source dzone.com 21
Figure 11. Bitergia Dashboard 27
Figure 12. Datadog Dashboard 28
Figure 13. Kiuwan Dashboards 29
Figure 14. Kiuwan simulation feature 29
Figure 15. New Relic Dashboard 30
Figure 16: SonarQube Dashboard 31
Figure 17. SonarCloud Dashboard 32
Figure 18. SonarCloud charts 32
Figure 19. SonarCloud code improvement suggestion 33
Figure 20. Splunk Dashboard 34
Figure 21. Splunk business processes 34
Figure 22. Splunk prediction 35
Figure 23. Tasktop Dashboard 35
Figure 24. Tasktop integrations visualization 36
Figure 25: CAST SW Dashboard 37
Figure 26: NDepend Dashboard 38
Figure 27: Teamscale Dashboard 39
Figure 28: TiCS Framework Dashboard 40
Figure 29: Grafana Dashboard 41
Figure 30. Codefeedr solution 42
Figure 31. MEASURE Platform Dashboard 43
Figure 32:Analysed tools by DevOps pipeline categorization 45
Figure 33 Dashboad focused on print usage 46
Figure 34: Example of an error occurrence graph 47
Figure 35: Example of an error occurrence table 47
Figure 36. QaSD visualizations (graphical & textual) 48
Figure 37. QaSD navigation schema 48
Figure 38. Q-Rapids Raw Data Dashboard 49
Figure 39. QaSD prediction view 49
Figure 40. QaSD simulation view 50
Figure 41. QaSDquality alert and QR suggestion 50
Figure 42. Q-Rapids QR simulation 51
Figure 43: Qentinel Quality Intelligence for DevOps Dashboard 52
Figure 44: Visualizations for teaching produced by TAU. 53

Figure 45: Visualizations produced by Agilefant 54
Figure 46: Example from the monitoring tool UI, showing e.g. some crucial SQL Server related information. 55
Figure 47: Example Power BI report from the certain BI environment status with drill-down capabilities 55
Figure 48: Example of a LaaS MixPanel (a) 56
Figure 49 : Example of a LaaS MixPanel (b) 56
Figure 50:Visualization of LaaS companion app data 57

Table 1: OSLC Domain Specifications 23
Table 2. Commercial Approaches Comparison 25
Table 3. Community Approaches Details 41

EXECUTIVE SUMMARY

VISDOM project aims at developing new types of visualizations that utilise and merge data from several data sources in modern DevOps development. The final goal is to provide simple "health check" visualizations on the status of the development process, software and usage.

Nowadays the visualization of data and information generated by a company has become an important part of the decision process that covers all the departments of a company. Visualization is a powerful method for internal communication within a team of developers, but even more, it is useful in interdisciplinary communication processes including stakeholders with different roles, skills and experience. Making the communication process effective needs the most advanced visualization tools. Visualization is particularly useful for communicating information related to software projects-related information at near real-time to stakeholders directing development-efforts, such as customers and managers.

Current IT tools used for software development collect data from several phases and tasks in the development process, and are thus a useful data source for various visualizations. Examples of such tools include project management, error tracking, version management, IDE (integrated development environment) and automated testing in different environments. We also need to particularly consider DevOps, as Continuous Deployment requires tools that monitor applications and systems administration tools, bringing with it additional data sources. However, while there are several software development tools that offer visualizations of some data source, there are none that would offer an integrated visual analysis that combines all the data sources involved in a development process.

The general aim of the project is both develop new types of visualizations and to develop an integrated platform that supports these visualizations of DevOps environments, enabling merging data from several data sources. The goal is to develop easy-to-use “health check” visualizations, leaning on analogies from medicine, that will enable quickly grasping the status of a software development project.

The proposed visualizations and platform have several goals and potential uses. Firstly, they are aimed to improve communication both within a software development team and between the team and outside stakeholders. Secondly, they can be used to improve DevOps practices in various contexts. Finally, visualizations can be used to train a new generation of software experts by effectively teaching modern software development processes and practices with the help of visualizations.

In summary, the ultimate objective of this document is to review the latest advances in analysis and visualization in the software industry and its development processes. It also reviews the advances achieved in DevOps environments and different support tools available.

In the introduction section, we offer an overview of all the layers covered by VISDOM, from data mining and modelling, to analysation and visualisation of information of interest. This is followed by an overview of the state of the art, which not only covers the quality of the software, but also covers current approaches to the use of visualisations in teaching and we pay special attention to the visualisation of SaaS (Software-as-a-Service), as this is the most common approach to software delivery in DevOps.

Subsequently we conduct a study of solutions that can currently be found in the market offering analysis and visualization of software quality and development, in order to assess our potential competition and differentiate ourselves.

Finally, we explore the existing visualization practices in the companies that participate in VISDOM.

1. INTRODUCTION TO VISDOM

The VISDOM project presents the opportunity to address the challenges of integrating data from different sources and visualizing metrics. More importantly, it will ensure feedback from different stakeholders in the software project to also include those outside of the software development and operation processes, thus differentiating it from existing displays used for dashboards, which typically target a single group of stakeholders.

The project involves the following processes:

1. **collecting data** from various DevOps development tools such as project management, time tracking, messaging and communication, issue tracking, version management, IDE, automated testing, and system monitoring.
2. **analysing the data** and forming new metrics to support DevOps development
3. **visualising the metrics** in the form of a customisable dashboard with the possibility of lowering the level to analyse in detail which are the possible problems and their causes.

VISDOM visualizations will use data from several sources, the system needs to combine data from several sources. On a conceptual level this is a known problem (Wiederhold, 1992; Wiederhold, 2017) that requires a lot of work in each practical case (Otto Hylli, 2015). Also, the implementation is not that straightforward as it involves dealing with multiple protocols with varying authentication mechanisms. Even though dealing with heterogeneous data sources is not the main focus of the project, we expect innovative solutions to the management of data sources, where authentication is required, and contributions to software architecture research.

According to what has been said, VISDOM could be structured in the following layers or processes:

The first step is to acquire the necessary data. Datamining is the process in which, through a set of techniques and technologies, large databases can be explored, automatically or semi-automatically, with the aim of finding repetitive patterns, anomalies, trends, correlations or rules that explain the behaviour of data in a given context to predict results. It uses statistical practices and in some cases, search algorithms close to artificial intelligence and neuronal networks.

Secondly it is needed to analyse all this data. The results will define the data mining model and it could be applied for forecasting, risk and probability, recommendations, finding sequences or grouping. To build a data mining model, there are various steps to follow in any type of project:

Define the problem and the objective:

Different businesses have different goals, so defining the problem and the goal is very important. The first step is to perform an availability study to investigate the needs of the business and the available data. It is necessary to be sure that the data support the needs of the goal.

Extract and classify information:

Data can be in different formats, contain inconsistences and incorrect or missing entries. Data can be stored in different places. Therefore it´s necessary to prepare the data such as cleaning, removing or interpolating missing values.

Depending of the nature of the problem, identifying sources of data that are the most accurate and determine which data are the most appropriate for use in analysis to get the results defined in the objective. Currently under WP2 umbrella, all use cases are under deep analysis, in order to identify

specific information that will guide VISDOMs final solutions as well as its implementation. Meanwhile, a first draft of the data classification related to the VISDOM project can be seen in Appendix A.

Analyze the information and find patterns:

The patterns, correlations and relationships identified by data mining techniques are inspected, evaluated and analyzed. The evaluation is done by using parameters of interest to determine which patterns are really valid for our purpose.

Development of knowledge through different techniques:

Define the data to create a mining structure. This structure is linked to the source data, but doesn´t contain any data until it is processed. When the mining structure is processed information is returned that can be used by any mining model based on the structure. This process applies algorithms to input data and uses parameters to adjust each algorithm.

Data Visualization:

The knowledge resulting from the evaluation and interpretation will now have to be presented to the interested parties. The presentation is done regularly through visualization techniques and other knowledge representation mechanisms. Once presented, knowledge can, or will be, used to make business decisions. The correct visualization of the data is essential to validate the models.

2. STATE-OF-ART OF VISUALIZATIONS IN SOFTWARE PROJECTS

2.1. State-of-the-art of Visualizations in Quality Aspects

In this section, we present a summary of recent works in the current state of the art for quality measurement and technical debt, including its visualization. Example figures of visualizations are given in Section 4, where we cover commercial and open source tools in this area in more detail.

Quality measurement

Quality attributes like maintainability and reusability are of paramount importance for the evolution of a software system. A complex and poorly designed software system can be hard and expensive for software developers to maintain – sometimes even impossible. Therefore, measuring and maintaining the quality of software is an essential task in software development.

There have been a number of metrics and metric suites proposed by software engineering researchers to assess the maintainability of a software system over the past decades. Some of the well-known ones have been extensively used in practice by software developers. For example, the number of lines of codes and cyclomatic complexity (McCabe 1976) are used to measure the complexity of methods, while depth of inheritance tree, coupling between objects and lack of cohesion of methods (Chidamber and Kemerer 1994) are used to assess the quality of an object oriented design. Several tools proposed by researchers (Tahir and MacDonell 2012), as well as industrial tools (e.g. SonarQube) are able to measure the proposed metrics for the quality of source code and software quality in general. Metrics to measure the quality of a software architecture are not commonly used. Recent studies have attempted to change that by proposing metrics that measure software architecture quality per se. For example, (Mo et al. 2016) propose a new metric for architectural-level maintenance. The metric depends on a clustering algorithm to identify independent modules.

Technical debt

Achieving high maintainability is rather challenging because of the realities of industrial software development. Due to business pressure and the fast pace of development, software engineers often prefer short-term (but lower quality) solutions, which might impact the quality of software gradually. Such trade-offs between expedience and software quality have been termed ‘technical debt’; researchers in the past few years focused on understanding and addressing the impact of technical debt on the quality of software system.

(Avgeriou et al. 2016) define technical debt as “a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability”.

In the past few years several empirical studies have attempted to understand the technical debt phenomenon. (T Besker, Martini, and Bosch 2017) conducted a survey with software developers to determine time, which is wasted during software development. The results of the study show that 36% of the development time is wasted due to technical debt. Another important finding from the study is the negative impact of complex architectural design, which increases the wasted time during software development. These results align with the study from Ernst et al. (2015), which shows that bad architectural choices are the most common source for technical debt.

Technical debt can be classified according to its sources and granularity (e.g. code debt, requirements debt, design debt, test debt and architectural debt). We elaborate on the three most prominent types of technical debt in the following paragraphs, namely code, design and architectural debt.

Code debt

Code debt is a type of technical debt, which emanates from bad source code and wrong programming practices (e.g. code smells, violations of programming guidelines). Due to the complexity and variety of modern software development environments (e.g. within a DevOps environment), researchers try to understand aspects of code debt in various settings. For example, (Ståhl, Martini, and Mårtensson 2019) explored the relationship between the complexity of source code and source code commits in a continuous integration development environment. The study shows that large commits increase the complexity of source code and consequently technical debt. (Lerina and Nardi 2019) conducted a quantitative study to analyse the impact of software code clones (i.e. duplicated source code) on technical debt. The study verifies that code clones increase technical debt. *Self-admitted technical debt* concerns intentional source code issues, which are introduced by developers as quick or temporary fixes, as described by (Potdar and Shihab 2014). Their study shows that up to 31% of the files in a project contain items of self-admitted technical debt. Moreover, only between 26% and 63% of the items are corrected. This phenomenon of self-admitted technical debt is further studied by other researchers (e.g. (Zampetti, Serebrenik, and Di Penta 2018)).

In addition to empirical studies on code debt, researchers proposed approaches to identify, measure and prioritize code debt. For example, (d. S. Maldonado, Shihab, and Tsantalis 2017) proposed an approach to identify self-admitted code debt from source code using methods from natural language processing. (di Biase et al. 2019) proposed a new maintainability model to identify technical debt of a software system. The model measures code changes and commits by assessing the risk of change for each commit. (Mori et al. 2018) evaluated the usefulness of metrics on several domains to identify technical debt. They determined a method to set suitable thresholds for a project and a domain.

Design debt

Design debt is concerned with technical debt, which occurs due to inappropriate design of software (e.g. wrong application of design patterns). Approaches have been proposed to identify and prioritize design debt items. For example, (Zampetti et al. 2017) proposed a machine learning approach to identify self-admitted design debt in source code. The approach depends on features from source code, as well as structural and readability metrics. (Plösch et al. 2018) proposed an approach to prioritize design debt items by quantifying them using benchmarking techniques. (Eposhi et al. 2019) analysed the impact of design refactoring on the remediation of design debt by assessing the density and diversity of certain symptoms for design quality.

Architectural debt

Architectural debt happens due to sub-optimal architectural design decisions and immature architectural elements, which impact the quality of a software system negatively (Terese Besker, Martini, and Bosch 2018). Independent studies by (Holvitie, Leppänen, and Hyrynsalmi 2014) and (Ernst et al. 2015) show that architectural debt is one of the most common occurring types of technical debt.

Researchers explored types of items, which could incur architectural debt. (Martini and Bosch 2015) identified 5 categories of architectural debt items based on an empirical study. Moreover, they determined their significant impact on the maintenance of the system. In another study, (Soares de Toledo et al. 2019) investigated different architectural technical debt issues, which occur within a

microservice architecture. In addition, they analysed their negative impact (i.e. interest) and possible solutions (i.e. principal).

Several approaches identify architectural debt items by assessing system quality aspects. For example, (Cai et al. 2019) proposed a model and method to identify architectural roots, which are main causes for maintainability issues. The method depends on capturing bug-prone files as a sign of critical architectural problems. Several approaches identify architectural technical debt through capturing architectural smells. For example, Fontana et al. (Fontana et al. 2016) developed an approach to detect architectural smells based on structural dependencies in source code. (Xiao et al. 2016) proposed an approach to identify architectural technical debt through capturing patterns of history changes and commits in a source code repository. (Li, Liang, and Avgeriou 2015) proposed a process to capture architectural technical debt during architectural design. The benefit of the approach is to prevent or track ATD before implementation. (Martini and Bosch 2016) proposed a method to decide on refactoring an architectural technical debt. The method supports identifying factors involved in the growth of technical debt interest. The results of the method provide indicators for stakeholders to decide on technical debt refactoring.

Visualisation of technical debt

The visualisation of technical debt has not been thoroughly studied. There are only a few related works for tools, which display technical debt to users. One way of visualising technical debt is using industrial reverse engineering tools. For example, (von Zitzewitz 2019) experimented with Sonagraph to compare a software architecture with current architecture, which is captured from source code. Sonagraph visualizes components of a software system, their relationships, and displays architectural deviations. (Tornhill 2018) used Codescene to visualise parts of source code, which require refactoring. The tool presents candidate elements as intersected circles. In addition, researchers proposed approaches to visualise technical debt. (Liu et al. 2018) proposed an Eclipse plugin to detect and display sections of source code, which are identified as self-admitted technical debt. (Eliasson et al. 2015) proposed an approach to visualise architectural technical debt and its interest in an automotive domain. The visualisation uses colours to distinguish between components with positive and negative impact on the system quality. DebtFlag (Holvitie and Leppänen 2013) is an Eclipse plugin and web application tool to manage and visualise technical debt in source code. The plugin colours technical debt code instances, while the web application model and colour classes and methods, which have technical debt.

Open areas for research

Measuring the quality of software and particularly its software architecture in a DevOps environment has been rarely investigated. Based on the above state of the art, we find the following shortcomings of current works that require further research:

a. Lack of sufficient metrics and tools to measure the quality of software architecture in a DevOps environment.
b. Lack of tracing architectural decisions to the architectural technical debt they incur.
c. Lack of tools to capture, measure and visualize architectural technical debt in a DevOps development environment.

2.2. State-of-the-art of Visualizations in International SaaS-business

To enter the international SaaS (Software-as-a-Service) business it is crucial to understand the users and how they use the system. Development teams need to identify where the bottlenecks in the system are and find out whether the end users really know how to use the system. In addition, we need to recognize the differences between user groups: Do admins, privileged users and others get their job done with ease. Are there differences in usage patterns in different countries?

Gartner consultancy included software usage analytics under name “Customer Journey Analytics” in its hype cycle graph in 2018 [GARTNER]. Gartner defined the addition as “the detailed tracking and analysis of users” interactions within a software application.

One of the main goals with usage analytics is to better understand which features are adopted and which are ignored by the end users. Using this information helps the team to guide the development efforts where the most value is. In addition, user analytics can help to identify problems in the UI or UX. Also, a typical use case is to help improve the conversion rate from non-paying users to paying ones.

Gartner also predicted that by 2021, 75% of the software providers will rely on insights from embedded software usage analytics to inform product management decisions.

One of the most common approaches to implement simple usage analytics is to use Google Analytics [GOOGLE]. Another approach is to record data to create a heat map on top of the views in the software. From the heatmap, developers and designers can analyze in which areas users click the most or spend time. Typically, you can also get an idea if users scroll up and down. However, the downside of a heat map is that usually you analyze only a single view with it. You need to combine data from multiple sources to analyze the hot paths through the views of the software.

Figure 1: Example of very simple heat map implemented with vue-heatmap.js (Vue-Heatmap, 2019)

2.3. State-of-the-art of Visualizations in Teaching

A systematic literature review on software visualization showed, that the most studied topics of software visualization are software structure, evolution and behavior (Mattila et al., 2016). Similarly, in the context of education, there are a variety of approaches and tools to visualize programming structures, such as Jeliot (Cisar et al. 2011), Jype (Helminen, 2009), Javavis (Oechsle et al., 2002) and AnyviewC (Wu, 2009), that are used in introductory programming courses. A survey on program visualizations in education has been conducted by Sorva et al. (2013).

Mattila et al. discovered only few studies in visualizing software processes. The same trend can be seen in utilizing visualizations in software engineering education concerning software processes or project management.

A very common way to teach project management is to use simulators, which allow students to practice schedule management and resource allocation, and may include gaming elements, such as

the project scheduling game by Vanhoucke et al. (2009). Simulators usually include some visualizations of the project progress or other data given by the students. The most basic ones only use common diagrams of key data, as with AMEISE (Bollin et al., 2011; 2015). The most common visualizations used as part of project management simulators are Gantt charts showing the project activities Vanhoucke et al. (2009). Other works have used Gantt charts alongside other visualizations. Deblaere et al. (2011) have create RESCON – an educational tool for illustrating scheduling and project management concepts, where they have included project duration curves and resource profiles in addition to Gantt charts (Figure 2). Salas-Morera et al. (2013), who have created PpcProject to teach project management, also include visualizations on resource allocations in addition to Gantt charts (Figure 3). Scheduling and resource allocation are also key elements in a simulator developed by Collofello (2000), where schedule pressure is visualized with a speedometer.

While these approaches show how Gantt charts can be utilized in teaching software processes and particularly scheduling in relation to project management, none of them use live or real data. Gantt charts are used to visualize data in simulators, where the data was either given by students or extracted from a predefined model given by an instructor.

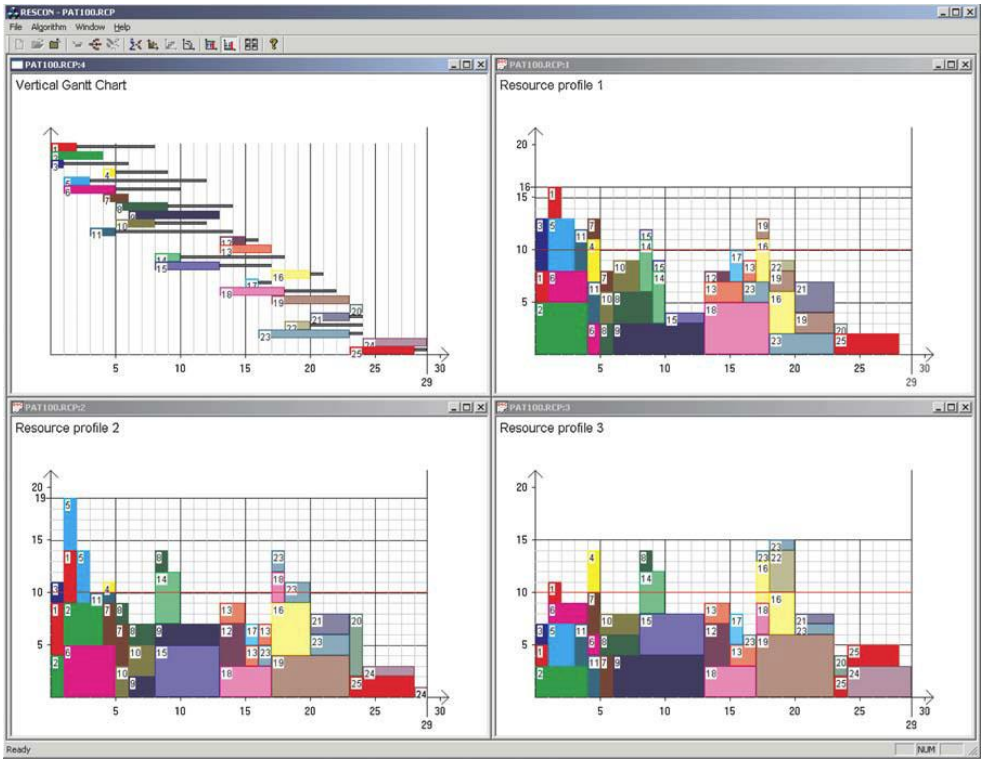

Figure 2: RESCON Gantt charts and resource profiles (Deblaere et al., 2011)

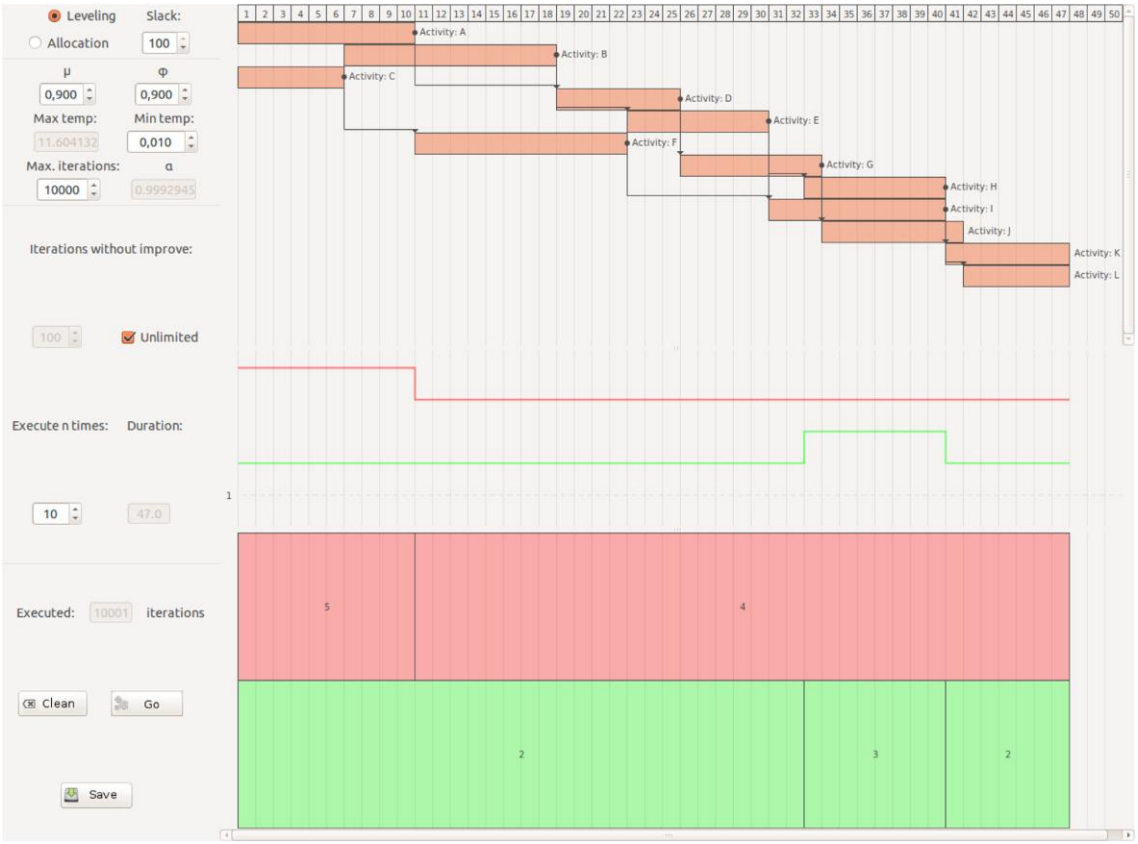

Figure 3: PpcProject Gantt chart and resource visualization (Morera et al., 2013)

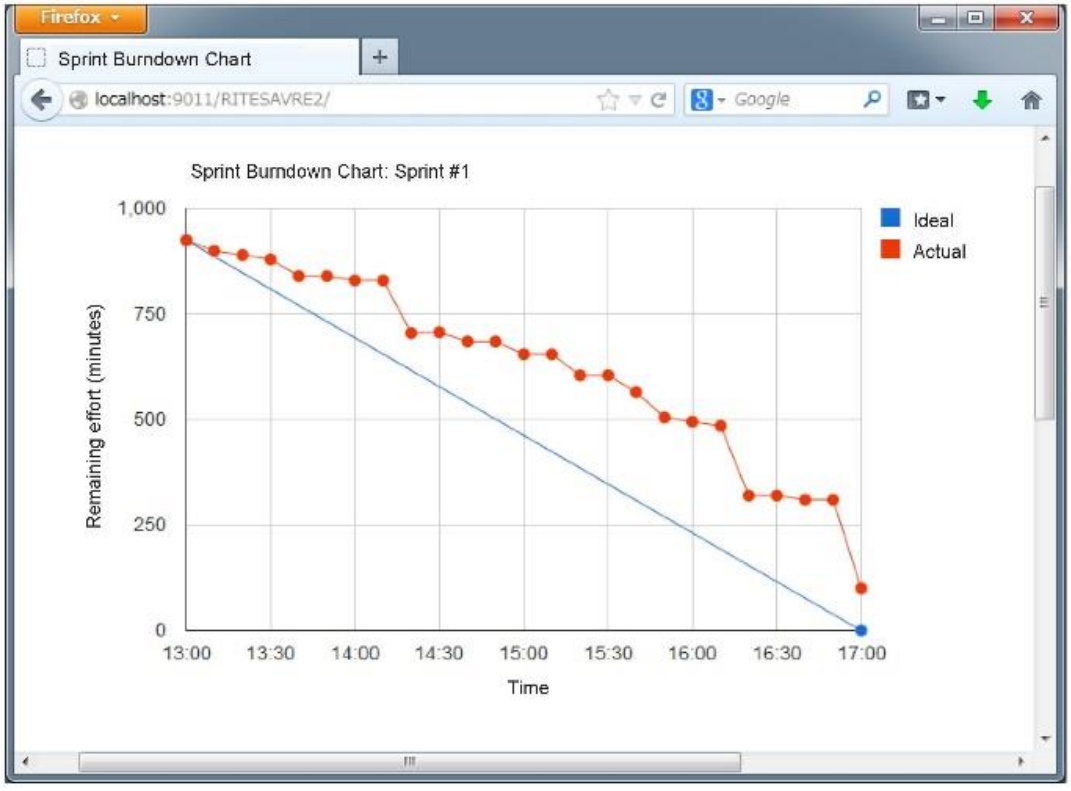

Figure 4: Sprint burndown chart (Igaki et al., 2014)

A common way to visualize progress is to use burndown charts. Igaki et al. (2014) have created a ticket-driven method to teach Scrum processes, and utilize sprint burndown charts that are specialized for their ticket-centered approach to visualize project status (see Figure 4). De Souza et al. (2015) have used simple burndown charts to show progress in each sprint in a capstone project course. Woodward et al. (2013) have similarly used burndown charts to help students track their progress, though they are not limited to teaching software process or projects.

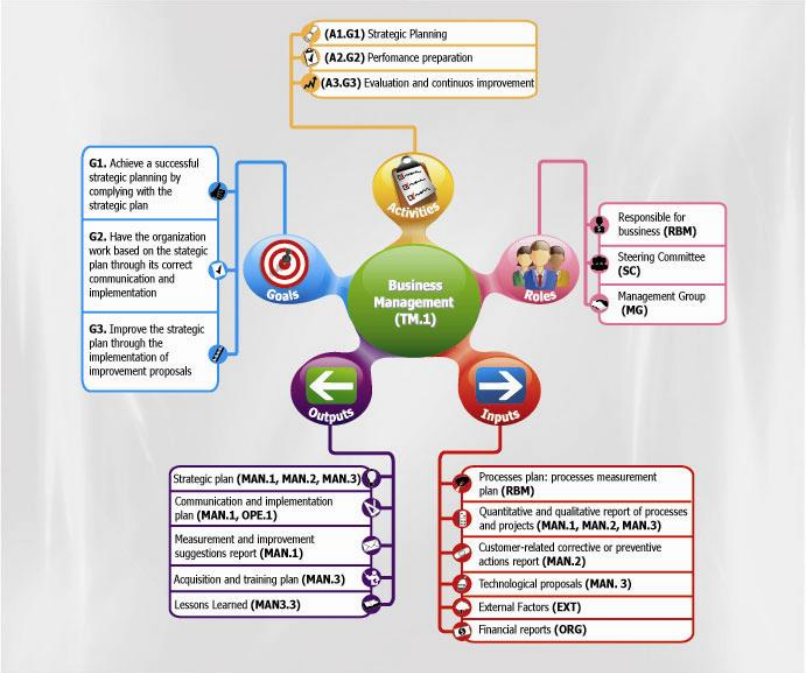

Figure 5: A second-level diamod (Espinosa-Curiel et al., 2010)

There are some approaches using specialized visualizations. Espinosa-Curiel et al. (2010) had the need to teach a very specific software process reference. Their audience was mainly companies, but the approach was teaching/learning oriented with a pedagogical approach. They adopted and created dimods – specialized diagrams based on mindmaps, role activity diagrams, rich picture technique and IDEF diagrams. Dimods of various levels of detail are used to portray and teach a textbook process, not to track how the process is advancing – example diamods of second and third level are given in Figure 5 and Figure 6, respectively. Their goal is to help understand the goals, activities, roles, inputs and outputs of the processes and sub-processes within the process model.

Matthies et al. (2016) used ScrumLint to check for process violations on a software project course using Scrum. ScrumLint enables comparing teams and there progress, and provides. e.g., radar graphs where teams' performance in metrics is visualized (see Figure 6), along with simpler line charts. Matthies et al. only used the tool post-hoc as a way to supplement their survey and tutor-based evaluations of students' performance, but discuss how the tool could be helpful already during the course in showing teams how they are performing.

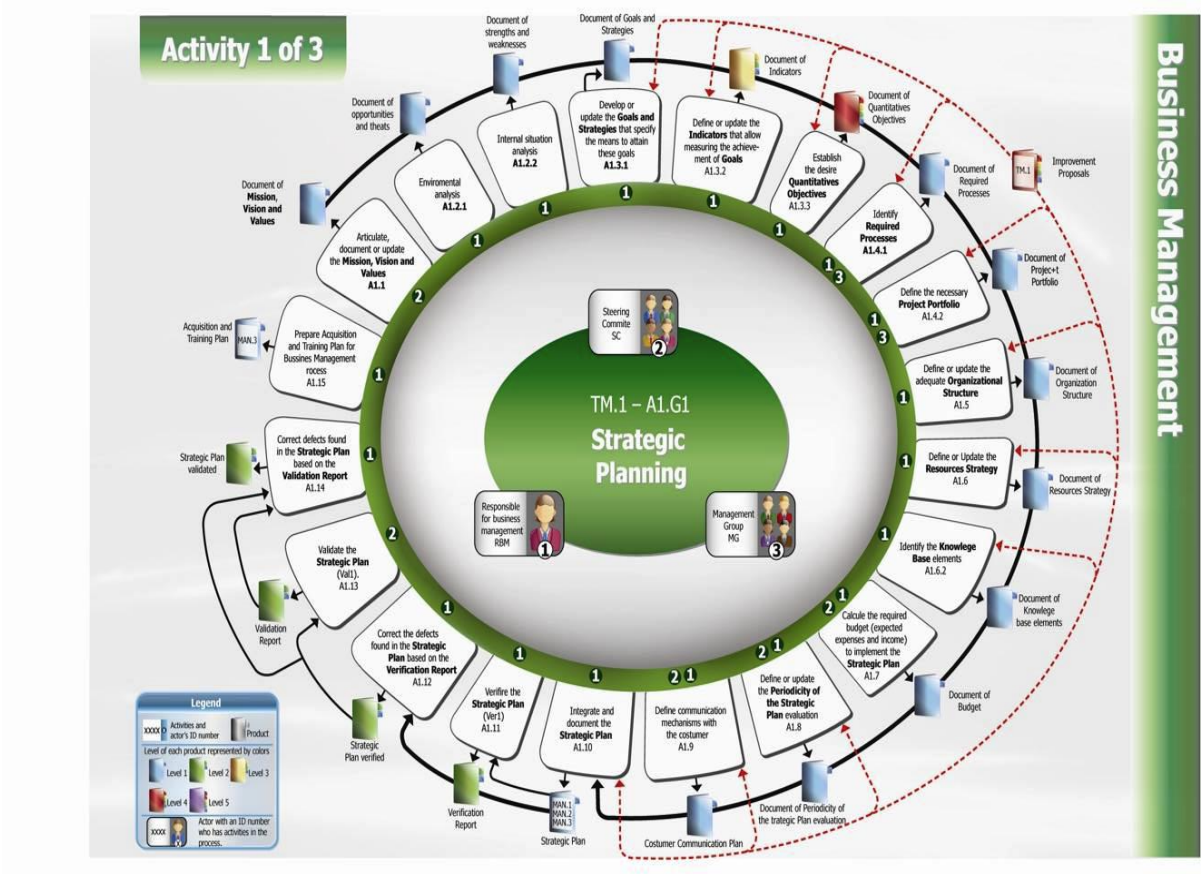

Figure 6: A third-level diamod (Espinosa-Curiel et al., 2010)

To summarize, using visualizations in teaching software processes or project management has so far been scarce. Gantt charts are used quite commonly in project management simulators, but not with live data. There have been some attempts at using burndown charts to track students' progress, but results are limited. One study used ScrumLint, which provides various charts and graphs, to check for process violations, but only utilized the tool in a post-hoc manner.

There are no approaches yet that would be truly comparable with the goals of VISDOM. No studies could be found where visualizations would have been used in the context of teaching software engineering processes or project management using real data from various tools and combining that data with informative visualizations. Further, existing studies have mainly used visualizations to illustrate one particular aspect, such as the duration of the projects or the progress, while visualizations in VISDOM can potentially be used to teach more complex issues related to agile software processes, actualization of sprints, and so on.

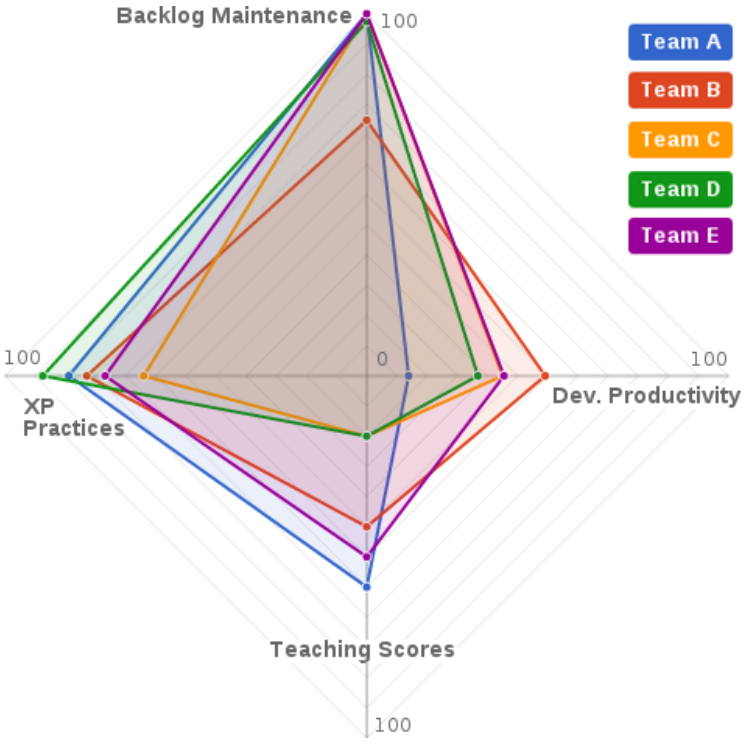

Figure 7: Radar chart provided by ScrumLint (Matthies et al., 2016).

3. STATE-OF-THE-ART ON THE DEVOPS & SUPPORTING TOOLS

As mentioned before, VISDOM project proposes novel visualizations to improve DevOps-based software development and project management in other industries with similar methodologies.

By definition, DevOps is a software development approach that encourages collaboration between software development and operations with the purpose of accelerating the delivery of software changes to production and ensuring resilient operation of the system in production (Penners & Dyck, 2015).

In DevOps the goal is to have minimum or preferably no manual intervention in management of the deployment pipeline, as shown in Figure 8. Therefore, once new software changes have been made and committed by software developers, they are automatically tested, integrated to create a new build when all test have successfully passed. The new build created by continuous integration server is automatically deployed to subsequent environments for additional tests, such as acceptance and performance prior to deployment in production. Such a way of working is increasingly becoming assimilated in software development companies (Puppet, 2017; Kratzke & Quint, 2017). Particularly in the development of cloud application where there is availability of vast tool-chain support for building the deployment pipelines that the teams take into use when adopting DevOps (Cito et all, 2015; Mäkinen et all, 2016). Despite the wide selection of tools presently available for DevOps, a number of challenges are still hindering companies from implementing DevOps and achieving full automation in deployment pipelines, particularly for large teams (Lwakatare, 2017). Some of these challenges include: insufficient test automation, limited automation of the network, lack of cross-discipline collaboration to help build the deployment pipelines and high learning curve for new tools (Lwakatare, 2017; Laukkanen, 2017; Bartusevics, 2017; Puppet, 2017).

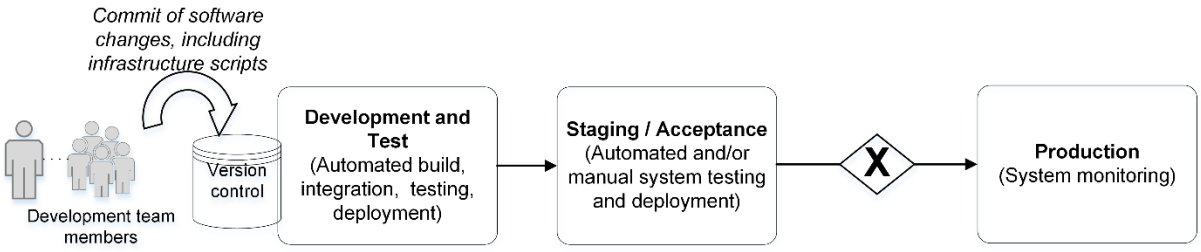

Figure 8: Continuous deployment pipeline

This tool- and automation-centric nature of DevOps creates huge possibilities for visualizations and dashboards. Continuous monitoring of software across the different environments, including production, becomes an important activity that is not just done by operations but also the developers of software project. Continuous monitoring of the software by developers serves various purposes, such as identifying software failures both at runtime and during post-mortems, identifying performance problems, capacity planning and user reactions to new software features. As such, it is quite common to find different monitoring dashboards in development teams' working environment when DevOps approach is adopted. However, for software developers to efficiently do said monitoring, several challenges need to be solved, including isolation of different monitoring tools, limitations in monitoring tools (especially when microservice architecture is used), as well as limited usage of post-deployment data as feedback given to different stakeholders of software project. There

are initial results on what kind of customer data can be used as feedback at different phases of the software development process (Sauvola, 2015).

Tools and automation are part and parcel of DevOps and DataOps is no different. Figure below provides example of tools and environments used in DataOps to provide value for the customer/end user.

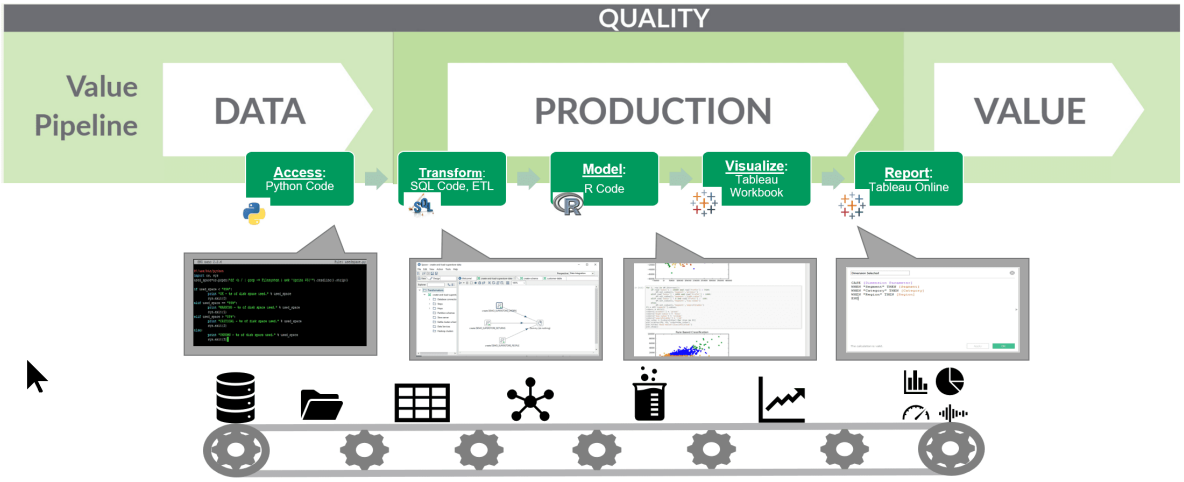

Figure 9: Example of tools used in DataOps Value Pipeline[1]

As a result, and as mentioned in the Introduction, one of the challenges to overcome is there will be data coming from different sources that the system needs to combine. Visualizations and other analyses cannot use data directly from individual tools. If data formats were standardized, it would be much easier to perform analyses from various data sources.

If all the tools used a standardized data format, the whole data utilization process — not only the analysis — could be based on the common data. However, common data format is not a reality at the moment and since the set of used tools are expanding from software engineering domain to other areas like business management and customer case. Thus, we cannot assume integrated or even interoperable data store in the near future either, and we propose the use of advanced data collection management methods to support creating common views from the multiple data sources.

Open Services for Lifecycle Collaboration (OSLC) is an emerging standard tool for interoperability. It has been widely used for tool integration in public funded research projects, and plugins are available for several tools. OASIS OSLC proposes the application of web principles to software interconnection. These specifications allow to conform independent software and product lifecycle tools to integrate the data and workflows in support of end-to-end lifecycle processes. However, the use of OSLC is not an industrial practice yet, and has not been used for data collection in a manner similar to the VISDOM approach. Further in this chapter the applicability of OSLC is covered in more detail.

[1] Source: http://dataopsmanifesto.org/index.html

3.1. DevOps Supporting Tools

There is a wide variety of tools used in DevOps, for example XebiaLabs lists 120 tool[2] sand the list is far from complete.

Software tools used in the DevOps pipeline produce log or job result data typically in JSON or XML format, although some, for example build tools, often produce plain text logs. Some of the tools have built-in visualization capabilities.

Data models are typically tool specific, unit testing frameworks being an exception as the xUnit-format is widely adopted.

Key findings:

- Data models are tool specific, except in Unit Testing area
- JSON, XML and plain text formats
- Data rarely contain metadata or context information
- Data model/format specifications not always available

A Continuous Delivery (CD) pipeline comprises multiple stages where each stage uses one or more specialized tools meaning that a typical pipeline uses dozens of tools. Figure 10 depicts an example of a logical CD pipeline.

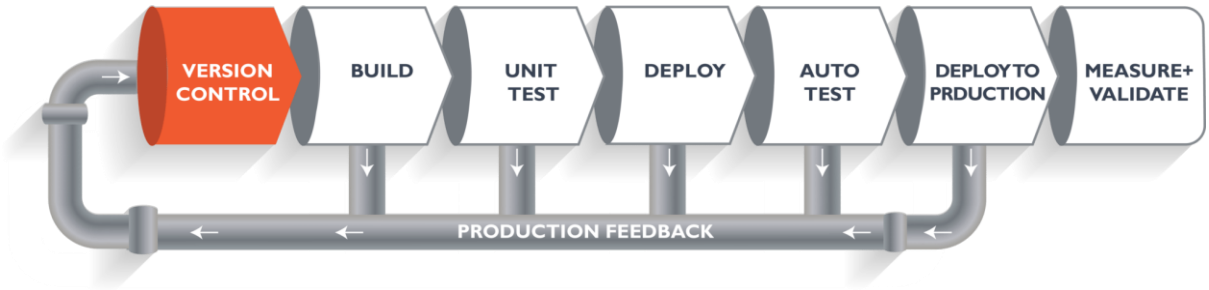

Figure 10: CD logical pipeline example, source dzone.com

Automation tools that orchestrate the CD pipeline or parts of it can collect the data from the pipeline and can produce reports and visualizations based on the data. These tools can also provide the collected data via a REST API or data export functionality. The typical data representation relies on JSON or XML data formats or they have a REST API available with a data format depending on the tool and usage. Similar to other tools used in DevOps, the data model itself appears to be tool specific and tailored for the tool specific use. For example, the lack of metadata becomes an issue if data from different sources is combined. Furthermore, the data is only partial and access to the full data requires either external plugins or other means to access the data.

Data Models Used

- **Linked Data**

Linked data as defined by W3C is a set of best practices, specifications and technologies for publishing structured data in Web[3].

[2] https://xebialabs.com/periodic-table-of-devops-tools/
[3] https://www.w3.org/standards/semanticweb/data#specifications

Linked Data's data model is based on the Resource Description Framework (RDF), which is a framework for describing resources. RDF itself is an Entity-Relation model based on subject-predicate-object triplets describing entities and their relations.

- **OData**

OData (Open Data Protocol) [OD] is an OASIS standard that defines a set of technologies and best practices for REST APIs.

An OData data model is defined using Common Schema Definition Language (CSDL). Some DevOps tools such as Microsoft Azure DevOps use OData for exposing CD pipeline artefacts.

The main difference between OData and W3C Linked Data approaches is that the OData does not have mechanism for linking data from different sources.

- **JSON schema**

A JSON schema [JS], currently an IETF draft, is a specification for defining the structure of JSON data. It provides a mechanism to describe contents and validation rules for JSON documents in human and machine-readable format.

Combining Data from Various Sources

While data models are typically tool specific, there are initial studies on how to create a unified model that would enable combining and particularly visualizing data from different tools, such as issue management systems, version control systems, and usage monitoring platforms, with one comprehensive data model. Mattila et al. (2015a) describe combining events from aforementioned tools under one general term - Software Engineering Event. Key elements of a Software Engineering Event are Event type, Feature and Sate. In an empirical study, data was collected data from Jira, Mercurial and Splunk. Using this unified model, a visualization was created where the authors were able to show the development progress and usage information for individual features alongside information of the status of the whole project.

The data model has been further developed (Mattila et al. 2015b), most notably by replacing the concept of "Feature" with "Artifact", which allows using the datamodel on an even wider scope. A proof-of-concept tool has been created based on the revised model, showing successful mapping of data from GitHub, Jira, test logs and usage via a REST API, and providing a timeline visualization based on various data. APIs for collecting issue management data are described in detail in Hylli et al. (2015).

A similar approach to unifying software development data for visualizations has also been presented by Benomar et al. (2015), whose motivation is to understand the time dimension of software execution. However, their approach lacks the vision of combining data from various sources.

OSLC

Open Services for Lifecycle Collaboration (OSLC), an OASIS open project[4], is a community defining a set of specifications that enable interoperability and integration of software development tools. OSLC is based on W3C Linked Data principles and technologies and uses the Linked Data Platform (LDP) client–server RESTful architecture.

OSLC specifications are organized in core and domain specifications.

[4] https://www.odata.org/

- OSLC Core Specification

OSLC core specifications define the common capabilities, patterns and protocols of OSLC clients and servers used across the domains.

- OSLC Domain Specifications

OSCL domain specifications cover the needs for a specific domain and define the required services and RDF resources. Domain specifications are scenario-driven meaning that they specify only minimal set of properties required for specified scenarios, but they may be extended as needed to cover new usage scenarios. Table 1 lists the available domain specifications and their intended use.

Table 1: OSLC Domain Specifications[5]

Specification	**Description**
Architecture Management	Defines the OSLC services and vocabulary for the Architecture Management domain.
Asset Management	Defines the OSLC services and vocabulary for the Asset Management domain.
Automation	Defines the OSLC services and vocabulary for the domain that supports automation of sequences of actions on OSLC resources.
Change Management	Defines the OSLC services and vocabulary for the Change Management domain.
Configuration Management	Defines the OSLC services and vocabulary for managing versions and configurations of linked data resources from multiple domains
Performance Management	Defines the OSLC services and vocabulary for the Performance Monitoring domain.
Project Management (PROMOCODE)	Defines OSLC services and vocabulary for exchanging project management information across organizational boundaries.
Quality Management	Defines the OSLC services and vocabulary for the Quality Management domain.
Requirements Management	Defines the OSLC services and vocabulary for the Requirements Management domain.
Tracked Resource Set	Allows servers to expose a set of resources whose state can be tracked by clients.

Each domain specification defines a data model for domain specific use cases. They are based on RDF and consist of vocabularies describing resource types and properties and resource shapes which define required representation and constraints.

Tool support of OSLC

OSLC standard assumes that the tool support by their data in OSLC format.
Several proprietary and open source tools used in the CD pipeline have some level of OSLC support via plugins or adapters.

[5] https://open-services.net/specifications/

The Eclipse Lyo project[6] provides a Java SDK for creating OSLC compliant tools. The project also provides a modelling tool that can be used to model RDF resources.

Suitability of OSLC for VISDOM

OSLC has been designed for tool integration use cases and does not cover data collection or the visualization use cases needed in VISDOM project. In addition, visualizations in Business Intelligence systems are typically based on separate data storage. However, existing domain specifications or their extensions could be used for some CD pipeline artefacts but for some artefacts new domain specifications are needed. The amount of required modifications and extensions depends on data required in use cases.

TOSCA

OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) "enables the interoperable description of application and infrastructure cloud services, the relationships between parts of the service, and the operational behavior of these services (e.g., deploy, patch, shutdown)--independent of the supplier creating the service, and any particular cloud provider or hosting technology. TOSCA will also make it possible for higher-level operational behavior to be associated with cloud infrastructure management"[7].
Since cloud operation and architecture are important part of DevOps development the data interoperability standards of that field are relevant to VISDOM, too.

[6] https://projects.eclipse.org/projects/technology.lyo
[7] https://www.oasis-open.org/committees/tosca/faq.php

4. COMPETENCE ANALYSIS ON SOFTWARE ANALYTICS AND VISUALIZATION

Time to market is key in today's technology-driven society. If a product enters the market too late, a competitor will eat its market share. This is especially true for software because technology and thus new opportunities evolve quickly in this field.
One possible downside of developing software under time pressure is that insufficient effort is put into making sure the quality of the software is sufficient. This might result in field defects and possible recalls of the product.
Fortunately, there are tools available in the market that monitor the quality of software during development. This helps development teams to adjust their software in time before release.
In this chapter we will focus on visualization techniques that are used by the state-of-the-art tools for monitoring software quality.
There is a big amount of commercial and academic or community-oriented software analytics tools that are available on the market. This report includes the result of analysing some of these tools.
For most tools, there is a summary giving concrete details about:

- Functionalities: Main functionality.
- Monitoring: Kind of monitored data.
- Data sources: Concrete data source tools from the data is gathered.
- Pricing: Commercial when you need to pay and Community when is an Open Source Software solution

4.1. Software Analytics based on Commercial Approaches

Commercial approaches provide software analytics solutions that monitors concrete quality aspects through interactive dashboards. The monitored quality aspects are mostly related with the data produced during the software development process (e.g. monitoring metrics related to static code analysis), and some of them aggregates these metrics in more abstract quality aspects, e.g., SonarCloud aggregates metrics into security and reliability indicators, among others.

The main functionality of these tools is the monitoring and visualization of indicators related to concrete software development quality aspects, e.g., code quality, DevOps flow, risk, and security. All of them provide to the user a graphical interface to visualise the evaluation of these indicators, i.e. interactive dashboards. The indicators evaluation can be complemented with some information that can help decision-makers to take concrete actions, e.g. list of open issues with priority.

Table 2. Commercial Approaches Comparison

	Bitergia	Datadog	Kiuwan	New Relic	SonarQube	Splunk	TaskTop
Monitoring	OSS projects	DevOps	DeveOps (Risk/Security)	DevOps	Software Product	IT/Security/IoT/Bussiness	DevOps (Flow)
Functionality							
Monitor	✓	✓	✓	✓	✓	✓	✓
Visualization	✓	✓	✓	✓	✓	✓	✓

	Discussions		✓					
	Alerts		✓		✓		✓	
	Action plan			✓		✓	✓	
	Simulation			✓				
	Prediction						✓	
	Integration visualization							✓
Data Sources								
	CI[8]	✓	✓			✓		✓
	Code Review	✓	✓					
	Communication Channels	✓	✓					
	Downloads	✓	✓					
	Documentation	✓	✓					
	Events				✓			
	Infrastructure				✓		✓	
	Logs				✓			
	SaaS/Cloud		✓					
	Source Code	✓	✓	✓		✓		
	Traces				✓			
	Tickets/issues	✓	✓			✓		✓
Pricing[9]		C/OSS	C/Free	C	C	C/OSS	C/Free	C

Some of these tools also provides complementary features. Kiuwan provides the functionality of simulating how the indicators would be affected depending on concrete metrics or effort. Splunk provides prediction applying some forecasting algorithms to visualise future values. Tasktop includes views for visualising the DevOps tools integration, i.e. how and which data is shared among tools used in DevOps processes.

Table 2 summarises the analysed aspects for each tool.

[8] CI: Continous Integration
[9] C: Commertial. OSS: Open Source Software version, Free: Free version, no OSS

4.1.1. Bitergia

Bitergia[10] provides a dashboard for monitoring Open Source Software (OSS) projects (see Figure 11). The monitored aspects are related to the community behind the project. In the sense of development, the aspects are similar to aspects that can be monitored in a non-OSS development team, e.g., development performance, complemented with some specific indicators to analyse the community maturity, e.g., size. Some of these indicators can help decision-makers to decide adopting or not a concrete OSS project.

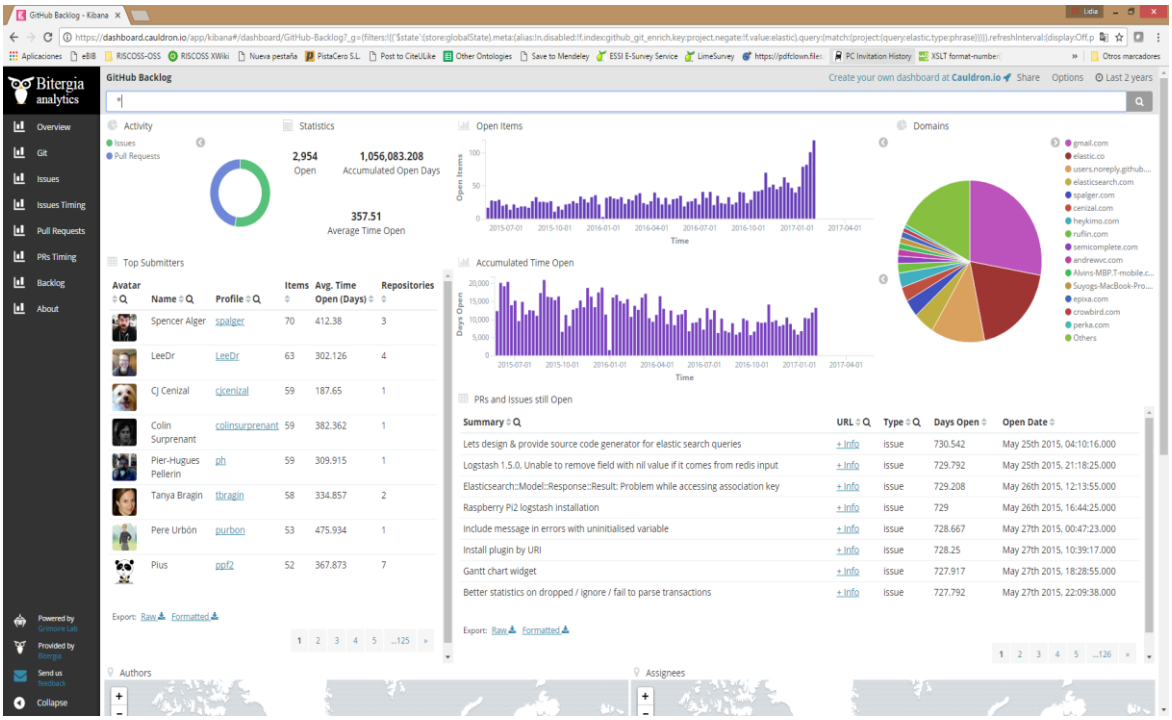

Figure 11. Bitergia Dashboard

Summary:

- *Functionalities:* OSS analysis
- *Monitoring*: activity, performance, diversity, size, and demography.
- *Data sources*: source code (e.g., git), tickets/issues (e.g. Jira), code review (e.g., Gerrit), continous integration (e.g., Jenkings), mailing lists/forums, chat, downloads, wiki, meetings, and others.
- *Pricing*: community and commercial versions

4.1.2. Datadog

Datadog[11] aggregates metrics and events across the full DevOps stack. It allows to customise dashboards to include concrete charts (see Figure 12).

[10] https://bitergia.com/
[11] https://www.datadoghq.com/

Figure 12. Datadog Dashboard

Summary:

- *Functionality*: monitoring and visualization, team discussion, performance alerts
- *Monitoring*: DevOps stack (servers, clouds, metrics, apps, and team).
- *Data sources*: more than 350 integrations[12], e.g., SaaS and cloud providers, automation tools, monitoring and instrumentation, and source control and bug tracking.
- *Pricing*: commercial (from $1.27 to $31 per month depending on the package). Infrastructure monitoring offers a free package option.

[12] https://www.datadoghq.com/product/integrations/#all

4.1.3. Kiuwan

Kiuwan[13] provides security solutions for DevOps processes (risk and security rating). It has been developed by the Spanish company Optimyth. Kiuwan is rather new in the market and offers a full Software as a Service (SaaS) solution for measuring code quality. Their primary focus is to check the security of software but the platform has also static code analysis capabilities, called Code Analysis.

It provides a dashboard analysing security in the context to secure code and architecture. It analyses code vulnerabilities and also provides some advices related to good practices. The analysis provides results at development time. Figure 13 shows the vulnerabilities dashboard (left) and details (right).

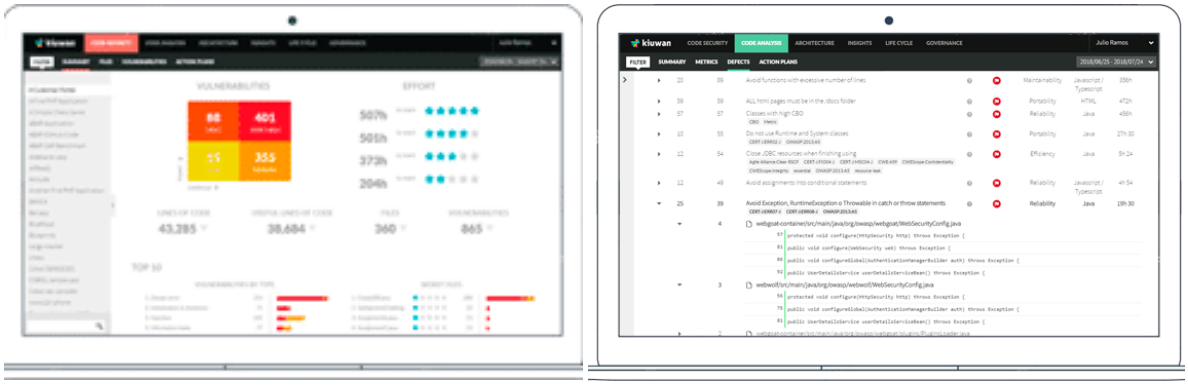

Figure 13. Kiuwan Dashboards

Kiuwan also provides simulation capabilities, in the context of adding resources or rating indicators (see Figure 14).

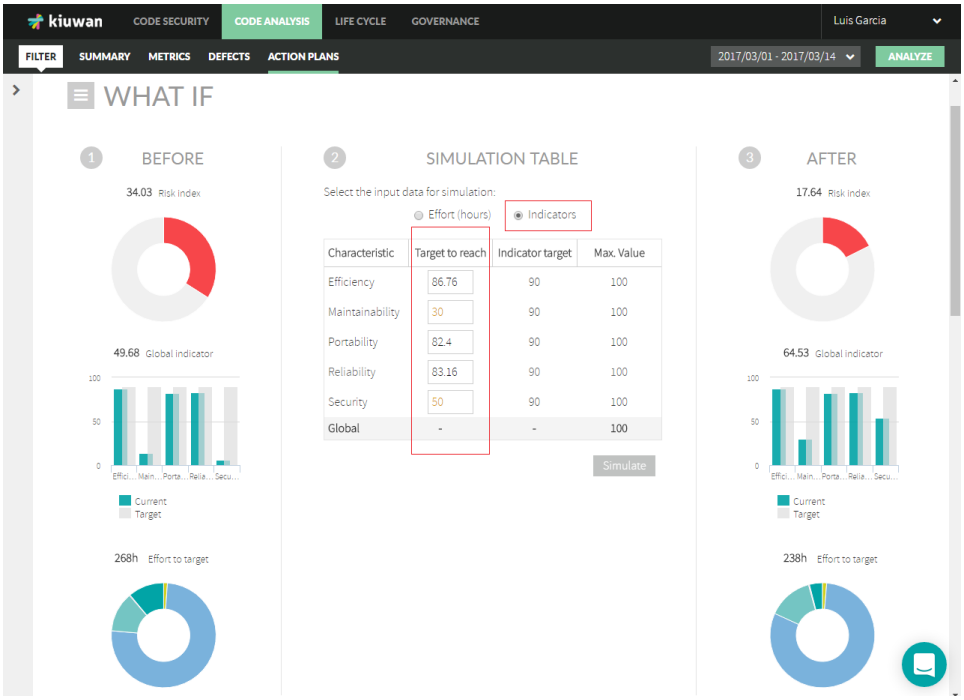

Figure 14. Kiuwan simulation feature

Summary:

- *Functionalities*: code analysis, what-if analysis, action plans suggestions (concrete tasks), and vulnerability propagation paths. It also includes some extra features, e.g., product comparison

[13] https://www.kiuwan.com/

- *Monitoring*: risk and security rating
- *Data sources*: souce code
- *Pricing*: Commercial ($599 to $2550 and on request)

4.1.4. New Relic

New Relic[14] is a software analytics platform to observe software quality including metric visualization, tracing data from services, and combining log data with application and system performance data to correlate the log data with the infrastructure events. (see Figure 15).

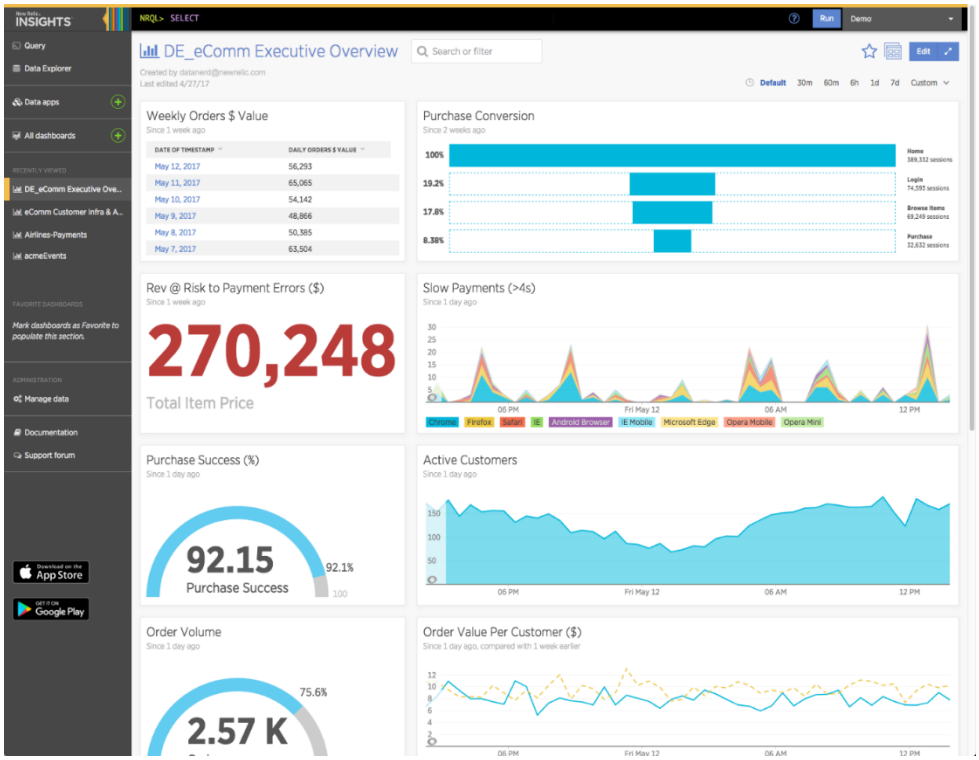

Figure 15. New Relic Dashboard

Summary:

- *Functionalities*: software data analysis, customizable dashboards, querying the data
- *Monitoring*: sofware product and infrastructure
- *Data sources*: APM events (e.g., transaction, span), browser events (e.g. page action, ajaxRequest), infrastructure events (e.g. operating system events like CPU and memory, amazon AWS EC2 attributes), mobile (e.g, mobile crash, mobile request) , and Synthetics (e.g, syntheticCheck)
- *Pricing*: Commercial

4.1.5. SonarQube

The SonarQube[15] tooling is the world's leading software quality dashboard by far. SonarQube is based on the ISO/IEC 25010 standard (see Figure 16).

[14] https://newrelic.com/platform
[15] https://www.sonarsource.com/

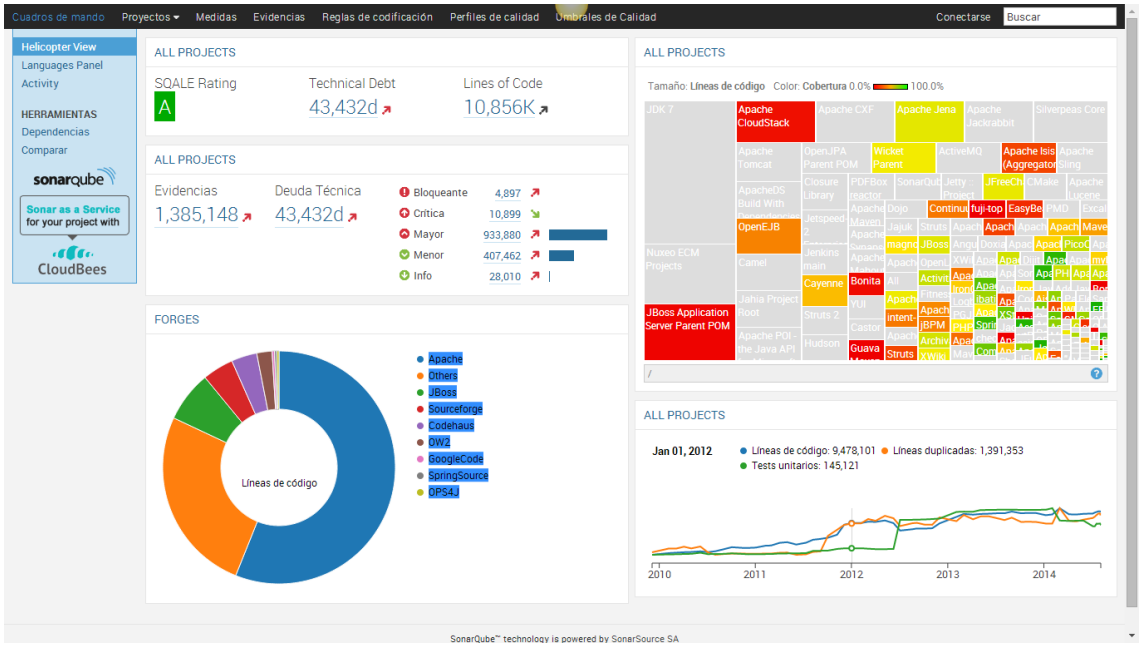

Figure 16: SonarQube Dashboard

The basic license of SonarQube is free (open source) to use but more and more plugins are becoming commercial. SonarQube supports about 20 programming languages. It is used in all market segments by all kinds of stakeholders, especially software project teams, because of its ease of use and ease of configuration.

SonarQube[15] provides continuous inspection on the source code, providing mechanisms to configure what they name “quality gates”. Quality gates allow the user to define a criterion to evaluate the analysis results in the context of passed or failed.

SonarQube analysis is based on the SQALE method (Letouzey and Ilkiewicz 2012), assessing the following concrete quality aspects: reliability, security, maintainability, coverage and duplications. As part of their products, SonarQube offers SonarCloud, a SonarQube version analysing OSS projects available in GitHub, Bitbucket and Azure DevOps.

Figure 17 shows a SonarCloud dashboard including the values for the different quality aspects with the corresponding rating (from A to D).

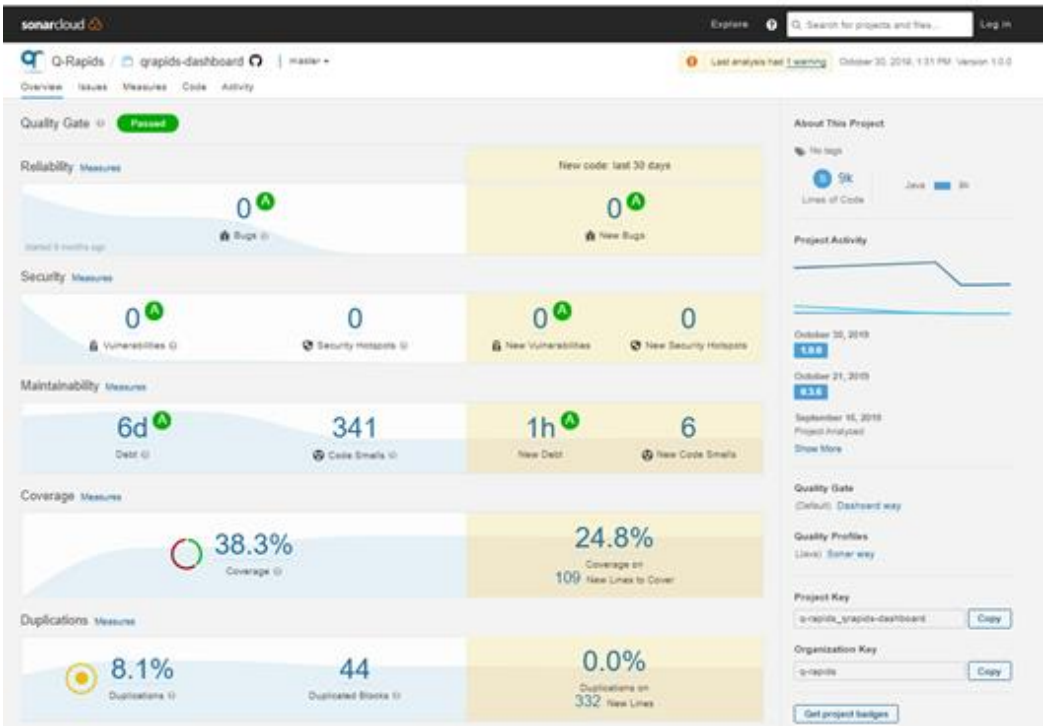

Figure 17. SonarCloud Dashboard

The information included in the dashboard is complemented with some graphical visualizations (see Figure 18) and links to the code that can be improved (Figure 19).

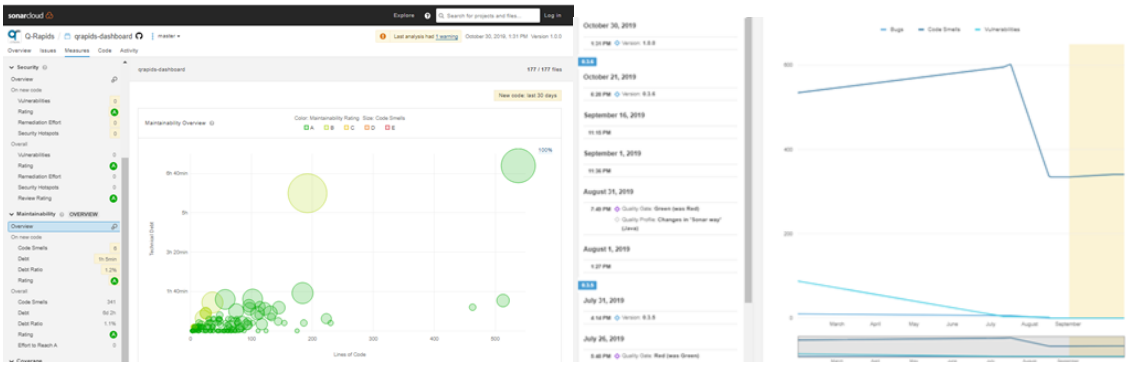

Figure 18. SonarCloud charts

Figure 19. SonarCloud code improvement suggestion

Summary:

- *Functionalities*: source code analysis and action plan suggestion (concrete tasks)
- *Monitoring*: reliability, application security, and technical debt
- *Data sources*: source Code (e.g, Git) for source code analysis (20+ languages) and continous integration tools (e.g, Jenkins)
- *Pricing*: community and commercial (from $6.000 to $15.000 per year and on request)

4.1.6. Splunk

Splunk[16] collects, analyses and correlates data generated by technology infrastructure, security systems, and business applications. Splunk solutions provides solutions for IT operations (including monitoring performance of infrastructure and applications), Security (security risk management), IoT, and business analytics. Splunk allows to drill down from high-level dashboards to concrete elements (e.g., tasks, log files) analysed.

Figure 20 shows a Splunk dashboard providing a complete view of the organization's systems, devices, and interactions (executive operations view). It also has business flows visualisations (Figure 21).

[16] https://www.splunk.com/

Figure 20. Splunk Dashboard

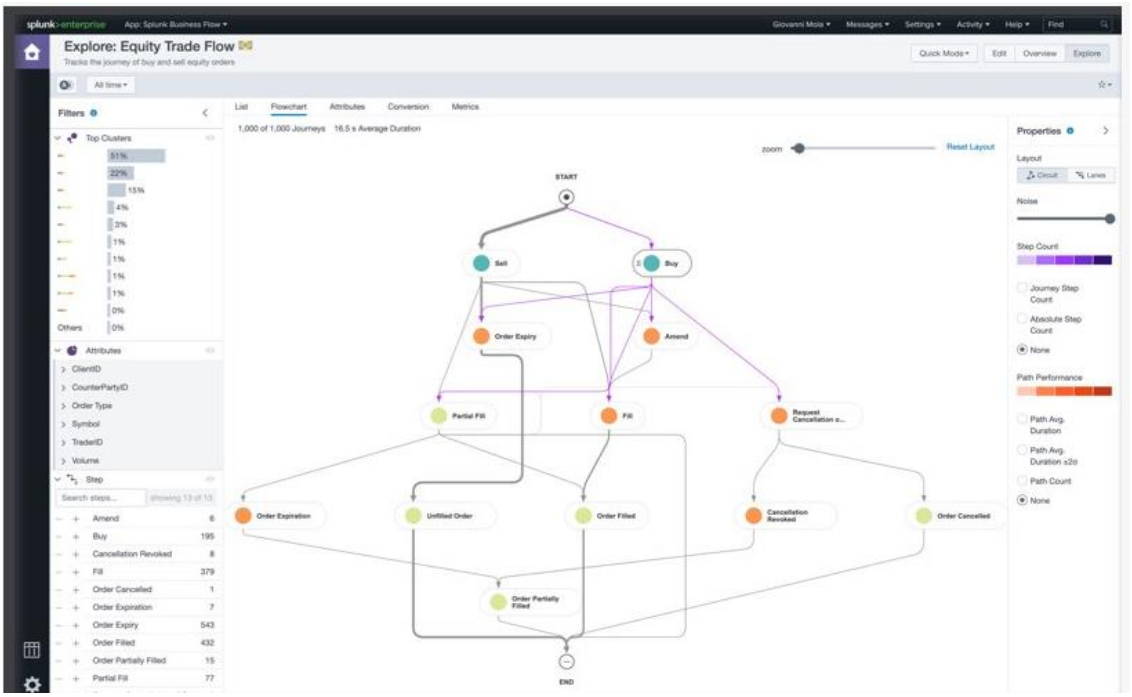

Figure 21. Splunk business processes

Splunk also provides prediction for the next several time steps, using several forecasting algorithms, producing some visual charts (see Figure 22).

Figure 22. Splunk prediction

Summary:

- *Functionality:* monitoring and visualisation, alerts, prediction capabilities (machine learning).
- *Monitoring*: IT (infrastructure, application), security (e.g., priority issues, security related incidents), IoT, and business.
- *Data sources*: No specified
- *Pricing*: Free (Splunk free) and Commercial ($173 per month for Splunk enterprise, another options price on request)

4.1.7. Tasktop

Tasktop[17] offers a turnkey solution to implement the Flow Framework, it is designed for business users. Connect the delivery tools and operation tools to automate information flow.

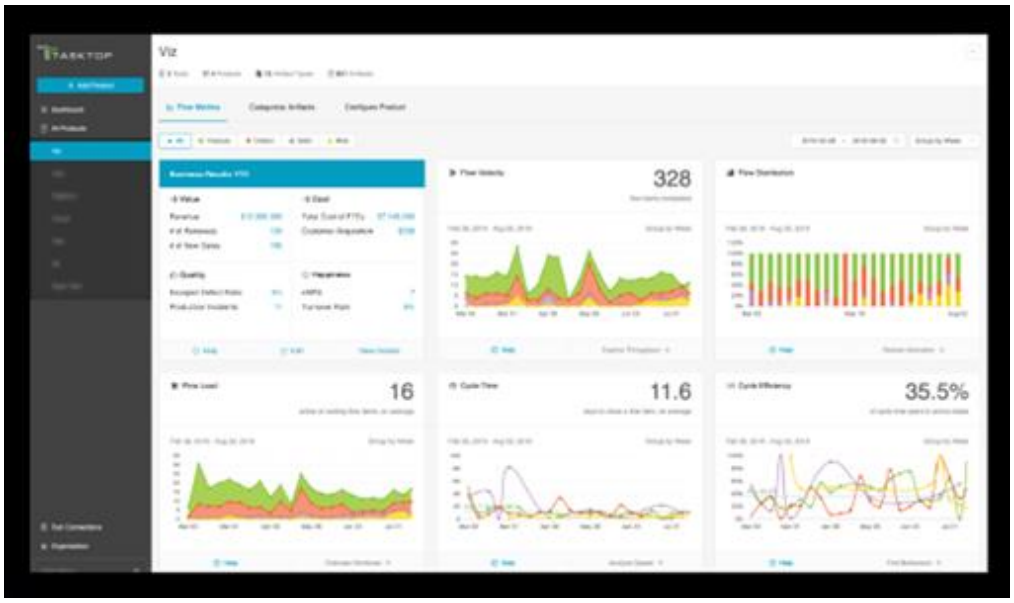

Figure 23. Tasktop Dashboard

Tasktop includes visualization of the integrated tools, including the data flows among them (see Figure 24).

[17] https://www.tasktop.com/

Figure 24. Tasktop integrations visualization

Summary:

- *Functionality:* flow metrics monitoring and visualization, and value stream visualisation (integrations landscape)
- *Monitoring*: flow metrics
- *Data sources*[18]: common tools integration (e.g., Azure DevOps), supported lifecycle tools (e.g., github issues), and DevOps tools (e.g., ca technologies, Jenkings)
- *Pricing*: commercial (price on request)

4.1.8. CAST Software

CAST Software[19] is a French company that provides a high end solution to monitor software quality according to the ISO/IEC 25010 standard and beyond. CAST's flagship is the Application Intelligence Platform (AIP). It consists of various high-end dashboards for different roles and aspects in a company. The Health Dashboard (see picture below) for instance aims to help management to monitor the quality of their comp any's application landscape.

[18] https://www.tasktop.com/integrations
[19] CAST Software company. See https://www.castsoftware.com/.

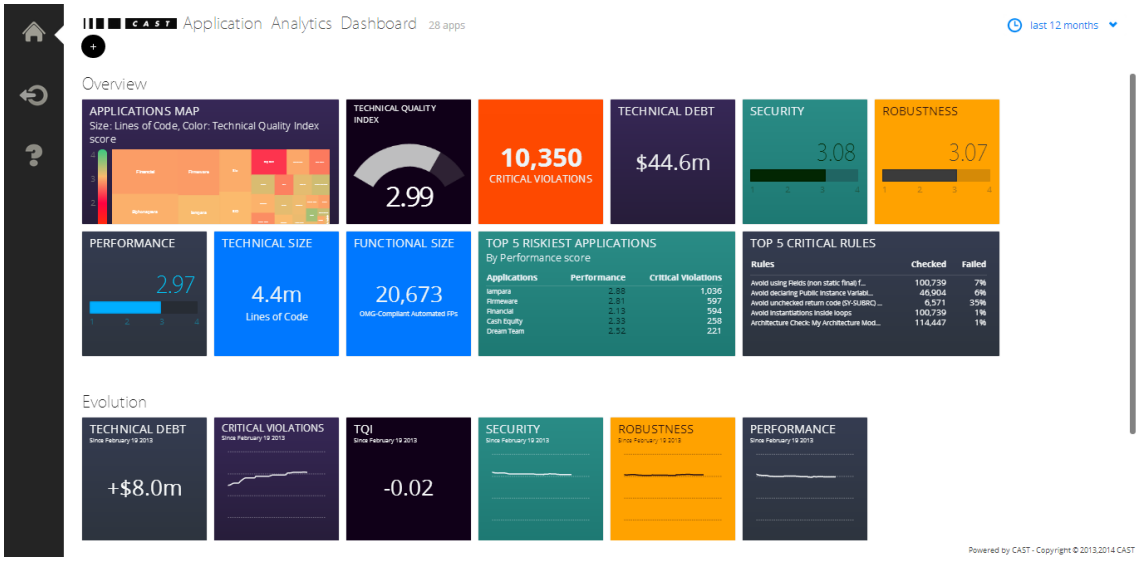

Figure 25: CAST SW Dashboard

The visualization techniques used by CAST are tables, heat maps, bar charts, and trend graphs.

Summary:

- *Functionality:* Fine-grained analysis engine that reverse engineers architecture, databases, frameworks, and transactions.
- *Monitoring*: individual SW components & interactions between the components
- *Data sources*[20]: Function Points (transactions) & Quality, Security and Sizing.
- *Pricing*: commercial

4.1.9. NDepend

NDepend is a plug-in in Microsoft's Visual Studio[21]. NDepend is available for all Visual Studio programming languages. It measures technical debt by using the SQALE methodology. However, technical debt parameters are configurable based on the actual circumstances of a software system.

[20] https://doc.castsoftware.com/display/DOC83/Covered+Technologies

[21] NDepend tooling. See https:/www.ndepend.com/.

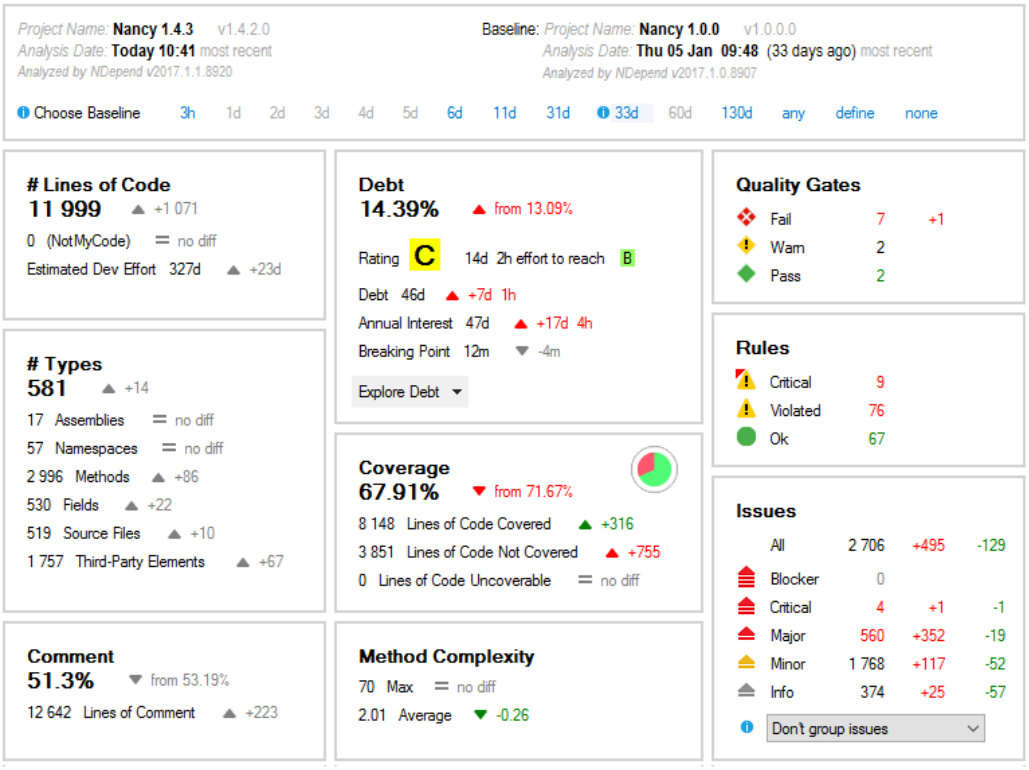

Figure 26: NDepend Dashboard

NDepend uses very simple visualization techniques, i.e. just displaying numbers.

4.1.10. Teamscale

The German company CQSE has developed the code quality dashboard Teamscale[22]. The Teamscale dashboard shows all kinds of software code, documentation and architecture metrics. The metrics can be calculated in a CI/CD environment or within the editor of a software developer.

[22] TeamScale dashboard. See https://www.cqse.eu/en/products/teamscale/landing/.

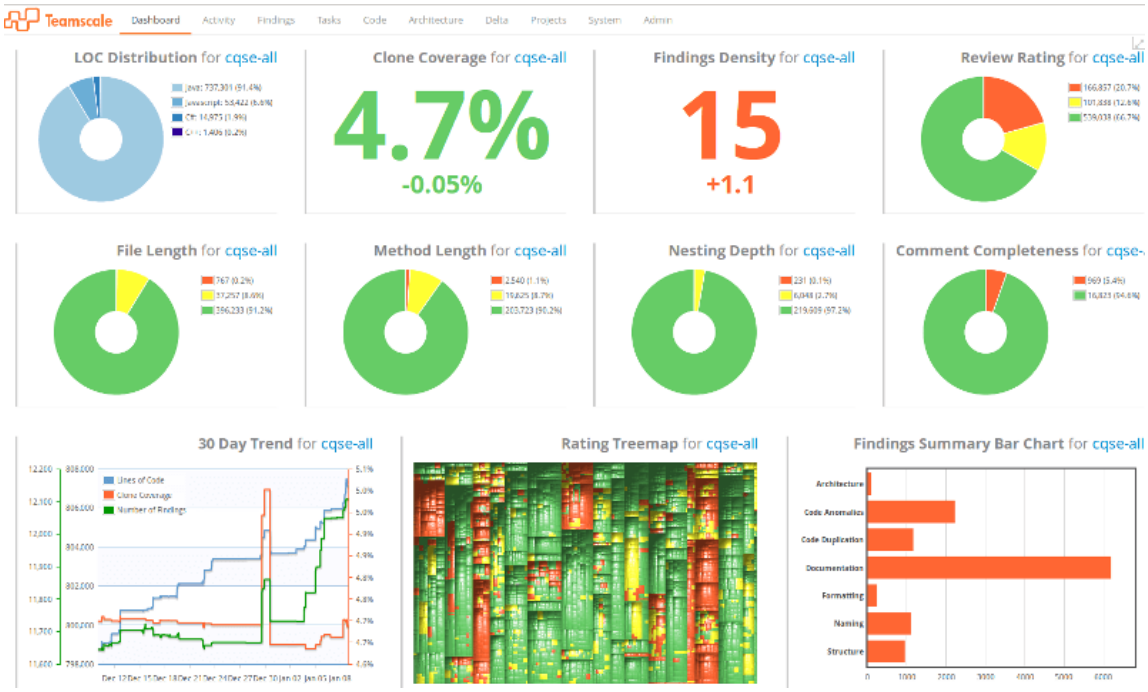

Figure 27: Teamscale Dashboard

The visualization techniques used by TeamScale are trend graphs, donut charts, heat maps and bar charts.

4.1.11. TiCS Framework

The TiCS framework[23] is a code quality dashboard to measure the TIOBE Quality Indicator (TQI), which is based on the ISO/IEC 25010 standard and contains technical debt metrics. TiCS has been developed by the Dutch company TIOBE Software. The dashboard is similar to SonarQube, SQuORE and Teamscale by showing trends, treemaps and drill downs to code level.

[23] TiCS dashboard. See https://www.tiobe.com/.

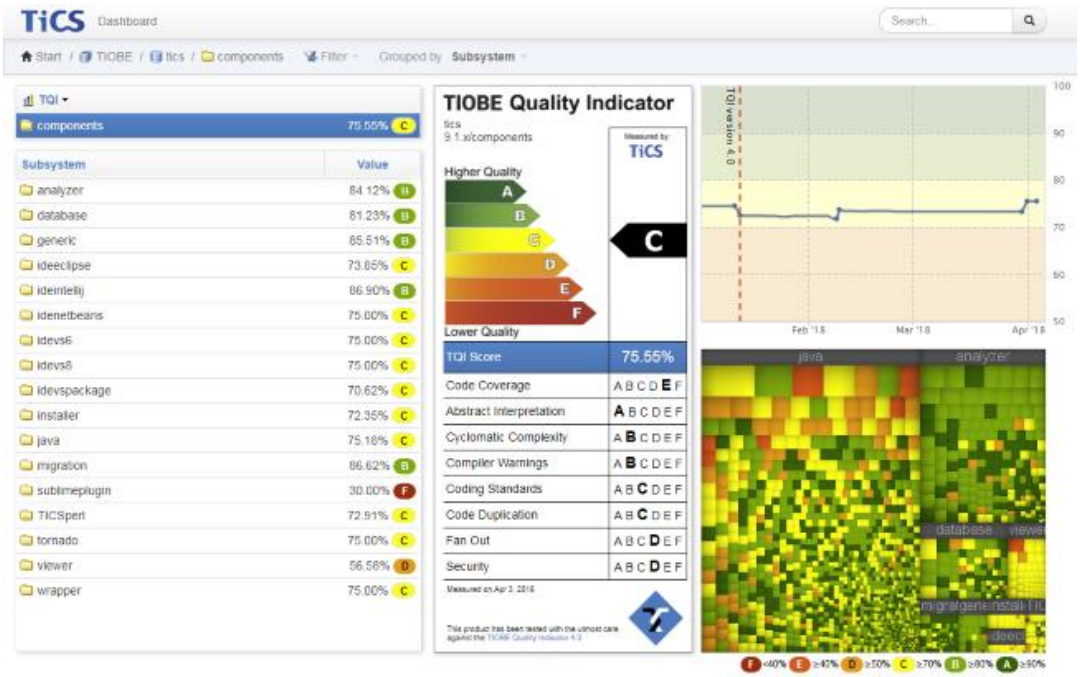

Figure 28: TiCS Framework Dashboard

The TiCS framework uses trend graphs, heat maps, labels, bar charts and scattered plots as visualization techniques.

Based on this analysis, we can conclude that market leaders in the software quality dashboard scene are all using straightforward techniques for displaying software quality. The techniques used are: bar charts, trend graphs, donut graphs, radar graphs, heat maps, pie charts, plain figures and labels.

There are newer experiments with newer techniques such as 3D heat maps, sun burst charts and complex node-link diagrams, but these are not yet adopted by the market leaders.

4.1.12. Grafana

Grafana[24] is an open source metric analytics and monitoring suite built to visualize a large amount of time-series from various databases. Grafana makes metrics visualization more accessible for DevOps operations, business analysts and product managers, having a commercial version called Grafana Enterprise.

Furthermore, Grafana enables unified monitoring for DevOps to compare performance data gathered by various management tools in one single fast view. As such, all relevant data across the organization are consolidated, making it possible to identify bottlenecks and predict problems early.

[24] Grafana platform https://grafana.com/

Figure 29: Grafana Dashboard

4.2. Software Analytics Community-oriented Approaches

Community-oriented approaches provide generic software analytics solutions that can be customised to monitor several quality aspects. All the solutions are structured in a way that they have specific software components to gather data, which can be customised by the adopter to be adapted to the organization tools.

The main functionality of these tools is monitoring indicators providing to the user an interface to visualise the evaluation of these indicators. All of the approaches, except CodeFeerds, provide a graphical interface for the indicators' visualization, i.e. a dashboard. Concretely, they provide mechanisms to create customizable interactive dashboards. The Q-Rapids tool provides the possibility of creating custom dashboards (using Kibana), but the main dashboard provides fixed visualizations, using the same kind of charts for all the monitored indicators. In this case, the tool needs to be customised defining how the concrete indicators are computed instead of how they are visualised.

Q-Rapids, complementing the dashboards visualising indicators, provides prediction and simulation of these indicators. The user can visualise the predicted values and set-up concrete scenarios in order to evaluate the impact on the indicators (e.g., setting up a concrete value for some metrics). The Q-Rapids dashboard has been developed to support decision-makers in the context of quality requirements (QR) management, therefore it provides the concrete functionality of receiving alerts, when indicators have low values, and suggesting quality requirements to address these alerts.

Table 3 summarises the analysed aspects for each tool.

Table 3. Community Approaches Details

		CodeFeedr	GrimoireLab	MEASURE	Q-Rapids
Monitoring			OSS Projects	Development	
Functionality					
	Monitor	✓	✓	✓	✓
	Visualization		✓	✓	✓
	Querying	✓			
	Correlations			✓	
	Discussions				
	Alerts				✓
	Action plan				✓
	Simulation				✓
	Prediction				✓

	Integration visualization				
Data Sources					
	CI		✓	✓	✓
	Code Review		✓	✓	✓
	Communication Channels		✓		
	Downloads				
	Documentation		✓		
	Events				
	Infrastructure				
	Logs		✓		
	SaaS/Cloud				
	Source Code		✓	✓	✓
	Traces				
	Tickets/issues		✓	✓	✓
Pricing		OSS	OSS	OSS	OSS

4.2.1. CodeFeedr

CodeFeedr[25] is a platform to ingest and process software analytics data, providing a query engine integrating data. This platform is produced by software analytics lab, as part of the software engineering research group from Delft University of Technology (The Netherlands).

The aim of the project is unifying data for software analytics under a single query language and apply new techniques for aggregation and summarisation of these data. Figure 30 shows the CodeFeedr vision.

The code is available in GitHub (https://github.com/codefeedr/codefeedr).

The information for this tool is scarce so not all the fields in Table 2 have been filled.

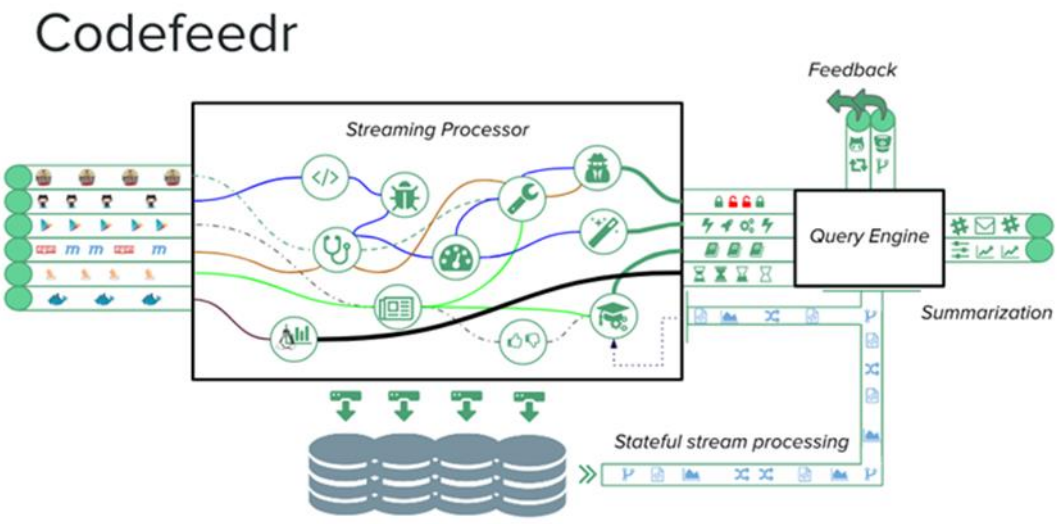

Figure 30. Codefeedr solution

[25] http://codefeedr.org/

4.2.2. GrimoireLab

The aim of GrimoireLab[26] is to provide an open source platform for automatic and incremental data gathering from almost any tool (data source) related with contributing to Open Source development (source code management, issue tracking systems, forums, etc). GrimoireLab is one of CHAOSS[27] Software founding projects. CHAOSS is a Linux Foundation project focused on creating analytics and metrics to help define community health.

Besides data gathering and analysis, it also provides automatic data enrichment, merging duplicated identities, adding additional information about contributor's affiliation, calculation delays, geographical data, etc. Data visualization tools allow also filtering by time range, project, repository, contributor, etc. Some commercial dashboards and services are built using GrimoireLab platform, for example the Bitergia analytics platform (see Section 4.1.1).

Summary:

- *Functionalities*: data gathering, data enrichment (e.g., merging duplicated identities, adding additional information), and data visualization.
- *Monitoring*: OSS projects
- *Data sources:* Askbot site, Bugzilla, Confluence, Discourse site, Gerrit, Git, GitHub, HyperKitty archiver, Jenkins, JIRA, MBox files, MediaWiki site, Meetup group, NNTP news group, Phabricator site, Pipermail archiver, Redmine, RSS feed server, Slack channels, StackExchange sites, Supybot log files, Telegram
- *Pricing*: community
- *OSS Repository*: https://github.com/grimoirelab

4.2.3. MEASURE

MEASURE[28] is an ITEA project involving twelve partners from four countries that aims to implement a set of tools for automated and continuous measurement. The Measure Platform allows you to collect, combine, visualize, analyse and share your metrics.

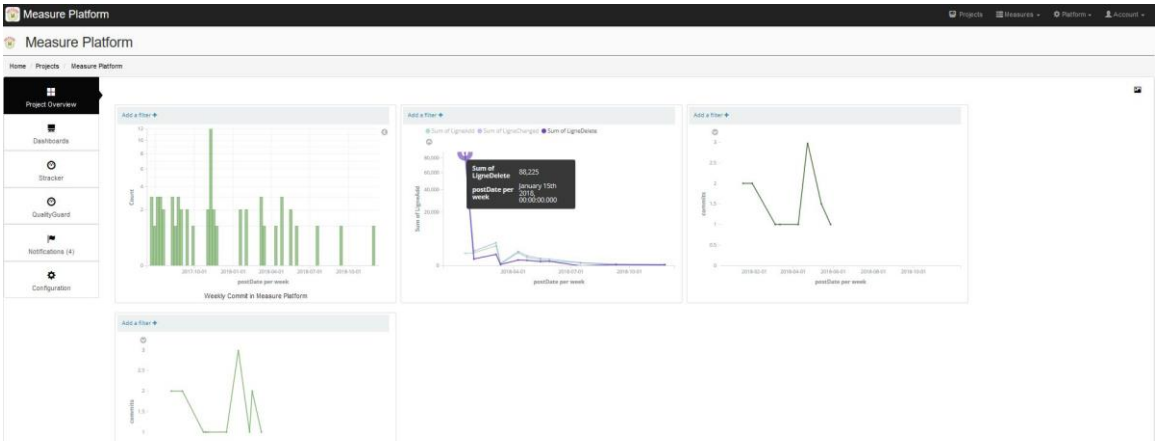

Figure 31. MEASURE Platform Dashboard

Summary:

[26] https://chaoss.github.io/grimoirelab/
[27] https://chaoss.community/software/
[28] http://measure-platform.org/

- *Functionalities*: metrics monitoring and visualization, metrics aggregation, and metrics analysis (correlations, metrics suggestion, clustering, metrics prediction).
- *Monitoring*: software engineering related measures.
- *Data sources*: A set of measures[29] for Github, Modelio, Jenkins, Mantis, Openproject, SVN, SonarQube
- *Pricing*: Community
- *OSS Repository*: https://github.com/ITEA3-Measure/MeasurePlatform

4.2.4. Q-Rapids

The Q-Rapids[30] tool provides a data-driven approach to monitor, analyse quality in the context of rapid software development. It is produced by Q-Rapids project, a European Union's H2020 research & innovation funded project involving seven partners from five countries. This tool is aimed at being improved during the VISDOM project by satisfying different necessities defined by Experis IT. For more information check Section 5.2.

Summary:

- *Functionalities*: quality-related indicators monitoring and visualization, customised raw data visualizations, prediction, simulation (in the context of quality-related indicators and quality requirements), quality alerts, and quality requirements suggestion.
- *Monitoring*: custom product and process indicators. The current version includes source code analysis (e.g., complexity), code quality (e.g. sonarqube analysis), continous integration (e.g., tests), process performance (e.g., development velocity).
- *Data sources*: set of data connectors[31] for issues management systems (Jira, Mantis, Redmine), project management (OpenProject), source code (SVN, Gitlab, GitHub), continous integration tools (Jenkings). That can be extended adding new connectors as Apache Kafka connectors.
- *Pricing*: Community
- *OSS repository*: https://github.com/q-rapids/q-rapids

The analysis of the commercial approaches reveals that the main functionality of these tools is monitoring indicators related to concrete software development quality aspects, e.g., code quality, DevOps flow, risk, and security. All of them provide the user a graphical interface to visualise the evaluation of these indicators, i.e. interactive dashboard. This information can be complemented with some information that can help decision-makers to take concrete actions, e.g. list of open issues with priority. Some of them also provide complementary features, like prediction, simulation, and DevOps tools integration visualization. Some of the tools have an Open Source Software version (Bitergia and SonarQube), combining commercial and community-oriented models, and some offer a free option including some specific functionality (Datadog and Splunk).

The analysis of the community-oriented approaches reveals that these approaches provide similar functionality to the commercial approaches. The main difference is that the community-oriented approaches provides more customisation at level of data gathering, the data gathering and analysis is

[29] https://github.com/ITEA3-Measure/Measures
[30] https://www.q-rapids.eu/
[31] https://github.com/q-rapids/qrapids-connect

not designed to monitor concrete quality aspects. All of them are available in GitHub, a popular Open Source Software repository.

Finally, analysing the relevant tools presented in this section and mapping according to the categories of DevOps pipes (see Figure 32), we can observe that none of these tools offers the same coverage that VISDOM seeks.

	Speciality	Software	Process	Organization	Deployment	Operation	Usage
Bitergia			■	■			
Datadog	Performance		■		■	■	
Kiuwan	Security	■					
New Relic						■	■
SonarQube	Qality	■					
Splunk	Capacity					■	■
Tasktop	Flow approach		■	■	■		
Q-Rapids	Quality	■	■				■
Measure		■	■				

Figure 32:Analysed tools by DevOps pipeline categorization

5. VISDOM STATE OF PRACTICE

VISDOM partners companies either use or develop dashboards in their internal development and have identified the need for moving forward from simple and single-purpose radiators to advanced visualization and thus joined in this project.

This chapter describes the state of practice embodied in VISDOM, in other words, it presents each of the tools that VISDOM partners currently use or are developing to commercialize.

5.1. State of Practice at OCE

In this section we focus on the *visualisation of quality of cyber-physical systems in the operations phase* (i.e. the machines are used by real customers).
For such cyber-physical systems high level quality is usually reported at the system level. Software quality is then a sub aspect.

System level quality indications are obtained by monitoring the machines used by customers. The software of these machines logs the machine status and usage and in particular also errors that occur. Machine status can also be logged at a very fine-grained level (for example, the status of individual sensors).

These aspects are automatically transferred on a regular basis to the back office of the manufacturer where they are analysed and presented in the form of lists, graphs and dashboards for particular stakeholders (marketing, service, manufacturing, development, etc.). A typical example of such a dashboard, which focusses on printer usage, is shown below.

Figure 33 Dashboad focused on print usage

At this level errors are reported as MRE (machine recoverable error) and ORE (operator recoverable error). A consolidated measure is given by MPBE (mean prints between error) number. Error occurrences are typically reported in bar graphs and tables.

Figure 34: Example of an error occurrence graph

A small subset of these errors (recognizable from their codes) is caused by software (indicating for example software error or software failure). Detailed error reports are listed in tables, with rows for each error occurrence. Excel is often used to summarize them, as shown below.

21515	**10**		**16**	**5**	**6**	**8**	**8**	**53**
//system/printer/21_CTL	**10**		**16**	**5**	**6**	**8**	**8**	**53**
SOFTWARE_ERROR	10		16	5	6	8	8	53
21516	**10**	**6**	**11**		**12**		**4**	**43**
//system/printer/21_CTL	**10**	**6**	**11**		**12**		**4**	**43**
SOFTWARE_ERROR_DS	10	6	11		12		4	43

Figure 35: Example of an error occurrence table

5.2. State of Practice at EXPERIS / UPC

As mentioned in Section 4.2, Q-Rapids tool provides a data-driven approach to monitor, analyse quality in the context of rapid software development. It was produced by a European Union's H2020 research & innovation funded project.

Figure 36 shows some of the visualizations provided by the tool using several visualization styles (gauges, radar charts, textual, network).

Figure 36. QaSD visualizations (graphical & textual)

The Q-Rapids tool includes the following features, accessible through QaSD (Quality-aware Strategic Dashboard):

- Quality-related indicators assessment and visualization.
- Customised raw data visualizations.
- Prediction.
- Simulation (in the context of quality-related indicators and quality requirements).
- Quality alerts notification.
- Quality Requirements (QR) suggestion.

Quality-related assessment is based on a custom quality model, defining a 3-level model that should be customized by the adopter organization. This quality model characterises key strategic indicators (strategic indicators) as an aggregation of product and/or process factors, and these factors as an aggregation of concrete metrics. QaSD includes several views to visualise the quality model elements assessment allowing the navigation from strategic indicators to metrics (see Figure 34). The user can visualize current assessment (Figure 37, gauge and radar charts) and historical data (Figure 34, linear chart) in graphical and textual way.

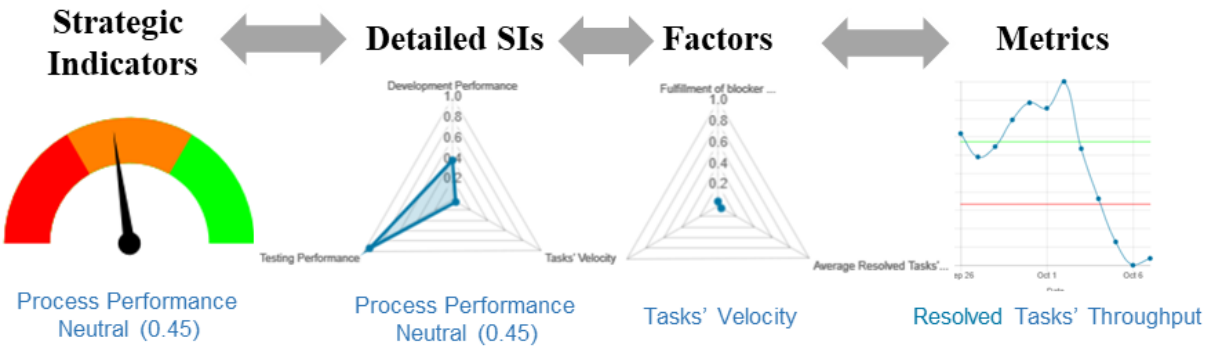

Figure 37. QaSD navigation schema

The Q-Rapids tool includes the possibility of creating *customised dashboards* (using Kibana) with actionable information, for example the critical issues to be solved (see Figure 38). These dashboards are accessible from QaSD.

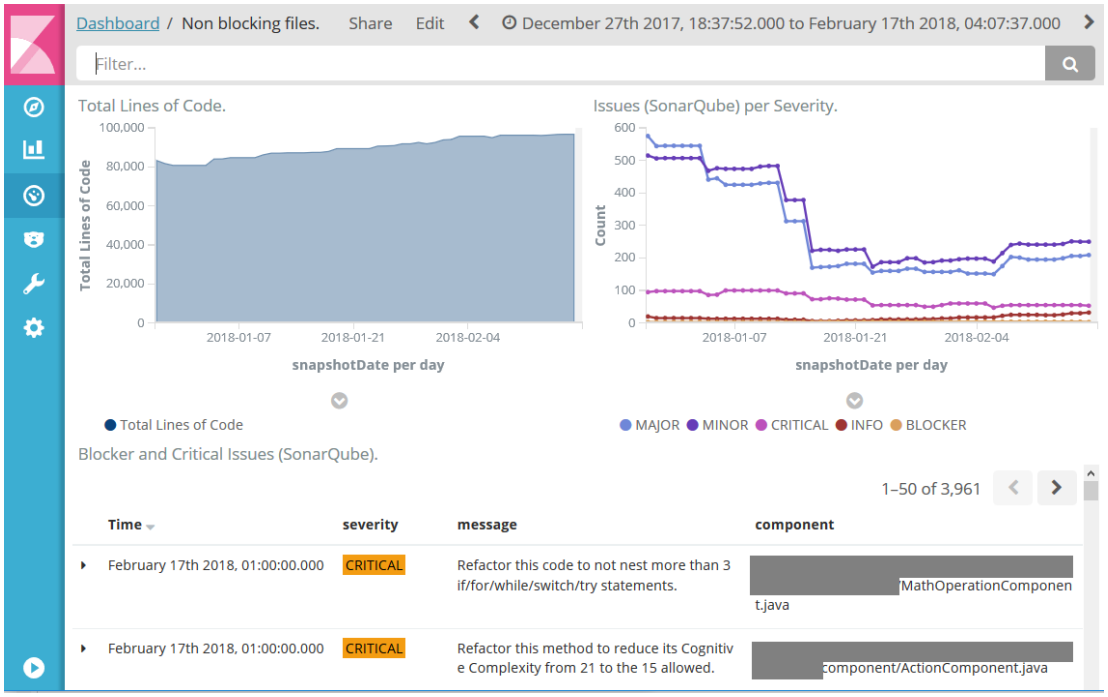

Figure 38. Q-Rapids Raw Data Dashboard

For *prediction*, QaSD integrates a set of forecasting techniques used to predict strategic indicators, factors and metrics (Figure 39).

Figure 39. QaSD prediction view

Simulation feature allows the decision-maker to set-up concrete values for metrics and factors and analyse how these values would affect to the strategic indicators (Figure 40).

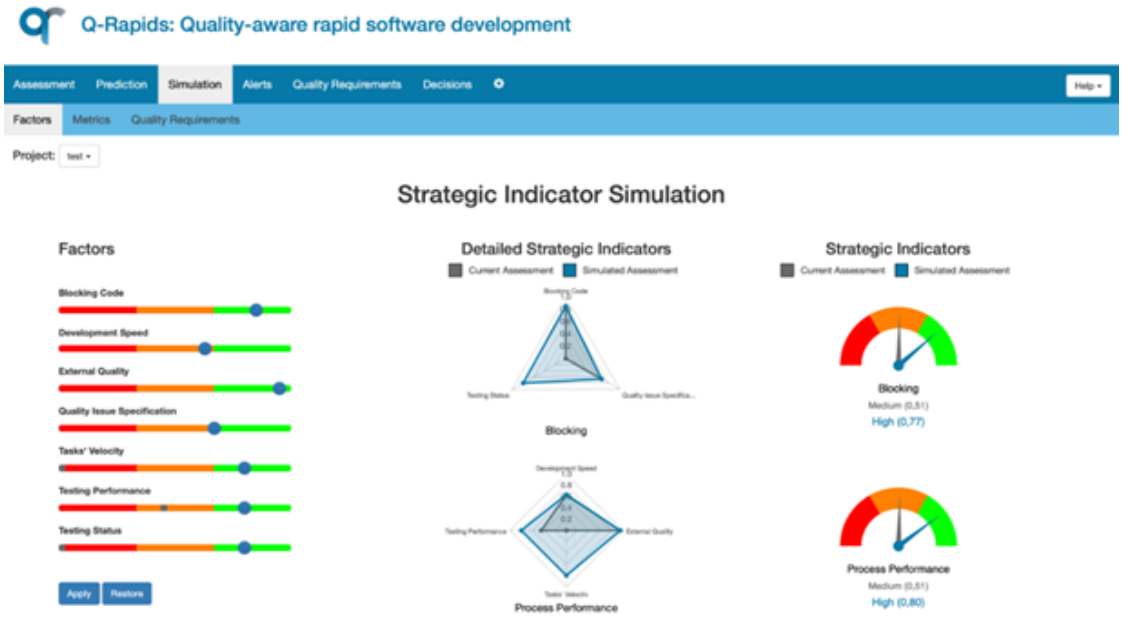

Figure 40. QaSD simulation view

Q-Rapids tool produces *quality-related alerts* when quality model elements have low values, Figure 41 shows how these alerts are visualised in QaSD (Figure 41, top). For addressing these alerts, QaSD *suggest QRs* (Figure 41**¡Error! No se encuentra el origen de la referencia.**, bottom. QaSD also provides the option to simulate the impact of adding the suggested QR on the strategic indicators (Figure 42**¡Error! No se encuentra el origen de la referencia.**).

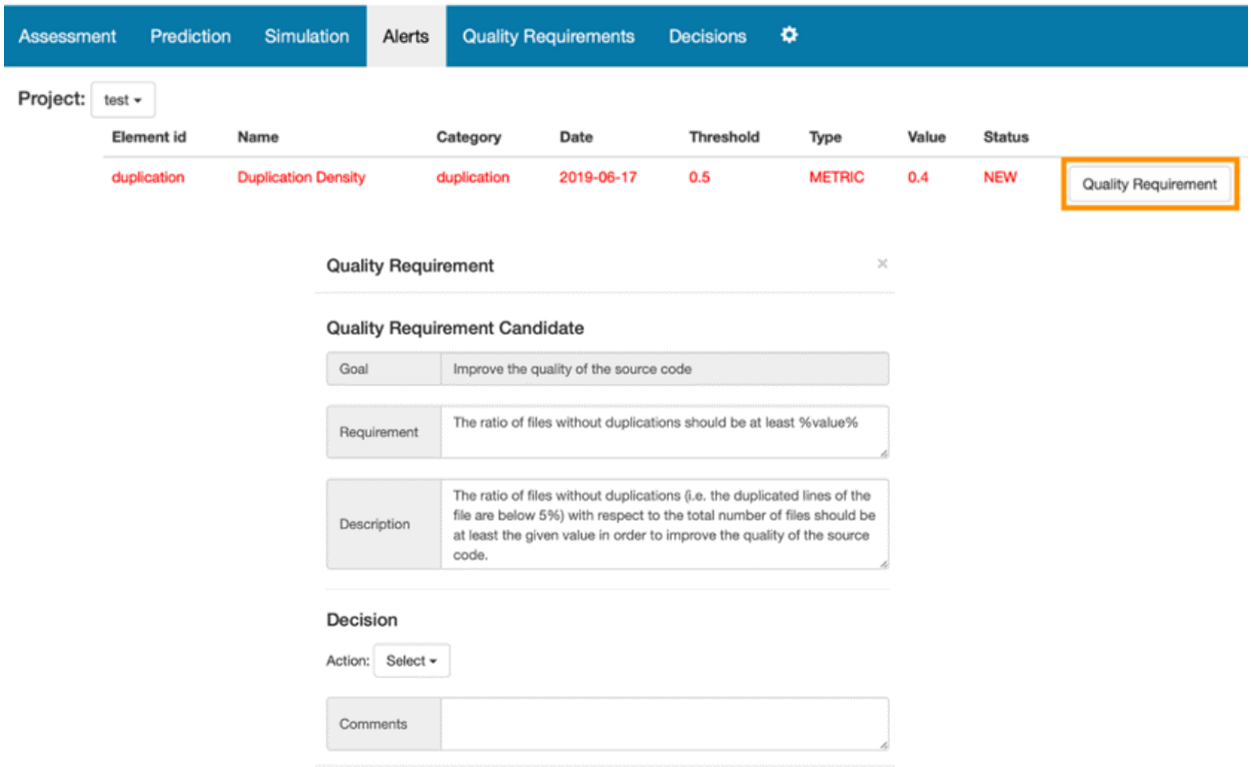

Figure 41. QaSDquality alert and QR suggestion

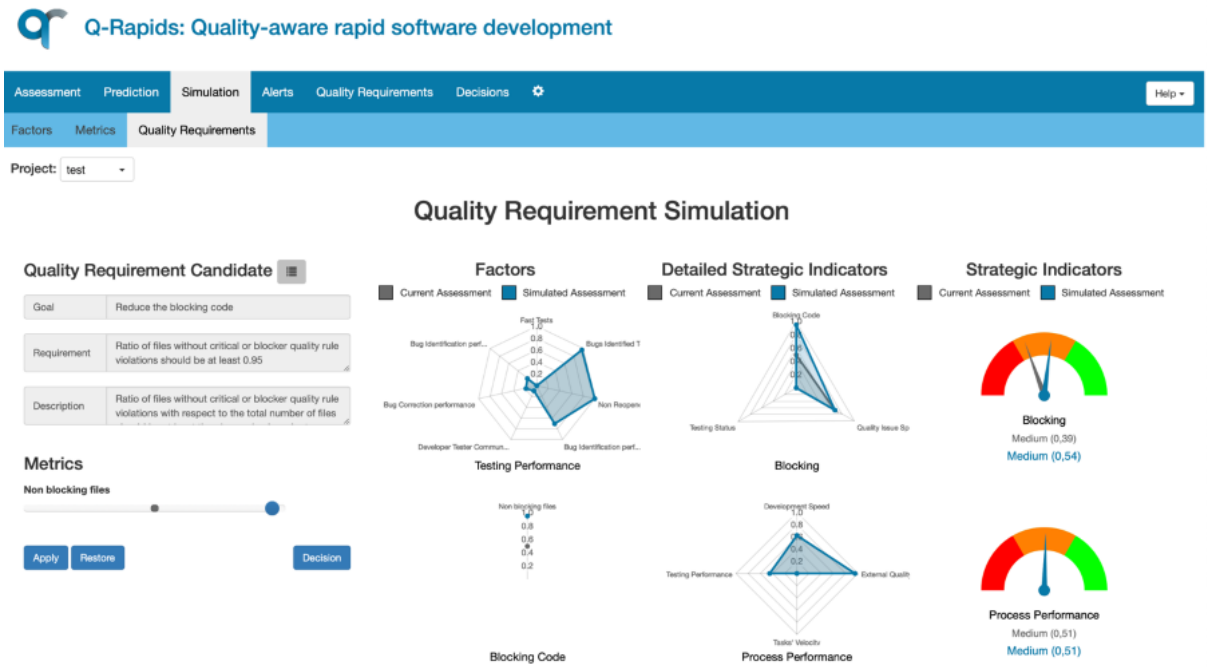

Figure 42. Q-Rapids QR simulation

5.3. State of Practice at QENTINEL

Qentinel has developed its own visualization tools based on the work done in Tampere university of Technology and Internal research. Some of the work was also done as part of the N4S-program by DIMECC in Finland.

The basis for the development of these tools has been to create a higher level dashboard, which gives you an overview to your whole software development process and integrates into the best of breed tools for different facets of the DevOps pipeline.

The data backend is based on the homogenized software engineering data model (Mattila et al., 2015) and enables us to collect data from all the sources. This data is then viewable at one source and data combining is possible. The software can be seen in Figure 43.

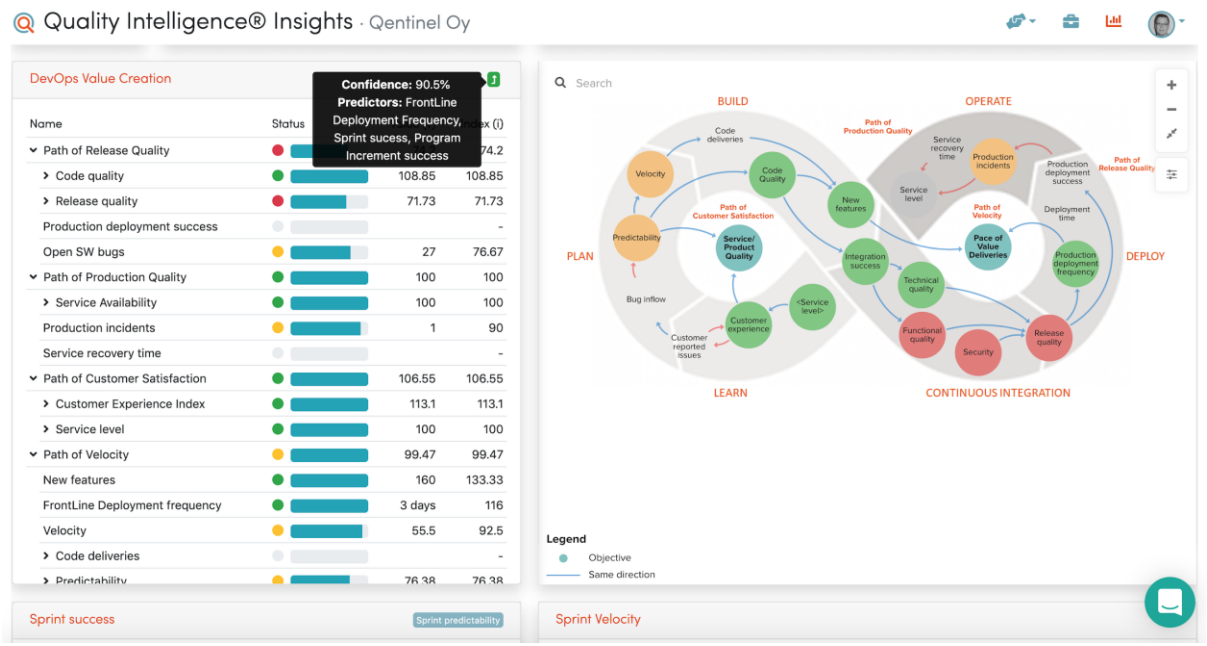

Figure 43: Qentinel Quality Intelligence for DevOps Dashboard

The dashboard combines a tree based grouping of different metrics for different parts of the software pipeline. These trees group similar metrics which relate to same values under one node in the DevOps causal loop diagram on the right. Every metric is indexed to the same index values, which is a value between 0 and 200. The indexes are then propagated through the tree models to the top level nodes. This will give you an index value for example. your code quality. The causal loop diagram on the right then displays inferred correlations between the different nodes and visualizes bottlenecks in development, which are visualized by color.

5.4. State of Practice at TAU

Research background

In the previous projects TAU has developed visualizations for various software projects, including those applying DevOps practices (Mattila et al., 2015a; Mattila et al., 2015b; Mattila et al. 2017). The visualizations have used data from issue tracking, version management, and usage. The common data model for these visualizations has been defined and data collectors for various tools have been implemented.

However, there are still several reasons for future research and development:

- Only a few industrial cases have been implemented with the tool so far. We need to collect more experience and evidence about the applicability of the system in real context.
- Most of the visualizations have been based on timelines into which we have mapped various types of information. Although, the timeline notation has proven to be useful, we want to explore new types of visualizations sketched is the FPP document.
- Our data models are currently specific to our visualizations and not yet tested with large number of data sources and visualization. In VISDOM, and especially in WP2, the data models will be developed for a wider set of data sources and use cases. In addition, we investigate opportunities to be more compatible with industry standards.

- The current architecture is similar to ETL process of BI system: the data is transferred and converted from the tools with a separate operation. Update of the information requires a manual operation by the user. In VISDOM project we will investigate more real-time approaches.

Visualization in teaching

In addition to normal office graphics to visualize numeric data we have used our teaching Gitlab to track and visualize student progress. For instance, we have used this data to see if students have started their projects in time, or if the work flow is split unevenly in group works. We have used both excel to produce our own visualizations and use the one provided by Gitlab as in the following:

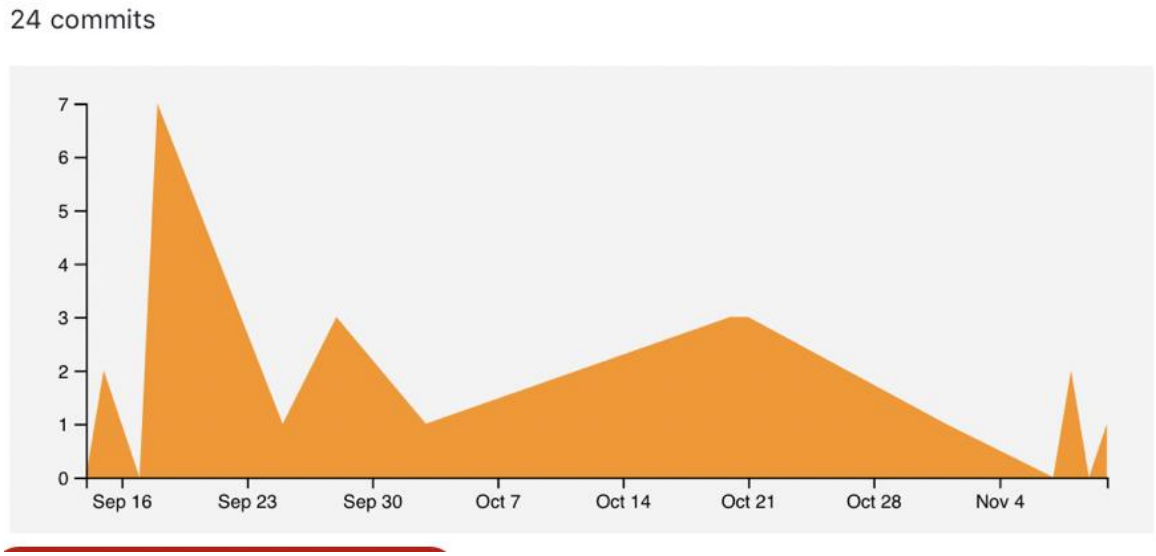

Figure 44: Visualizations for teaching produced by TAU.

In another case we have asked students to use Agilefant[32] for project management and from that produces the following types of visualizations:

[32] https://www.agilefant.com

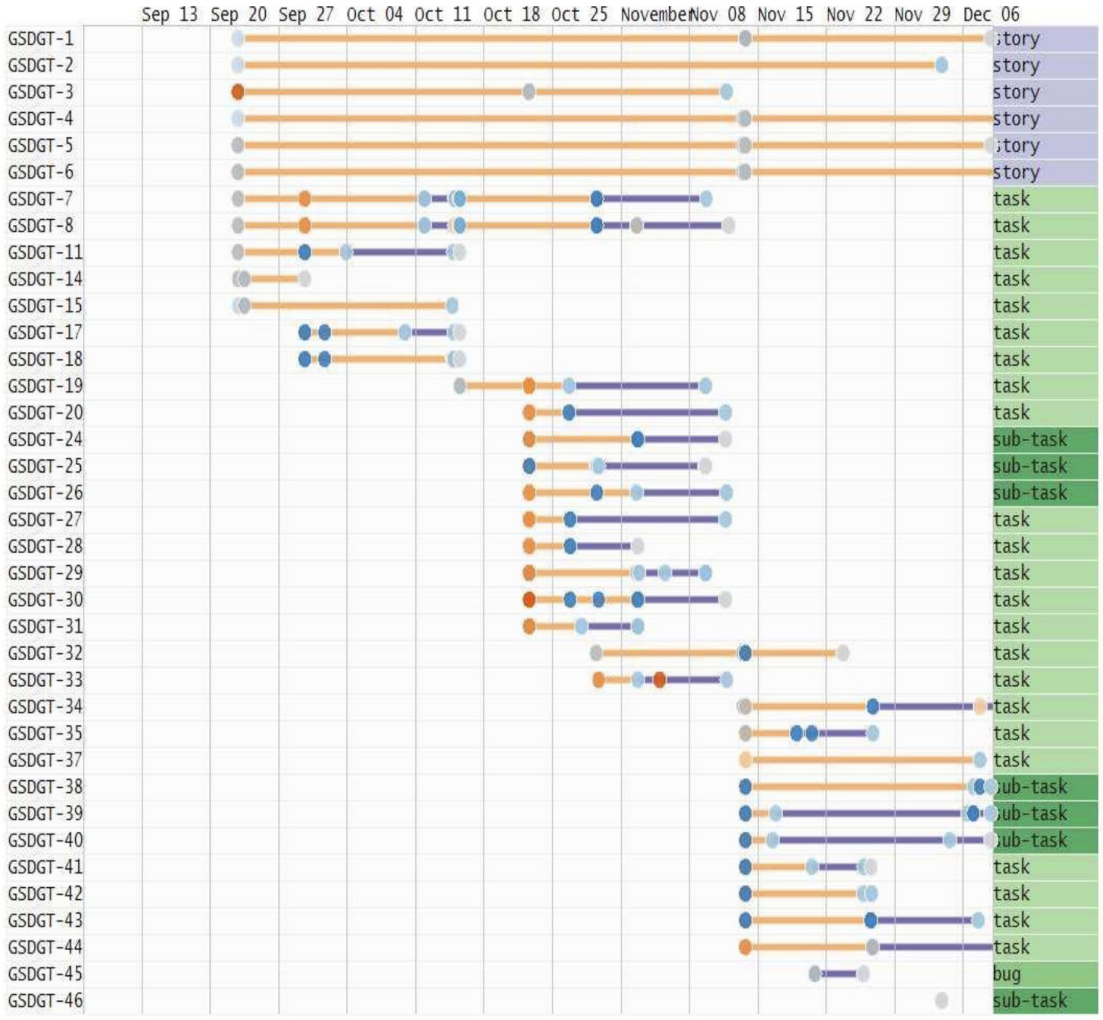

Figure 45: Visualizations produced by Agilefant

As can be seen from above we have just used some simple and separated visualizations. In the teaching use case of VISDOM we will develop more holistic dashboards.

5.5. State of Practice at INVENCO

Invenco's experiences regarding visualisation of software and data related processes originate mostly from three sources: 1) customer cases (projects or consultancy) 2) implemented visualisation features in own products and 3) visualisation of own development related processes. When it comes to visualisation of in-house development processes, there is a recent experience in fetching data e.g. from Trello into InControl platform database, and the first experimentations visualising this data are currently being carried out.

In addition, new experience has been acquired with DataOps-oriented visualisations a. More information regarding this experience and case is explained con be find in blog the following blog-posting (in Finnish), main idea being though, that BI-environments and their operations can be divided in four areas: 1) Monitoring, 2) Service operations, 3) Documentation automation and 4) Resource capacity planning. In order to support all these areas, the BI Monitoring Tool is being developed.

Figure 46: Example from the monitoring tool UI, showing e.g. some crucial SQL Server related information.

This tool has also reporting and drill-down capabilities, in order to get fast into bottom of the whatever went wrong with the BI environment.

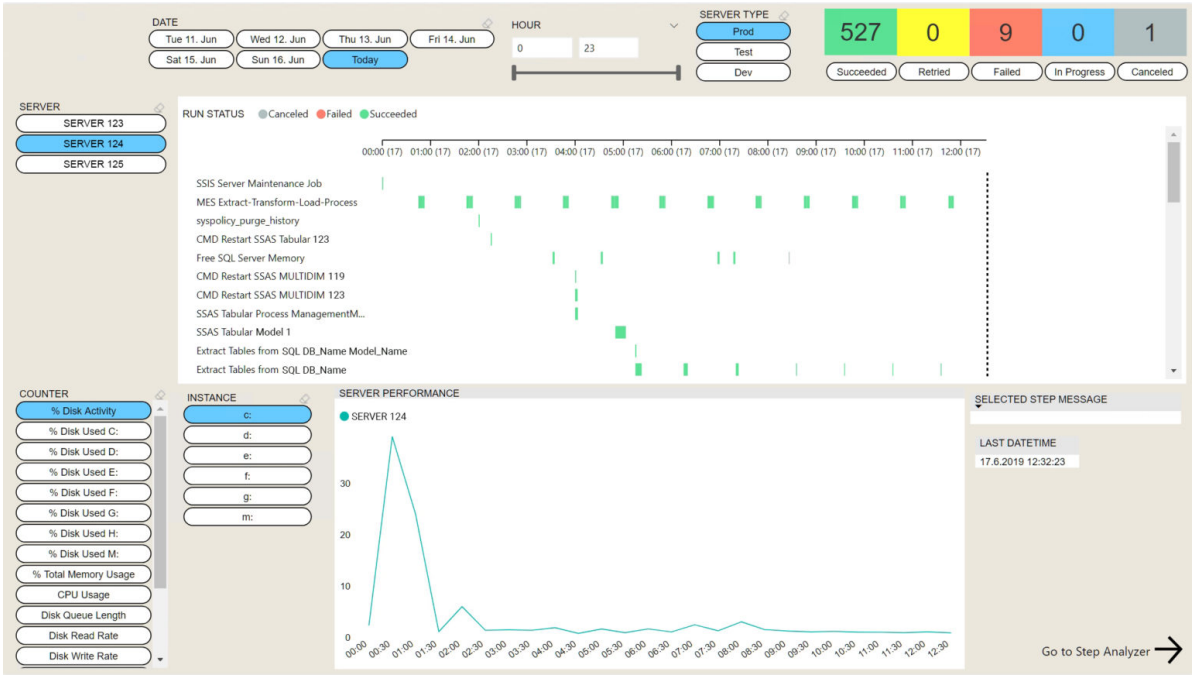

Figure 47: Example Power BI report from the certain BI environment status with drill-down capabilities

BI Monitoring Tool exemplifies the data intensive development teams' need for a similar tool support as more traditional development tool have provided for SW developers quite some time already. What these tools are still lacking is e.g. consolidated and user-friendly role-based visualisation for the process and environments together, and that is one of the themes Invenco is looking forward to focus our industrial research efforts in VISDOM.

5.6. State of Practice at VINCIT

In SaaS systems it is crucial to understand your users. You need to know their needs and how they interact with the system. Google Analytics or other commercial alternatives such as MixPanel can be used to create funnels that show how users interact with the feature. If the feature consists of multiple steps, funnel shows how many percent of the users make it to the next step. This is important for analysing, for example, conversion rate from free users to paying ones. Which are the obstacles in the way of purchase?

As a State of Practice in LaaS MixPanel is used to create simple funnels. For example, if a user orders a service from LaaS, he or she needs to select who is going to provide the service. In the figure below, there is a visualization of different funnels related to selecting service provider. There is not much data shown in the figure as screenshots are taken from the staging environment to protect the real user data.

Figure 48: Example of a LaaS MixPanel (a)

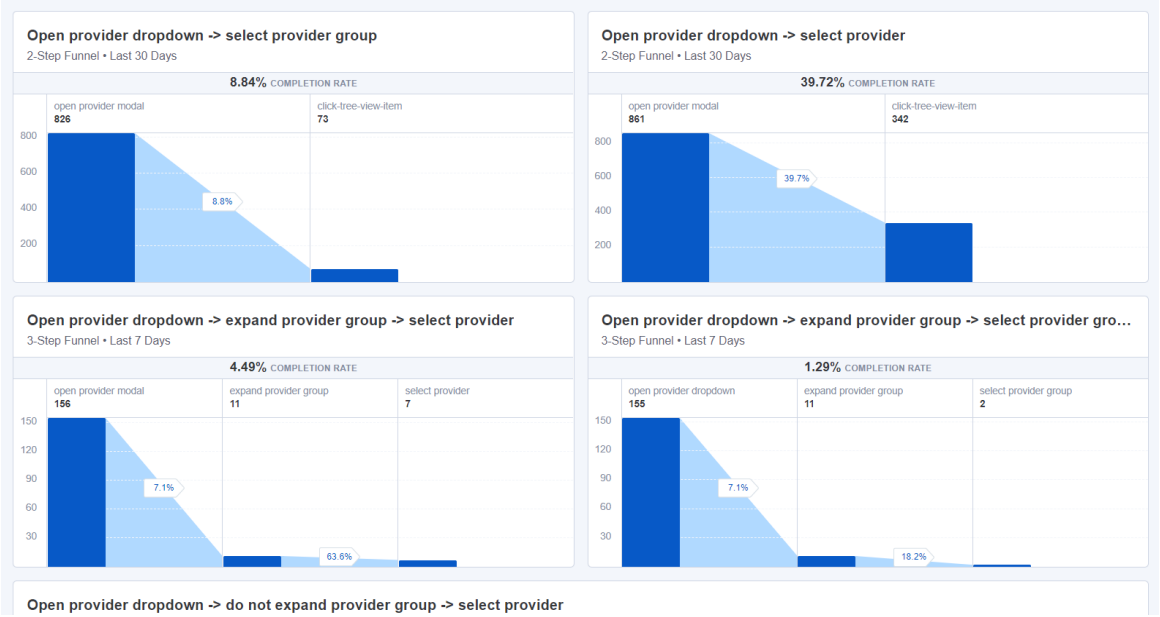

Figure 49 : Example of a LaaS MixPanel (b)

In addition, to follow the data about conversion in various funnels, the team monitors various data points: How many unique users have sent feedback within a day after ordering a leadership service from LaaS, how many unique moods have been tracked using companion application, which devices have been used to access different features in companion application. Current visualizations of this companion app data are shown in Figure 50.

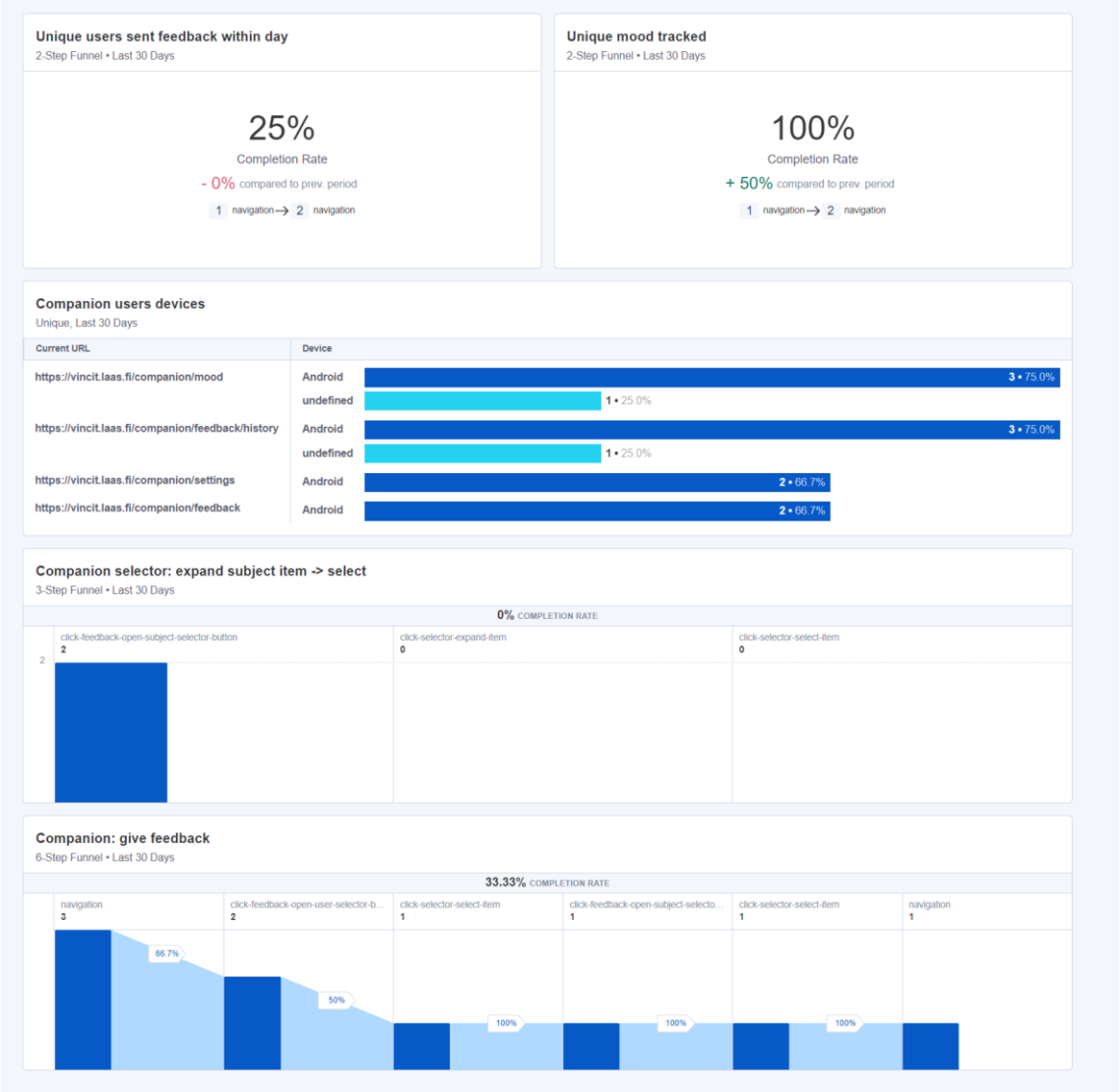

Figure 50:Visualization of LaaS companion app data

6. CONCLUSIONS

Throughout this document, we have analysed the state of the art of the main factors on which VISDOM wants to provide value by providing a solution that goes beyond what exists in the market, which are the visualization applied to the DevOps environment. Following this analysis, it can be concluded that most of the solutions that currently exist look at independent aspects, without being able to provide information on the impact of each of the development categories into which a DevOps development can be divided. No solution covers the range of analysis to which VISDOM strives. In addition, some of the tools studied do not provide any visualization mechanism, or if they do, they are usually of complex interpretation or oriented to only one type of user.

The state of the practice provided to VISDOM by the project partners was also assessed, revealing the powerful knowledge and background contained in this project.

Finally, this document not only shows us which are the main techniques and technologies linked to the project, but also configures a tool to understand which the novelties are proposed by VISDOM and its potential impact on the sector.

REFERENCES

Avgeriou, Paris, Neil A Ernst, Robert L Nord, and Philippe Kruchten. 2016. “Technical Debt: Broadening Perspectives Report on the Seventh Workshop on Managing Technical Debt (MTD 2015).” SIGSOFT Softw. Eng. Notes 41 (2): 38–41. https://doi.org/10.1145/2894784.2894800.

Bartusevics, A. (2017). Automation of Continuous Services: What Companies of Latvia Says about It?. Procedia Computer Science, 104, 81-88.

Besker, T, A Martini, and J Bosch. 2017. “The Pricey Bill of Technical Debt: When and by Whom Will It Be Paid?” In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 13–23. https://doi.org/10.1109/ICSME.2017.42.

Besker, Terese, Antonio Martini, and Jan Bosch. 2018. “Managing Architectural Technical Debt: A Unified Model and Systematic Literature Review.” Journal of Systems and Software 135: 1–16. https://doi.org/https://doi.org/10.1016/j.jss.2017.09.025.

Biase, M di, A Rastogi, M Bruntink, and A van Deursen. 2019. “The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes.” In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 113–22. https://doi.org/10.1109/TechDebt.2019.00030.

Bollin, A., Hochmüller, E. and Szabó, C. "Teaching Software Project Management by Simulation: Training Team Leaders for Real World Projects," 2015 IEEE 28th Conference on Software Engineering Education and Training, Florence, 2015, pp. 7-9.

Bollin, A., Hochmüller, E., and Mittermeir, R. T. "Teaching software project management using simulations," 2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T), Honolulu, HI, 2011, pp. 81-90.

Cai, Y, L Xiao, R Kazman, R Mo, and Q Feng. 2019. “Design Rule Spaces: A New Model for Representing and Analyzing Software Architecture.” IEEE Transactions on Software Engineering 45 (7): 657–82. https://doi.org/10.1109/TSE.2018.2797899.

Chidamber, S R, and C F Kemerer. 1994. “A Metrics Suite for Object Oriented Design.” IEEE Transactions on Software Engineering 20 (6): 476–93. https://doi.org/10.1109/32.295895.

Cito, J., Leitner, P., Fritz, T., & Gall, H. C. (2015, August). The making of cloud applications: An empirical study on software development for the cloud. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 393-403). ACM.

Collofello, J. S.. "University/industry collaboration in developing a simulation-based software project management training course," in IEEE Transactions on Education, vol. 43, no. 4, pp. 389-393, Nov. 2000.

d. S. Maldonado, E, E Shihab, and N Tsantalis. 2017. “Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt.” IEEE Transactions on Software Engineering 43 (11): 1044–62. https://doi.org/10.1109/TSE.2017.2654244.

de Souza, R. T., Zorzo, S. D., and da Silva, D. A. "Evaluating capstone project through flexible and collaborative use of Scrum framework," 2015 IEEE Frontiers in Education Conference (FIE), El Paso, TX, 2015, pp. 1-7.

Deblaere, F. , Demeulemeester, E. and Herroelen, W. (2011), RESCON: Educational project scheduling software. Comput. Appl. Eng. Educ., 19: 327-336. doi:10.1002/cae.20314

Eliasson, U, A Martini, R Kaufmann, and S Odeh. 2015. “Identifying and Visualizing Architectural Debt and Its Efficiency Interest in the Automotive Domain: A Case Study.” In 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), 33–40. https://doi.org/10.1109/MTD.2015.7332622.

Eposhi, A, W Oizumi, A Garcia, L Sousa, R Oliveira, and A Oliveira. 2019. “Removal of Design Problems through Refactorings: Are We Looking at the Right Symptoms?” In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 148–53. https://doi.org/10.1109/ICPC.2019.00032.

Ernst, Neil A, Stephany Bellomo, Ipek Ozkaya, Robert L Nord, and Ian Gorton. 2015. “Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt.” In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 50–60. ESEC/FSE 2015. New York, NY, USA: ACM. https://doi.org/10.1145/2786805.2786848.

Espinosa-Curiel I.E., Rodríguez-Jacobo J., Fernández-Zepeda J.A. (2010) Graphical Technique to Support the Teaching/Learning Process of Software Process Reference Models. In: Riel A., O’Connor R., Tichkiewitch S., Messnarz R. (eds) Systems, Software and Services Process Improvement. EuroSPI 2010. Communications in Computer and Information Science, vol 99. Springer, Berlin, Heidelberg

Fontana, F A, I Pigazzini, R Roveda, and M Zanoni. 2016. “Automatic Detection of Instability Architectural Smells.” In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 433–37. https://doi.org/10.1109/ICSME.2016.33.

Helminen, J. 2009. Jype -- An Education-Oriented Integrated Program Visualization, Visual Debugging, and Programming Exercise Tool for Python, M.Sc. thesis, Helsinki University of Technology.

Holvitie, J, and V Leppänen. 2013. “DebtFlag: Technical Debt Management with a Development Environment Integrated Tool.” In 2013 4th International Workshop on Managing Technical Debt (MTD), 20–27. https://doi.org/10.1109/MTD.2013.6608674.

Holvitie, J, V Leppänen, and S Hyrynsalmi. 2014. “Technical Debt and the Effect of Agile Software Development Practices on It - An Industry Practitioner Survey.” In 2014 Sixth International Workshop on Managing Technical Debt, 35–42. https://doi.org/10.1109/MTD.2014.8.

Igaki, H., Fukuyasu, N., Saiki, S., Matsumoto, S., and Kusumoto, S.. 2014. Quantitative assessment with using ticket driven development for teaching scrum framework. In Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014). ACM, New York, NY, USA, 372-381. DOI=http://dx.doi.org/10.1145/2591062.2591162

J. Letouzey and M. Ilkiewicz, "Managing Technical Debt with the SQALE Method," in IEEE Software, vol. 29, no. 6, pp. 44-51, Nov.-Dec. 2012. DOI: 10.1109/MS.2012.129.

Kratzke, N., & Quint, P. C. (2017). Understanding cloud-native applications after 10 years of cloud computing-A systematic mapping study. Journal of Systems and Software, 126, 1-16.

Laukkanen, E. (2017). Adoption problems of modern release engineering practices. Aalto University publication series DOCTORAL DISSERTATIONS, 220/2017,http://urn.fi/URN:ISBN:978-952-60-7714-7

Lerina, A, and L Nardi. 2019. “Investigating on the Impact of Software Clones on Technical Debt.” In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 108–12. https://doi.org/10.1109/TechDebt.2019.00029.

Li, Z, P Liang, and P Avgeriou. 2015. “Architectural Technical Debt Identification Based on Architecture Decisions and Change Scenarios.” In 2015 12th Working IEEE/IFIP Conference on Software Architecture, 65–74. https://doi.org/10.1109/WICSA.2015.19.

Liu, Z, Q Huang, X Xia, E Shihab, D Lo, and S Li. 2018. “SATD Detector: A Text-Mining-Based Self-Admitted Technical Debt Detection Tool.” In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), 9–12.

Lwakatare, L E. (2017) DevOps adoption and implementation in software development practice: concept, practices, benefits and challenges (doctoral dissertation). University of Oulu, Finland.

Mäkinen, S., Leppänen, M., Kilamo, T., Mattila, A. L., Laukkanen, E., Pagels, M., & Männistö, T. (2016). Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises. Information and Software Technology, 80, 175-194.

Maravić Cisar, S.; Pinter, R.; Radosav, .P., Cisar, P.: Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3. International Journal of Computers Communications & Control, [S.l.], v. 6, n. 4, p. 668-680, dec. 2011.

Martini, A, and J Bosch. 2015. “The Danger of Architectural Technical Debt: Contagious Debt and Vicious Circles.” In 2015 12th Working IEEE/IFIP Conference on Software Architecture, 1–10. https://doi.org/10.1109/WICSA.2015.31.

Martini, A, and J Bosch. 2016. “An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt.” In 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 31–40.

Matthies, B., Kowark, T., Richly, K., Uflacker, M. and Plattner, H. "How Surveys, Tutors and Software Help to Assess Scrum Adoption in a Classroom Software Engineering Project," 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, 2016, pp. 313-322.

Mattila, A-L, Ihantola, P, Kilamo, T, Luoto, A, Nurminen, M & Väätäjä, H 2016, Software visualization today - Systematic literature review. inAcademicMindtrek 2016 - Proceedings of the 20th International Academic Mindtrek Conference.ACM, pp. 262-271, Mindtrek Conference, 1/01/00. https://doi.org/10.1145/2994310.2994327

McCabe, T J. 1976. “A Complexity Measure.” IEEE Trans. Softw. Eng. 2 (4): 308–20. https://doi.org/10.1109/TSE.1976.233837.

Mo, R, Y Cai, R Kazman, L Xiao, and Q Feng. 2016. “Decoupling Level: A New Metric for Architectural Maintenance Complexity.” In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 499–510. https://doi.org/10.1145/2884781.2884825.

Mori, A, G Vale, M Viggiato, J Oliveira, E Figueiredo, E Cirilo, P Jamshidi, and C Kastner. 2018. “Evaluating Domain-Specific Metric Thresholds: An Empirical Study.” In 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), 41–50.

Oechsle, R., and Schmitt, T. Javavis: Automatic program visualization with object and sequence diagrams using the java debug interface(jdi). In Stephan Diehl, editor, Software Visualization, State-of-the-art survey, pages 176{190. Springer, 2002.

Otto Hylli, Anna-Liisa Mattila and Kari Systä, Collecting Issue Management Data for Analysis with a Unified Model and API Descriptions, 14th Symposium on Programming Languages and Software Tools,

October 9-10, 2015, Tampere, Finland. Published in CEUR Workshop Proceedings, Vol-1524, ISSN 1613-0073, pages 251–265

Penners R & Dyck A (2015) Release Engineering vs. DevOps-An Approach to Define Both Terms. Full-scale Software Engineering 49–54

Plösch, R, J Bräuer, M Saft, and C Körner. 2018. “Design Debt Prioritization: A Design Best Practice-Based Approach.” In 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), 95–104.

Potdar, A, and E Shihab. 2014. “An Exploratory Study on Self-Admitted Technical Debt.” In 2014 IEEE International Conference on Software Maintenance and Evolution, 91–100. https://doi.org/10.1109/ICSME.2014.31.

Puppet. 2017 State of DevOps Report. URL: https://puppet.com/resources/whitepaper/state-of-devops-report

R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics", 2012 34th International Conference on Software Engineering (ICSE), Zurich, 2012, pp. 987-996. DOI: 10.1109/ICSE.2012.6227122

Salas-Morera, L., Arauzo-Azofra, A., García-Hernández, L., Palomo-Romero, J. M., Hervás-Martínez, C.. PpcProject: An educational tool for software project management, Computers & Education, Volume 69, 2013,Pages 181-188,

Sauvola, T., Lwakatare, L. E., Karvonen, T., Kuvaja, P., Olsson, H. H., Bosch, J., & Oivo, M. (2015). Towards customer-centric software development: a multiple-case study. In Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on (pp. 9-17). IEEE.

Soares de Toledo, S, A Martini, A Przybyszewska, and D I K Sjøberg. 2019. “Architectural Technical Debt in Microservices: A Case Study in a Large Company.” In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 78–87. https://doi.org/10.1109/TechDebt.2019.00026.

Sorva, J., Karavirta, K., and Malmi, L... A Review of Generic Program Visualization Systems for Introductory Programming Education. Trans. Comput. Educ. 13, 4, Article 15 (November 2013), 64 pages. DOI=http://dx.doi.org/10.1145/2490822

Ståhl, D, A Martini, and T Mårtensson. 2019. “Big Bangs and Small Pops: On Critical Cyclomatic Complexity and Developer Integration Behavior.” In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 81–90. https://doi.org/10.1109/ICSE-SEIP.2019.00017.

Tahir, Amjed, and Stephen G MacDonell. 2012. “A Systematic Mapping Study on Dynamic Metrics and Software Quality.” In Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM), 326–35. ICSM ’12. Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/ICSM.2012.6405289.

Tornhill, A. 2018. “Prioritize Technical Debt in Large-Scale Systems Using CodeScene.” In 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), 59–60.

Vanhoucke, M., Vereecke, A., and Gemmel, P.. The project scheduling game (PSG): Simulating time/cost trade-offs in projects, Proj Manage J 36 (2005), 51–59.

Wiederhold, G., Mediators in the architecture of future information systems. Computer, 1992, 25(3), s. 38-49.

Wiederhold, G., Mediators, Concepts and Practice. Information Reuse and Integration in Academia and Industry, 2017, s. 1-27. Springer.

Woodward, C. J., Cain, A., Pace, S., Jones, A. and Kupper, J. F. "Helping students track learning progress using burn down charts," Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Bali, 2013, pp. 104-109.

Wu, W., Cao, Y., Chen, B., Su, Q., and Li, K. "AnyviewC: A Visual Practice Platform for Data Structures Course," 2009 WRI World Congress on Computer Science and Information Engineering, Los Angeles, CA, 2009, pp. 493-497.

Xiao, L, Y Cai, R Kazman, R Mo, and Q Feng. 2016. “Identifying and Quantifying Architectural Debt.” In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 488–98. https://doi.org/10.1145/2884781.2884822.

Zampetti, F, A Serebrenik, and M Di Penta. 2018. “Was Self-Admitted Technical Debt Removal a Real Removal? An In-Depth Perspective.” In 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), 526–36.

Zampetti, F, C Noiseux, G Antoniol, F Khomh, and M D Penta. 2017. “Recommending When Design Technical Debt Should Be Self-Admitted.” In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 216–26. https://doi.org/10.1109/ICSME.2017.44.

Zitzewitz, A von. 2019. “Mitigating Technical and Architectural Debt with Sonargraph.” In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 66–67. https://doi.org/10.1109/TechDebt.2019.00022.

APPENDIX A:

This appendix aims at providing an overview of all the different tools and mechanisms that will be used to collect all different data within this project.

Tool	Data	Format	Mechanism
Jenkins	TBD/depends on pipeline /	JSON, XML	REST API, wrappers e.g. JenkinsAPI
	data related to testing status, like unit test errors, line coverage per unit test and duration of unit test execution.		
JetBrains TeamCity	TBD/depends on pipeline	XML, JSON	REST API
CircleCI	TBD/depends on pipeline	JSON	REST API
Travis	TO BE CHECKED, NO ACCESS TO API SPEC	JSON	REST API
BuildBot	TBD/depends on pipeline	JSON	REST API

Collaboration

Tool	Data	Format	Mechanism
JIRA	TBD	XML, HTML, CSV	REST API, data export
	data related to issues like time of creation, estimated time and issue duration		
Redmine	Similar to JIRA		
Clubhouse	TBD	JSON	REST API
Trello	TBD	JSON	REST API

Version control

Tool	Data	Format	Mechanism
Git/GitHub	Commits etc	JSON	REST API
Git/BitBucket	Commits etc	JSON	REST API
Git/GitLab	Commits etc	JSON	REST API
SVN	Commits etc		
MANTIS	data concerning releases like total on-time releases per time frame or total features released		

CI/CD as a Service

Tool	Data	Format	Mechanism
AWS	To be checked		
Google Cloud Build			
Azure DevOps	See [Azure]	OData	OData REST API
GitLab CI/CD			