VISDOM
Visual software diagnostics

Evaluation and validation of the VISDOM framework in real use cases

Table of Contents

Revisions

Version	Date	Author	Notes
0.1	30.11.2021	Roelof Hamberg	First version
0.2	07.02.2022	Roelof Hamberg	Adapted structure, content additions
0.3	14.02.2022	Paris Avgeriou, Yikun Li	Added content on Technical Debt use case
0.4	05.04.2022	Roelof Hamberg	Further document structuring and intro
0.5	21.04.2022	Roelof Hamberg	Added content to starting points section
0.9	13.05.2022	Roelof Hamberg	Added chapters 4, 5, executive summary
1.0	27.06.2022	Roelof Hamberg	Review comments processed. Upload to ITEA.

Contributors

Roelof Hamberg	Canon Production Printing
Paris Avgeriou, Yikun Li	University of Groningen
Gema Maestro, Javier Gavilanes Ruano	Experis IT
Henri Terho, Esko Hannula	Copado (was Qentinel)
Esa Ronkainen	Invenco
Veli-Pekka Eloranta	Vincit
Kari Systä	Tampere University of Technology

Reviewers

Henri Bomström	University of Oulu

Executive Summary

The goal of the VISDOM project is to provide visual software diagnostics in the context of DevOps development environments. At its core is a framework containing a set of guidelines, a reference architecture, and several reference implementations to be able to develop new types of visualisations that utilise and merge data from several data sources in modern DevOps development. To cover this framework, three different aspects with innovative value were taken to define use cases and demonstrators: Quality of software, Software as a Service, Teaching.
In this document, the evaluations in practice and validations for purposefulness of these three use cases and the reference architecture elements and their evolution are discussed. This is done on a case-by-case basis for the use cases and in a generic way for the reference architecture. A large majority of the initial VISDOM framework elements have been confirmed and adapted where necessary, albeit that not for all use cases the solutions have been brought to an operational level.

1 Introduction

1.1 Purpose

The purpose of this document is to report on the activities done in the context of evaluating and validating the VISDOM use cases in the context of the VISDOM framework. Evaluation is the process of getting tangible ratings, either reflecting opinions or measured numbers, on the proposed solutions serving the use cases. Validation relates to finding out how well the solution serves its purpose. In this document these two terms are mostly used in close conjunction to each other.
The VISDOM framework is a set of guidelines, a reference architecture, and several reference implementations to be able to develop new types of visualisations that utilise and merge data from several data sources in modern DevOps development. The final goal is to provide simple "health check" visualisations on the status of the development process, software, and usage.
To cover the VISDOM framework, three different aspects with innovative value were matched with three different use cases: Quality of software, Software as a Service, Teaching.

1.2 Document structure

This document is organised as follows.
First, in chapter 2, some starting points are revisited. The state-of-the-art, the use case topics, and the VISDOM framework are summarised at a high level in order to provide the generic stage as a starting point for the evaluations and validations.
Next, chapter 3 contains the use case evaluations themselves. Each use case is covered in a similar way: starting with a generic description of the use case itself with its goals, this is followed by the evaluation and validation activities that were done in order to improve the fulfilment of its goals and concluded by a valuation of the activities with respect to the VISDOM KPIs.
In Chapter 4 the perspective of the general concepts of VISDOM is taken in order to discuss the findings from the use cases. The last Chapter 5 concludes this deliverable.

2 Background

2.1 State-of-the-art

The state-of-the-art in visual diagnosis for DevOps software development is elaborated in Deliverable D1.1.1 and the appendix to the VISDOM progress report of 2021 H1. Some highlights are given here in order to set the stage for the VISDOM work and its evaluation.

In relation to the three demonstration areas (the demonstrators in the VISDOM project are defined in deliverable D1.2.1), the following observations can be made.
First, in the *visualisation of quality aspects*, the notion of technical debt and its visualisation is a relatively new area that is in high demand from industry. Furthermore, contextualising the quality aspects in a complete round-trip DevOps process is a challenge that is not readily available by any product in the market.
Second, in the *international software-as-a-service business* the basic principles of working with customer journey analytics, Google analytics, and Azure application insights to get feedback on individual initiatives or product features are well known, but if one wants to plan ahead using

projected values and estimated development efforts, there are hardly any products that provide this functionality.
Third, *visualisation in teaching practices* is only reported in a few studies. Some approaches based on simulators with inclusion of gaming elements do use visualisations like Gantt charts to show progress but working with live or real data in these studies is currently lacking.

In relation to DevOps and its supporting tools, a lot of focus has been on automation of development tasks and the trends have been largely tool-centric. There is a good opportunity for dashboards/visualisations, as the standardisation of data and interoperability is gaining prominence with gradually increasing openness of these DevOps tools.

Lastly, the area of commercial tools in software analytics and visualisation is considered. This is a huge market with many providers, which mostly originate from a specialist focus area. Considered examples of such tools include those of bitergia, datadog, kiuwan, new relic, sonarqube, splunk, tasktop, cast, ndepend, teamscale, TiCS framework, and there are many more.
The applied visualisations are mainly based on straightforward techniques.

2.2 Use cases

The detailed definitions of the VISDOM visualisation use cases are documented in deliverable D1.2.1. They are threefold, focusing on different aspects related to software development as such: 1) quality aspects of the software to be developed, 2) business aspects of the software to be developed, and 3) training of developers. The quality aspects use case has been further split into 4 sub use cases, each covering different parts of the development process: quality assurance, value creation and quality, technical debt, and process management.
The use cases provide the structure of Chapter 3 in this deliverable which provides the main body of evaluations and validations that have been done in the context of the VISDOM project.

2.3 VISDOM reference architecture

The final technical architecture of VISDOM is detailed in deliverable D2.5.2. It starts from a number of high-level business requirements and scoping of the VISDOM contribution in the area of enabling visual diagnosis in DevOps software development. The essential concepts are summarized here.
The reference architecture is based on the observation that data sources are actually external to the VISDOM scope and heterogeneous in character. Therefore, the need for fetching data and recombining different sources into dependent data sets that are the basis for innovative visualisations, is central to the architecture. This is dealt with in the data management system, preferably in automated ways, sometimes manually, and dependent on its goals in a scheduled or on-demand manner. Designs for visualisation must be flexibly deployable on top of the data management system and should be part of configurable dashboards which are stakeholder-dependent. Micro front-ends enable these properties.
New designs for visualisations make use of cross-links with other, unrelated domains, such as healthcare (EKG, pulse, blood pressure, X-ray, see also deliverable D3.1.1).
The dashboard composer enables configuring different dashboards for different stakeholders, using as conceptual ingredients service information, stakeholder information, view information, view logic, layout logic, and visualisation control logic.

3 Evaluations and validations

In this chapter the VISDOM use cases are evaluated and validated. This has been done for each use case in an individual manner as the topics and/or actual technical context were too different to be combined in a single evaluation.
The discussion on each use case follows a standard structure.
First, a simple description of the use case is given in order to make the actual context and starting points clear. Second, a description of pilots, evaluations, and optionally improvements with re-evaluations are presented. Also, suggestions for future improvements are added in this part. Third and last, a valuation with respect to the 4 main VISDOM KPIs is given. These relate respectively to dashboard developments for different stakeholders, advanced visualisations, utilised software engineering tools to serve as data sources, and business impact in terms of improvements in development effort/cost or DORA[1] metrics.

3.1 Use case 1: Visualisation of quality aspects

This section is devoted to the visualisation of quality aspects of products being developed and operated in a DevOps setting. Four different sub-use cases have been identified in VISDOM and are described independently in the following four sub sections.

3.1.1 Use case 1.1: Quality Assurance

Description

The execution of this use case has been a collaboration between Experis and UPC. It consists of a dashboard to monitor the full life cycle of a software project, from the development to the operations phase. It gathers information from commonly used software tools, such as Jenkins or SonarQube, processes this information and displays it for the project stakeholders to get an overview of the project status.

Evaluation and validation

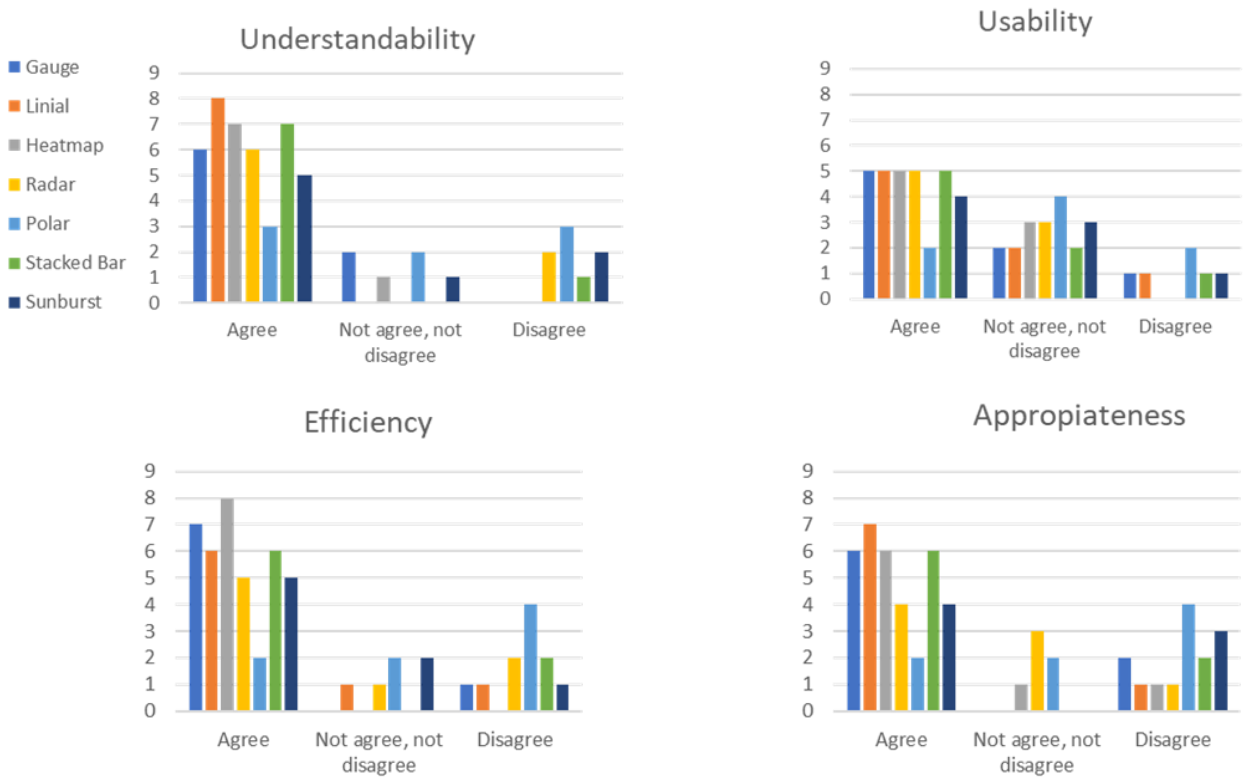
The dashboard has been evaluated in two different workshops with end users to evaluate its usefulness and identify possible improvements. The end users represented the various types of users that will utilise the platform: project director, project owner, system analyst, software developer and server technician. The first workshop took place at the end of 2020, and it was useful to gather information about potential improvements that were later implemented, and the users' evaluation was positive.

The second workshop took place at the end of 2021. In it, the last version of the platform including the previously mentioned improvements was evaluated, with positive results. This session was also used to give training for platform usage and metrics creation.

The evaluated charts were:

[1] See https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance.

As mentioned before, the metrics visualisations were positively evaluated. The following figure shows the results:



- Almost all chart types were well accepted
- Polar chart was the less evaluated chart for all evaluated qualities (light blue)

Highlights:

- Using heatmaps for showing evolution was well considered by the users, the best qualified quality is Efficiency (in terms of time and mental)
- Stacked charts were the best evaluated for Detailed Views (showing relation between levels)

Some feedback received was:

- “Gauge charts provide a quick view for the current status, it is highly visual”
- “Linear charts for expressing evolution is quite familiar”

- "Polar charts allow to see quickly which factors affect to the indicator, their assessment value, and weights"
- "I like the idea of navigating from high-level indicators to lower-level"
- "Sunburst is clear and dynamic"
- "A remarkable aspect is providing different charts to visualise the same data, each user can choose which is better for him/her" (detailed views: radar, polar, and stacked bar)

It was concluded that Gauge was not appropriate if there are too many indicators (e.g. Metrics), suggestions using linear-gauge and using grouped bar chart besides/instead of stacked bar chart (for detailed views).

During the first workshop, it was proposed to include a new type of visualisation for cases where multiple metrics had to be displayed, which was implemented as shown in the following image:

VISDOM KPIs evaluation

KPI 1: Dashboard development (number of configurable dashboards, stakeholders ≥ 3)

- Number of configurable dashboards: **4**
- Stakeholders: **5**

KPI 2: Advanced visualisations (e.g. EKG, Pulse, X-ray, Blood-pressure):
5 (radar, polar, linear, heatmap, gauge) for all metrics

KPI 3: Utilised SW engineering tools (tools used as data sources for visualisations):
5 (Jenkins, SonarQube, Github, Metricbeats, Jolokia)

KPI 4: Business impact (improvement project development effort, cost, DORA metrics): According to the participants in the evaluation workshops, the dashboard will increase efficiency in project monitoring. Estimations point out an increase from 20 to 40%.

3.1.2 Use case 1.2: Dashboard for supporting quality intelligence and value creation

Description

The execution of this use case has been a collaboration between Qentinel (now Copado Inc.) and the University of Oulu. With Qentinel's quality intelligence tools and robotic testing as a starting point the direction in this use case is to move towards the Quality Intelligence for DevOps product.

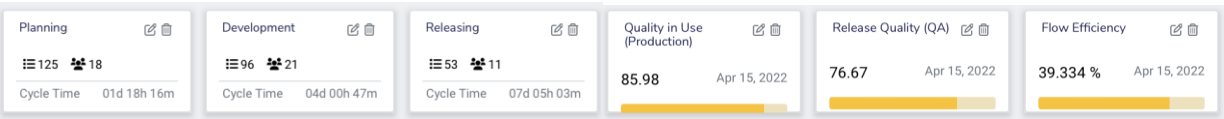
Evaluation and validation

The dashboard that Qentinel, now Copado Inc. has been working on has been deployed to a few customers and data on customer feedback on the architecture of the dashboard and its data ingestion features has been collected. One bigger client, Softbank Robotics, has been evaluating the dashboard and provided feedback on the level of data homogenization and its ingestion into the system. The data homogenization for the platform still remains the biggest skill-based bottleneck in taking these kinds of systems into use. This is caused by the fact that the homogenization formulas have to be created by the users of the platform to feed software engineering metadata, such as ticketing data, version history data and test result data into the system. The usefulness of the data has been totally dependent on who has been doing the data homogenization.

Another complicated step has been to configure the right metrics to be visible in the dashboard and understanding how the inferred causalities between the different metrics trees work in our dashboards. As the dashboard is based on metrics trees under different topics, such as technical debt, release quality and employee happiness, it was not always clear to users what the inferred causalities told them. We have now removed the inferred causalities from the dashboard to make it simpler and let the background visualisation provide the reference for causality.

Following pictures illustrate the different dashboard views.

Value Stream View is meant for executive management who typically don't have neither the time nor the need to dig into details but want a quick-glance-traffic-light-view that tells whether things are progressing in the right direction. The metrics shown in the view are highly customizable but high-level metrics that illustrate the flow performance have proven to be the best. The three boxes on the left illustrate work completed and work in progress, during a chosen time window, the rightmost three boxes illustrate typical flow metrics with traffic lights. The original design did not include the value stream view at all but, through customer interviews, we learned that just view is needed the complement the more flow-oriented visualisation.



A greater level of detail is provided by a hierarchical KPI tree and the unique value creation model that ties the KPIs to the different value drivers and illustrates the cause-and-effect relationships among them. This view is typically interesting to people who need to figure out forecasts, corrective action plans, or continuous improvement plans.

The KPI tree and the value creation model are customizable as the tool was originally designed to model and measure any kind of process. Experience has proven, however, that such customization requires both a lot of understanding and a lot of detailed work. Therefore, a pre-configured DevOps metrics tree and value creation model are provided out of the box. The user still has to configure the connections to the different data sources, of course.

During the project, we became aware of an important customer dilemma. On one hand, many user organisations seem to believe their process is somehow unique and its metrics and value creation model would have to be customised accordingly. On the other hand, there rarely is enough understanding or patience in the user organisation to actually do such customisation. These learnings guided us to design a general ready-to-run DevOps dashboard that can still be customised to accommodate the customer's specific needs. Special attention was paid to making the connections to external data sources easy to set up.

On the most detailed level, any KPI can be opened on a zoomable timeline to get its detailed history, as illustrated below.

One of the key lessons learnt in the project was that, in addition to different levels of detail, there has to be "personalised" data, too: users want to view their own individual data, their team, their product, the company, etc.

The results presented herein are based on observations with some 40 user organisations and 200+ individual users, including Qentinel itself. Only a handful of them made extensive use of the features while the vast majority could be considered "passive users". Test statistics was the most popular use case and, in general, willingness and capability to customize the dashboards was found to be surprisingly modest.

VISDOM KPIs evaluation

KPI 1: Dashboard development (number of configurable dashboards, stakeholders ≥ 3)
The exact number is not available as some customers have configured dashboards themselves but, including internal use, the total number of dashboards and stakeholders is at least 20.

KPI 2: Advanced visualisations (e.g. EKG, Pulse, X-ray, Blood-pressure)
Visualizations are mostly traditional as illustrated above. The value creation model visualisation on top of the DevOps infinity loop is truly unique and new. The KPI tree with coloured status bars is unique, too. A third example of advanced visualisations is a two-dimensional coloured heatmap that can be used to correlate two factors, such as team and velocity, for example.

KPI 3: Utilized SW engineering tools (tools used as data sources for visualisations)
An open, extensible connector architecture for obtaining data from different sources. Both push and pull models are supported. Pull services were created for Gitlab, Github, Azure DevOps, Jira, and Tiobe. The data volumes and data update frequencies of different tools vary a lot. A key finding in the project was that although a daily data update is adequate for displaying trends and statistics, the users actually want to have everything near real time. Because they know there is more recent data in some tool they get annoyed if it is not immediately visible in the dashboards, too.

KPI 4: Business impact (improvement project development effort, cost, DORA metrics)
The market is still clearly divided - in two dimensions, in fact. Large software organisations with multiple teams and project/product portfolios tend to recognize the need for consistent metrics-based management of their software value streams. Less mature organisations, as well as mature but small organisations where everyone "feels the pulse" of the process directly are less willing to adopt this kind of solution.

3.1.3 Use case 1.3: Visualising Technical Debt

Description

The execution of this use case has been a collaboration between the University of Groningen, Canon Production Printing (CPP) and Vincit. The use case pertains to visually managing technical debt in a DevOps setting.

Technical debt (TD) refers to compromising the long-term maintainability and evolvability of software systems by selecting sub-optimal solutions, in order to achieve short-term goals [1]. When software developers incur technical debt, they sometimes explicitly admit it; for example, software developers may write “TODO” or “Fixme” in a source code comment, indicating a sub-optimal solution in that part of the code. Potdar and Shihab [2] called these textual statements Self-Admitted Technical Debt (SATD). SATD can be found in several sources such as source code comments [2], issues in issue tracking systems [3], [4] and commit messages [5]. The visualisation that was built by the University of Groningen in the context of the VISDOM project is the first one that allows the identification of SATD from a combination of the aforementioned three sources. This has clear advantages over the state of practice, as it allows for a more complete picture to be formed regarding the overall technical debt of a system: the three sources used are complementary and provide different angles on the maintainability and evolvability of software systems.

Evaluation and validation

The University of Groningen and CPP conducted an exploratory case study to investigate how SATD is managed and how this can be supported within CPP. We collected data in two steps. First, we identified and characterised SATD in projects within CPP from these sources: issues, source code comments and commits. This step took place by using pre-trained machine learning models [6]. Second, we carried out a series of interviews with 12 software practitioners from CPP to understand their perception of what SATD really is, how it is managed, and how this management can be potentially improved. We note that none of the previous studies in literature has surveyed software developers about SATD, in order to capture their perspectives towards SATD management, and tooling support for different sources. This is another novelty of our work, as we are the first to take into account the input of software engineers regarding the merit of the visualisation and potential improvements to it.
More formally, the goal of this study, formulated according to the Goal-Question-Metric [7] template is to “analyse self-admitted technical debt in source code comments, issue tracking systems, and commit messages for the purpose of understanding and improvement with respect to the nature and management process of self-admitted technical debt in practice from the point of view of software engineers in the context of the embedded systems industry.”. To address this goal, we first chose a large-scale CPP project which contains eight sub-projects. More specifically, the selected project has over 475k lines of comments, 21k commits, and 130k files (including documentation, test files, configuration files etc.). Regarding issues, we collected 78k issues from the issue tracking system. Subsequently we conducted semi-structured interviews with 12 participants from CPP: architects, software developers, team leads and project managers.

In collaboration with Vincit, the technical debt tool was evaluated in the context of the Roadmapper tool. We run a study on how to integrate the tool to mark items on the roadmap as technical debt so that the stakeholders can balance the development of the new features versus paying back the technical debt. The analyzer tool works for the purpose and could be integrated in the data fetching of the Roadmapper tool. So, when fetching the items to Roadmap, we could automatically mark items that are technical debt instead of new features. Currently, what is lacking is the bridge from the data that the analyzer tool produces to the data model Roadmapper uses.

The source code is available as open source here: www.github.com/yikun-li/visdom-satd-management-system
A video showcasing the visualisations is available here: www.youtube.com/watch?v=QXH6Bj0HQew

Pilot evaluations
The main findings of the evaluation are summarised as follows:

- The visualisation is able to effectively identify and characterise SATD in industrial projects. Specifically, the results indicate that most technical debt is admitted in issues, followed by source code comments and commit messages. Non-SATD issues take a significantly shorter time to close, compared to SATD issues.
- The software engineers have acknowledged the identified SATD and the accompanying statistics. However, they do need more information to assess the importance of individual SATD items.
- The visualisation has given us the information to be able to establish relations between SATD from different sources. For example, we found that SATD in code comments and issues is referenced in the other sources, while SATD in commits is not referenced in other sources. This can be used in future improvements of the tool, by making such traces automatically identified and visible.
- The visualisation in combination with the interviews has provided triggers on SATD introduction and repayment. Specifically, the results show that developers have different reasons to introduce and pay back SATD, depending on the data source (code comments, issues, commits). This could be used to further improve the visualisation.
- We have established practices within CPP that are used to assist in SATD prioritisation and repayment. These practices are relatively straightforward and can be supported by future versions of the visualisation.

VISDOM KPIs evaluation

KPI 1: Dashboard development (number of configurable dashboards, stakeholders ≥ 3)
One dashboard has been developed that is able to visualise Technical Debt with 5 different visualizations by mining source code comments, issue tracking systems and commit messages. This dashboard, as confirmed by the validation, can be used by at least 4 stakeholders: architects, software developers, team leads and project managers.

KPI 2: Advanced visualisations (e.g. EKG, Pulse, X-ray, Blood-pressure)
The visualisations provide include: the number of Technical Debt items per source (source code comments, issues, commits), the types of technical debt (e.g. architecture/design debt, code debt), the evolution of technical debt over time, the source files data for each Technical Debt item, as well as a heatmap of Technical Debt over the system components.

KPI 3: Utilised SW engineering tools (tools used as data sources for visualisations)
We mine data from Git (in terms of source code comments and commits) as well as issue trackers (specifically CPP uses Microsoft TFS).

KPI 4: Business impact (improvement project development effort, cost, DORA metrics)

The software practitioners within CPP have acknowledged the merit of the visualisation in providing an overview of the Technical Debt within their system, as well as a number of insightful metrics regarding its evolution and repayment. It is expected that this visualisation will effectively reduce the maintenance and evolution costs and help the development teams better prioritise which Technical Debt items to repay in both the short-term and long-term.

3.1.4 Use case 1.4: Trello process management proof of concept

Description

The execution of this use case has been a collaboration between Invenco and the University of Oulu. Invenco InControl SW demonstration is based on different source items to the multiple stakeholders with configurable dashboard. You can see the big picture of the product business (net sales/profit), release information, software and production issues and even market indication in one dashboard. Naturally, you can build different views based on user groups.

Product Information
You can manage the product information from an administration view. In the demo, product-related financial information has been integrated from Visma Severa tool and Excel. The product owner can see the monthly view of how the product net sales/profit has been developed.

DevOps process
Trello is a one source item of our demo, and we have visualized data from it in order to show issues from selected release. Releases consist of items which has been selected from Roadmapper tool.

Production phase
Production phase bugs information has been gathered from Trello. Different stakeholders can see the monthly view of release related bugs.

Market weak signal indicator
This feature can bring the overall market status depending on the online video selection. It can be different to every product.

Evaluation and validation

We have had a few internal evaluation sessions, and we use this Invenco's monthly product development meetings.

We have planned to show this demo to selected customers with different domain area.

VISDOM KPIs evaluation

KPI 1: Dashboard development (number of configurable dashboards, stakeholders ≥ 3)

- Number of configurable dashboards: **5**
- Stakeholders: **3**

KPI 2: Advanced visualisations (e.g. EKG, Pulse, X-ray, Blood-pressure):
4 (Financial Information, issues, market weak signal indicator, bugs in production phase)

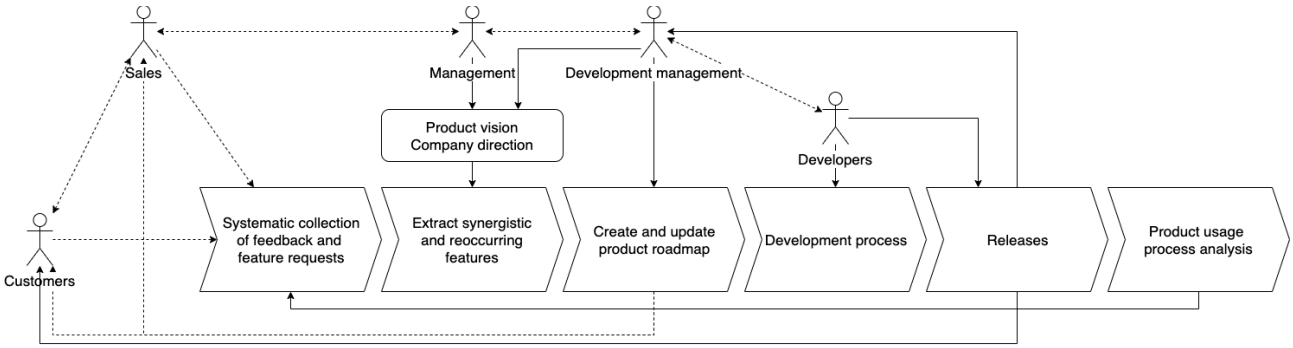
KPI 3: Utilised SW engineering tools (tools used as data sources for visualisations)
4 (Visma Severa, Trello, Roadmapper, Excel).

KPI 4: Business impact (improvement project development effort, cost, DORA metrics)
Less mature organization this gives a big picture to guide software development process with financial information.

3.2 Use case 2: Entering international SaaS business

Description

The execution of this use case has been a collaboration between Vincit, the University of Oulu, and the Technical University of Tampere. Vincit roadmap visualisation demonstration is based on requirements collected and analysed from visualisation needs in Vincit's software development process. The specific software development process affected is described in figure 1.



Based on the interviews of different stakeholders in the development process, specific needs were identified and described below.

Customer feedback visualisation

Development direction should be communicated to customers and users but also analyse how well the customers' feedback ends into product backlogs or the number of features based on customer feedback is in the next release(s). A goal of the customer feedback or satisfaction visualisation is to determine whether the customer is listened to, and if customers can be communicated about the fact whether or not their feedback results in action.

Development process overview

Generally a big picture view was seen as necessary, but it was also recognized that it is hard to achieve. Essentially the bigger problems arise when there is a need to see and evaluate whether certain things are pending on something (for example if some item is pending a review) and what is required to be done in order for the process to progress. Another issue is how to determine what features are planned for each release.

Systematic feature planning
Synergies between requests was identified as an important activity during feature planning as the clearest need was to identify recurring requests and features but also similarities between different requests from multiple sources, i.e., multiple customers. In addition, finding synergies with features already existing in the product or grouping existing features and requests with new requests requires supporting visualisations or tooling. Several sources were identified to contain information that supports feature planning (requiring connections to these systems or some other mechanism to obtain the information in a meaningful format), such as sales talks, phone calls, emails, or feedback from different channels. At least such information should be collected in one place to be available.

Feature roadmap
A view of the development process is needed to see what is coming, what is going on, when things get done, what has been done. The view is needed to see releases in longer term in order to have a schedule prediction or assist in determining what kind of features are done when and what they need to contain (for example in case of dependencies). Workload estimation alone is not always useful with feature evaluation and prioritisation, realised production capacity could be a point of further study for feature estimation.

Development process visibility
Visibility of the development process (especially features or roadmap) would increase the effectiveness of different stakeholders working within the process:

- Sales knows if they can sell more, what has been requested multiple times, when something has been promised and what is coming already.
- Development knows what is promised and when.
- Potential customers or their representatives, i.e., salespersons, could see how the promised feature is progressing and when it might be completed.

Based on these identified problem areas and Vincit's needs, a tool that supports creation and visualisation of roadmapping activities should be developed. The tool has been developed in cooperation with University of Oulu, University of Tampere and Vincit as continuous research and development activity.

Source code accessible to everyone, see https://github.com/visdom-project/visdom
A video showcasing the Roadmapper tool is available here:
https://www.youtube.com/watch?v=bkbOGrplGbI

Evaluation and validation
To evaluate and validate the tool we run action research where we conducted three iterations. For each iteration we conducted a focus group study where we invited people who acted as product owner, sales representatives (or account directors) and developers. In these focus groups, people used the tool to plan a roadmap for Roadmapper. Participants were given different goals for the meeting and researchers observed how they used the tool. Based on the feedback and observations from each session, we improved the Roadmapper tool. In the next sessions, we tested the improved functionality again and tested new features that were introduced in between

the sessions. Preliminary results show that the tool helps stakeholders to facilitate the discussions in the roadmap planning. The final results of the evaluation and validation will be published in a separate scientific publication.

VISDOM KPIs evaluation

KPI 1: Dashboard development (number of configurable dashboards, stakeholders ≥ 3)

- Number of configurable dashboards: **4**
- Stakeholders: **3**

KPI 2: Advanced visualisations (e.g. EKG, Pulse, X-ray, Blood-pressure):
7 (Heatmap, Time estimation, Roadmap plan, Value creation per milestone vs target, Roadmap progress tracking, task dependencies, task synergies)

KPI 3: Utilised SW engineering tools (tools used as data sources for visualisations):
3 (JIRA, Trello, Gitlab).
KPI 4: Business impact (improvement project development effort, cost, DORA metrics):
According to the participants in the evaluation workshops, the tool creates discussion between the stakeholders and is useful in roadmap planning.

3.3 Use case 3: Teaching use case

Description

The execution of this use case has been a collaboration between the Technical University of Tampere and the University of Oulu. Within the teaching use case we have implemented a dashboard to help teachers to follow the course implementation and students taking the course. We validated the dashboard with the course “Programming 2”, where we monitored two implementations: the first implementation lasted the whole Autumn 2021 semester (August-December) and had 341 students, the second implementation was more condensed for the last half of the Autumn (October-December) and had 51 students. The course was mostly on-line and run under Covid-19 restrictions.

The visualisations, dashboards and experiment were based on our earlier publication:

> H. Bomström, O. Sievi-Korte, T. Kilamo, K. Systä, E. Annanperä, M. Kelanti, K. Liukkunen, *Towards Stakeholder Specific Visualization of Learning Paths in Software Engineering Teaching*, Proceedings of the 49th Annual SEFI Conference. SEFI.

Here we analysed 14 (9 in Tampere and 5 in Oulu) teacher interviews, and results indicated that teachers would be interested in following students and discovering students that should receive teachers’ attention. Our dashboard was designed so that it allowed for monitoring the progress of the whole student body taking the course, so that one could see if there were individuals deviating from the average. Further, we had implemented a way to compare the progress of a student to that of students who had taken the course previously, allowing for light-weight predictive analytics on the prospective grades of students on the current course. With our dashboard we wanted to validate the finding of the interview study, i.e., how teachers would utilise the information provided in the visualisations.

The experiment was implemented with three major steps:

1. We implemented a dashboard that uses the data from the course's tool infrastructure. While the basis of the implementation was based on earlier interviews, the dashboard was augmented with features drawn from planning workshop(s) with the course staff.
2. To ensure appropriate handling of the student data and privacy, we documented the research plan and asked permission from the faculty. This plan included anonymization of the data and asking permission from students. Only data from students who granted permission were included.
3. VISDOM project members participated in the course's weekly staff meetings, where they operated the dashboard and observed how the staff interpreted the visualisations.

The implementation of the dashboard was also connected to our research in WP2 and WP3. The implementation is the one of first proof of concepts of the VISDOM reference architecture (see deliverable D2.5.1). Especially, we prototyped the data backend solutions of the VISDOM reference architecture. The implementation is based on two already published open source components:

- Source code of the configurable dashboard proof-of-concept: https://github.com/visdom-project/VISDOM-PoC-Composer
- Source code of the visualisation components: https://github.com/visdom-project/VISDOM-micro-frontends

Implementation of the teaching use case will also be published as open source.

Evaluation and validation

The findings contradict a bit with the findings of the interview paper. The teachers were more focused on the overall course and its tasks (exercises) than the individual students. The teachers were mainly interested if some part of the course was too difficult (students were skipping an exercise entirely or failing it) but showed less interest in the progress of individual students or what kind of grade distribution could be expected. We are not yet sure about the reason for this difference, but we assume that stress and high workload on running the course with so many students are a big factor. It may also be that the student/teacher ratio was too high for being interested in individual students. The course was also offered again during the next spring, so students have several opportunities in retaking the course and failure will not instantly be an obstacle in their studies.

Although interpretation of the visualisation required some learning in the beginning, the teachers learned to read them pretty quickly. The teachers were also able to give improvement ideas.

VISDOM KPIs evaluation

KPI 1: Dashboard development (number of configurable dashboards, stakeholders ≥ 3)

Two dashboards are/have been developed.

1. In Tampere University we developed a dashboard and visualisations for course Programming 2. The dashboard has been used in workshops with the course staff. The main purpose of the dashboard was to get feedback about visualisations and especially how visualisation could help the teachers.

The dashboard includes four different views.
The dashboard uses the first prototype of the VISDOM reference architecture. The data backend (fetchers, storage) was used in the real pilot and worked as assumed. The configurable dashboard was used in the initial prototype, but due to implementation immaturity of the proof-of-concept it was not used in the pilot.

2. In University of Oulu we have developed a dashboard and visualisation for programming 3 and requirements engineering course. The purpose of the dashboard for requirements engineering was to visualise for teacher and student what was the status of the coursework. In programming 3, the visualisations and dashboard support teachers by showing the course progress and individual progress. The requirements engineering dashboard uses Taiga as the main source of data where the dashboard the source is GitLab repository and pipeline.

KPI 2: Advanced visualisations (e.g. EKG, Pulse, X-ray, Blood-pressure)

In the Tampere case two visualisations were mainly used:

- Submission mode in status view showed the overall situation and allowed the teaching staff to recognize students with potential problems.
- For the EKG, the course used a custom configuration that allowed more detailed investigation of the progress of individual students.

The visualisations worked mostly as assumed, but a small number of improvement ideas were discovered.

KPI 3: Utilised SW engineering tools (tools used as data sources for visualisations)

A+ learning environment. The used data included:

- Metadata information about the course instance from the course, module and exercise documents
- Submissions to the exercises by the students
- Student-specific result data about the number of submissions made and the number of points received from each exercise

GitLab version management tools. Each student in the course had their own GitLab repository. The used data included:

- Repository files and folders pushed by the students to their repositories
- Commits made by the students

GitLab CI/CD tools. Each assignment has a CI/CD pipeline that builds, tests, and (sometimes) runs the produced build. The build data included:

- Unit test reports
- Build step reports
- Runtime testing reports for built binaries

KPI 4: Business impact (improvement project development effort, cost, DORA metrics)

Findings from the studies:

- The dashboard and visualisation worked as assumed but since the teaching staff was rather busy they did not have many resources for helping students lacking behind, the overall impact was not as promising as the assumed based on reviews before the pilot.

- Our original idea was that the dashboard could be integrated into our teaching infrastructure for the purpose of tracking students. The findings did not exclude this, but experiences indicated that the dashboard could be more useful in course and curriculum planning. With help of this tool, we can ensure stable difficulty level of the courses and efficient production of new professionals.

4 Evaluation of VISDOM general concepts

The evaluation of the VISDOM general concepts is based on the reference architecture as described in deliverable D2.5.2. The evaluation consists of a discussion on how the various implementations follow the reference architecture. The conceptual elements of the reference architecture itself provide the framework for this evaluation, whereas also the evolution of insights on the architecture as well as the business requirements play a relevant role.

Data sources

From the start, the heterogeneity of the data sources that are at the basis of visualizations has been identified as an important business requirement. In the use cases and demonstrators this has become very apparent as well. Furthermore, because of the context of DevOps development, it is clear that in many, if not all cases amongst the data sources is a software delivery platform that typically supports build, test, deploy, and run phases of software development encompassing version control, continuous integration, continuous deployment. While these latter tools have generally dashboarding capabilities on basis of their private data on board (which is an extensive set), they also provide interfaces to retrieve data to external tools. During the project these interfaces have evolved towards more openness and richer data sets, which still fits the original reference architecture setup.

The requirement of dynamical addition of data sources has been relaxed in the course of the project, as the full integration of new data sources up to visualizations and dashboards requires adaptation effort which does not justify implementing a full runtime or easily configurable data addition functionality. In the TAU programming 2 study (teaching case) the concepts of the data backend were iteratively added over time. The degree to which the dynamical addition of data sources requirement is met is driven by economics: especially for the tool vendors in VISDOM, their implementation architecture allows for additions with minimal effort.

Data fetchers

The data fetchers are responsible for retrieving the data from the data sources into the visualization system. In all use cases and demonstrators, the data fetching is the first functional step in the system, the implementation of which ranges between manual actions to retrieve the data, designed queries which are executed in a scheduled or on-demand manner, and getting data pushed from the data sources at regular or irregular time intervals. Simple re-combinations of original data into new, dependent data elements have also been observed regularly.

Typically, it is our observation that in nearly all cases all data required for the different “zooming” levels in visual analysis later on, comes along in single comprehensive data fetching actions. This results often from the requirement that the load on the data source system, caused by the data exports, needs to be distributed in a controllable way.

Where possible, data fetching is done using (de facto industry) data (interface) standards such as ODATA. Nevertheless, it is not limited to that; the visualization goal and the required data set to fulfil that goal are more leading in the design of the data fetching than the standards alone. Enrichment of the meta data set during fetching the data also has been observed regularly, and the architectural guideline on this aspect was therefore confirmed.

Data adapters

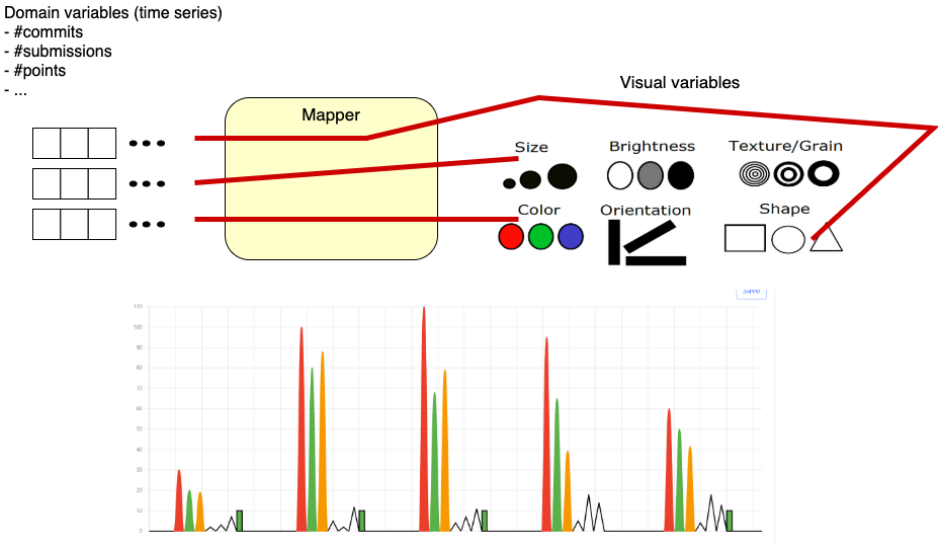
The data adapters are responsible for bridging the gap between the available data from a set of data sources and the data needed for visualization purposes. They were intended to be operating at the semantic level of data, i.e., irrespective of syntactic differences that exist between different data sources. This concept has been validated in the teaching case (TAU programming 2 study), where the visualization has been shown to be portable between Oulu and Tampere implementations, regardless of the fact that these two universities use partially different tools for running the courses.
In the reference architecture the part of the data adapters has evolved towards a close marriage with the “data model” that they provide towards the visualizations to be built on top. The design of such data models is essential to serve flexible visualization possibilities that are sufficiently responsive for the task they serve. This element of the reference architecture is now in line with ETL-like concepts in data pipelines. This data model is present in all the use cases and demonstrators, either in a volatile, cached, or stored form.

Analyzers

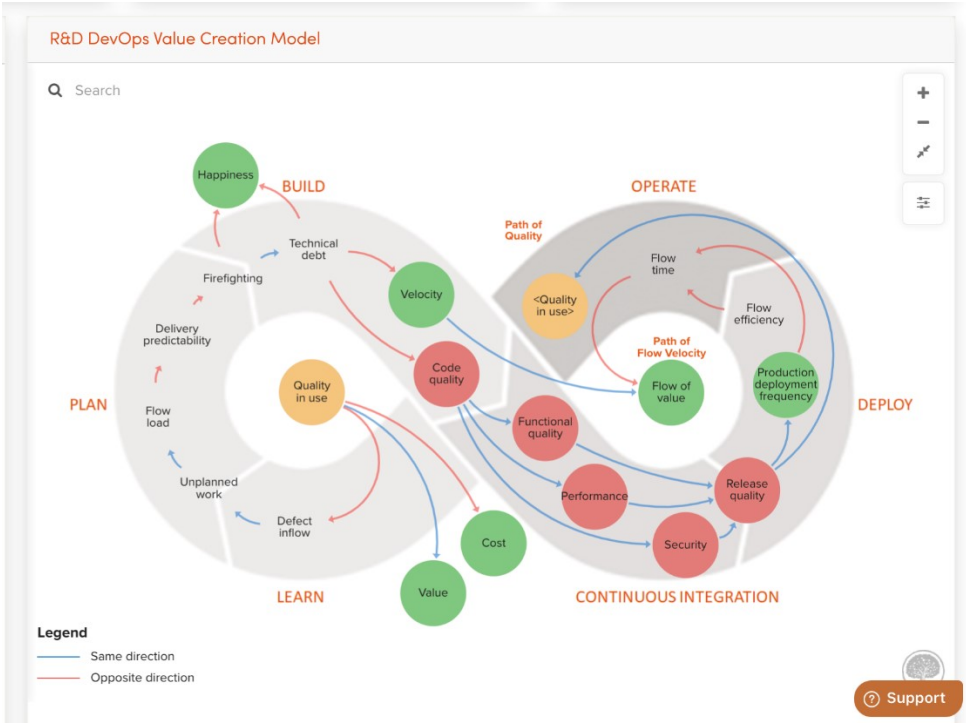
The reference architecture provides the opportunity to do more extensive data analysis by external tools on (a subset of) the data in order to arrive at a new dependent data set. The analysis may consist of model-based transformations, predictions of trained models, etc. We have not seen examples of this in operational cases, although the flexibility is required for research into new types of data adapters to enhance existing visualizations and/or dashboards.

Visualizations

The flexibility for new visualization designs, possibly inspired by cross-links with other, unrelated domains such as healthcare, has proven to be very useful although the number of really new visualizations is limited. As an example, the Programming 2 study included an EKG (see figure on the next page) and pulse visualisation. The concept of micro front ends was validated in several contexts, at least at the conceptual level. It is also compliant with dashboard solutions such as provided by Microsoft’s PowerBI.



Visualizations are more successful when they are recognizable, fit-for-purpose, and match the stakeholder expectations. In that respect, the regular use of heatmaps in the different use cases and demonstrators shows that this visualization method is very appropriate to serve the need of giving an overview that points the user into a direction where to zoom in. A second nice example is provided by the application of the DevOps infinity loop visual as a background for providing overview (see the figure below), which provides clear recognizability for stakeholders dealing with the entire process.



Dashboards

Dashboards are part of VISDOM's reference architecture to provide the flexibility towards different stakeholders and roles. This element is confirmed by a number of implementations in use cases, although in prototype settings this can take the form of configuration items of a single

dashboard or switching between web pages. For example, in the teaching use case Oulu and Tampere have experimented with the concept and technology prototype of the configurable dashboard, but it was not, however, implemented for Programming 2 experiment.
The information elements to be stored in the dashboard configuration (service information, stakeholder information, view information, view logic, layout logic, visualization control logic) make sense. Typically, the actual implementations use a smaller or larger subset of these information elements.

Security and Privacy

As the dashboards are intended to provide holistic overviews of product development and operational processes, special attention for security and privacy is required.
Security relates to the fact that not every stakeholder can see all data, and that unauthorized access to the data is blocked. At the edge of the system this is generally dealt with by a user/access token federation system that interacts with the system as a whole. In the data management system data security is regularly visible in the setup of the data model as resulting from the data adapters.
The need for anonymization service in the data management sub-system was identified, and also implemented in some of the use cases and demonstrators. On the other hand, the need for being able to view one's own data, or one's own project team data, is also still present. The exact balance between the partially conflicting requirements is being dealt with on a case-by-case basis.

5 Conclusions

The activities to evaluate and validate the VISDOM use cases in the context of the VISDOM framework were reported in this document. All use cases covering the three different aspects of DevOps development, product planning, and operation have been discussed in terms of the evaluations done, suggested improvements, and rating with respect to the VISDOM KPI's. Furthermore, the elements of the VISDOM reference architecture were evaluated for their applicability and evolutionary steps.
A large majority of the initial VISDOM framework elements have been confirmed and adapted where necessary, albeit that not all plans have been executed to the full extent. The COVID period in the middle of the project was one of the main inhibitors causing this. Nevertheless, the activities in VISDOM have led to promising new ideas, the best proof of this being delivered by the business growth of the tool vendor partners in the project, Qentinel (now Copado) and Tiobe.

References

[1] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162),” Dagstuhl Reports, vol. 6, no. 4, pp. 110–138, 2016.
[2] A. Potdar and E. Shihab, “An exploratory study on self-admitted technical debt,” in 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, 2014, pp. 91–100.
[3] K. Dai and P. Kruchten, “Detecting technical debt through issue trackers.” in QuASoQ@ APSEC, 2017, pp. 59–65.
[4] Y. Li, M. Soliman, and P. Avgeriou, “Identification and Remediation of Self-Admitted Technical Debt in Issue Trackers,” Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, pp. 495–503, 2020.
[5] F. Zampetti, G. Fucci, A. Serebrenik, and M. Di Penta, “Selfadmitted technical debt practices: a comparison between industry and open-source,” Empirical Software Engineering, vol. 26, no. 6, pp.
1–32, 2021.
[6] Y. Li, M. Soliman, and P. Avgeriou, “Automatic identification of self-admitted technical debt from different sources,” 2022.
[7] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, “Goal Question Metric (GQM) approach,” in Encyclopedia of Software Eng. Hoboken, NJ, USA: John Wiley & Sons, Inc., jan 2002, pp. 528–532.