VISDOM reference architecture

Deliverable 2.5.1

Version history

Version	Date	Author	Notes
0.1	12.02.2020	Kari Systä (TAU)	Initial draft based on discussions with Mika Koivuluoma, Markus Kelanti, and Henri Bomström.
0.2	24.02.2020	Kari Systä (TAU)	Responses to comments from Outi Sievi-Korte
0.3	09.03.2020	Kari Systä (TAU)	Solved all comments from Outi, added a few lines about security, privacy and access control
0.4	04.05.2020	Kari Systä (TAU)	More about dashboard composition
0.5	10.05.2020	Kari Systä (TAU)	Added more text to intro to explain the purpose and scope.
0.6	12.05.2020	Outi Sievi-Korte (TAU)	Added text on micro-frontends.
0.7	18.05.2020	Kari Systä	Small bug fixes
0.8	19.05.2020	Kari Systä	Notes from the plenary/workshop
0.9	25.05.2020	Kari Systä	Further notes from plenary/workshop
0.91	08.09.2020	Kari Systä	Towards deliverable 2.5.1
0.92	10.09.2020	Henri Bomström	Dashboard architecture description.
0.93	13.09.2020	Vivian Lunnikivi	Several small fixes and comments
0.94	13.09.2020	Kari Systä	Responses to comments
0.95	14.09.2020	Henri Bomström	Multiple fixes based on comments.
0.96	15.09.2020	Henri Bomström	Revisited sections 2.2, 2.4, and 3.
0.97	16.09.2020	Henri Bomström	Revisited section 2 and fixed inconsistencies.
1.00	21.09.2020	Kari Systä	Checked latest changed, moved version to 1.0,
1.01	30.09.2020	Kari Systä	Responses to review comments

1 Introduction

This document describes the top-level reference architecture of VISDOM. The purpose of this document is to

- provide a reference architecture for the research and technical work,
- recognize exploitable or standardizable components and interfaces, and
- act as a basis for research prototypes.

It should be noted that this technical reference architecture is assumed not to be implemented as such in each partner specific implementation.

The description is rather abstract in that many details are to be solved during the project. The idea is to describe the requirements and high-level concepts. The work will continue with analysis, more detailed designs and pilots. It is assumed that a new version of this document will be written during the last year of the project.

1.1 Business requirements

The proposed and developed architecture should support the overall goals of the VISDOM project. In particular, the following requirements should be supported by the architecture:

- The architecture shall support stakeholder and company specific dashboards that are composed of new visualizations developed in VISDOM project.
- The visualizations are usually based on several data sources (e.g. software engineering tools) and hence provide synthesized and abstracted views. The mainstream software engineering tools include visualizations about their data, one the main innovations of VISDOM is to combine data from several tools.
- The architecture shall support zooming-in/investigating the problems. This means that the users can investigate the original reason behind the interesting detail discovered from the visualization. This means that the "zooming in" needs access to all data used in the visualizations.
- The architecture shall support a large and extendible set of data sources (SW engineering tools). This means that it should be possible to dynamically add new tools as data sources.
- The architecture shall support all activities in DevOps development. This means that stakeholders interested in business, management, development, technology, deployment, hosting and operation should be supported. Furthermore, tools and data sources related to all these aspects can be used.
- The architecture shall support research and development in the VISDOM project. This means that we can test and demonstrate integrations with external and internal components in an agile way.
- The architecture shall support use of industry and de-facto standards.
- The architecture shall support exploitation of the results. This means that developed ideas and components can exploited in several context individually or together.

1.2 Architecture as described in the full project proposal

The technology value chain, as specified in the VISDOM Full Project Proposal (FPP), is depicted in the Figure 1:

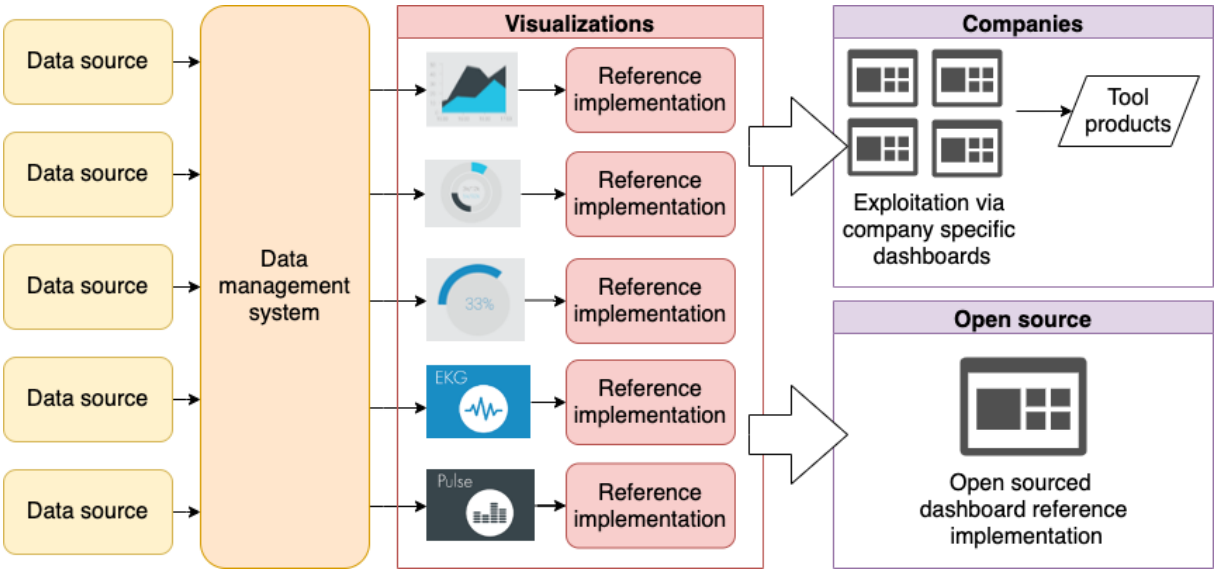

Figure 1 Technology value chain of the VISDOM project

The components in Figure 1, re-drawn from the FPP Figure 4, are the following:

- **Data sources** are the different development tools, databases and repositories used in software engineering. In VISDOM we develop methods and tools (open source by default) for collecting data from a variety of existing sources. We will also create a framework and instructions for developers who want to integrate new data sources.
  Architecture note: the data sources are external to VISDOM and we are dependent on the interface and data formats supported by the tools.
- The **data management system** includes ways to unify, merge, link and store the input data for effective visualizations and possible further processing. The data models and analysis methods will be realized as running software and included in the prototype implementation, making it possible to use the resulting metrics even without visualization.
  Architecture note: the data management system is the most challenging part of the architecture and thus it is the main topic of this document.
- The project produces a set of evaluated and piloted **designs of visualizations**. The designs include implementation guides, guidelines for users and requirements for the data management solution.
  Architecture notes: the designs can be accompanied with reusable reference implementations but can also support independent implementations on the same data.
- For the visualizations, we also develop **reference implementations** that can be used as components in various dashboards.

The project creates a configurable **dashboard** concept that can display different content and views to different stakeholders. We will also create prototype dashboards for the pilots, and the companies may exploit these dashboards by integrating them in their current and future development support tools. A dashboard is not necessarily just for direct presentation of data, but it can also include diagnostics tool and help the stakeholders to investigate possible problems in their projects.

2 The VISDOM architecture

This section describes the general architecture specified during the first 12 months of the project. The provided architecture is a high-level design that will eventually lead to the final architecture with more detail and empirical validation. The presented reference architecture is aimed to result in slightly different real-world implementations for individual cases. Thus, we also specify possible implementation related issues and points of consideration as a part of the solution to highlight a need for customization per each unique case. One of the most important aspects for consideration includes the centralization of different components in the architecture. For example, to avoid having single monolithic components that handle each and every situation and need, the actual implementation may feature multiple customized versions of the components deployed in parallel. As an example, a real-world implementation of the architecture may utilize one or more data management systems to match the specific needs of certain teams while having a single data management system serve other users. As a final note, the remaining open questions have been marked with a research question (RQ) tag within the text.

The overall architecture is comprised of three main elements: data management, visualizations, and dashboards. First, the data management system is responsible for providing a unified interface for data access and facilitates dynamically adding new tools as data sources. Second, visualizations represent generic components that offer advanced functionality, such as zooming in to the problem root causes, and can be configured based on stakeholder needs and the underlying dashboard. Lastly, the dashboard composer is responsible for matching visualizations to user needs with customizable views. The composer decouples visualization logic from the dashboard itself, allowing the architecture to support both stakeholder and company specific dashboards that can support all activities in DevOps development.

2.1 Data architecture

The data management system fetches data from various tools used in software engineering and provides a uniform data interface for visualizations to consume. The following figure describes the general idea of the data architecture.

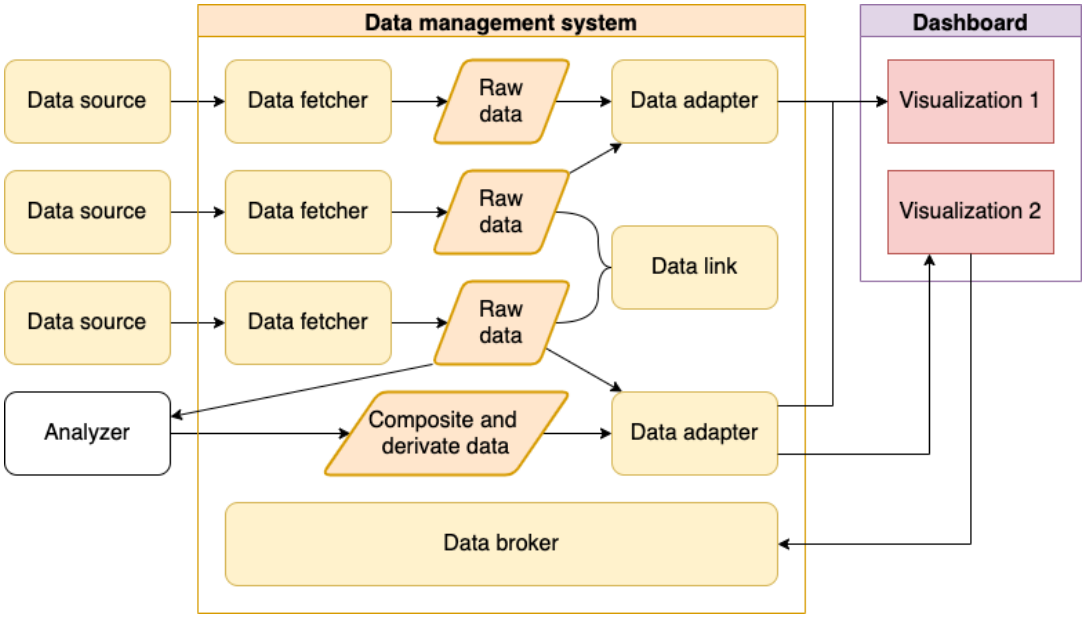

Figure 2 Data architecture

This data management architecture has been developed in a brainstorming meeting, and we have noticed similarities with data architecture used in Q-Rapids project (https://www.q-rapids.eu). ***RQ0****: how does this architecture compare to research state of the art?*

The components of the data architecture are data sources, data fetchers, raw data, data adapters, data links, and data broker.

- **Data sources** are typically SW engineering tools that produce data, but our system is flexible for other types of data sources, too. In particular, various analysis tools can compose new data items, composites and derivates, based on existing raw data items. The VISDOM project aims to develop such tools and enable the use of external tools. The architecture shall support both push and pull paradigm for data retrieval, but the actual reading from the data sources is done with data fetchers.
- **Data fetchers** are responsible for transferring data from data sources to the data management system. These fetchers may be called both by the data management system in a pull approach and by data sources in a push approach.
  However, the data fetcher architecture has two open research questions:

  RQ1*: Should data fetchers filter out obviously erroneous data or is it up to* data adapters?
  RQ2*: The mixture of push and pull approaches increases complexity, which in turn sets a requirement to carefully design update cycles. We need to consider performance, synchronization of data, and possible real-time properties like in real-time BI[1]. More analysis and detailed design are required.*
- **Raw data** represents unmodified data as it is received from tools. Using raw data instead of pre-filtered data provides better support for advanced visualization operations, such as zooming in, stated in business requirements. Additionally, this approach allows for data storage in the original tool itself.
- **Data adapters** implement a two-way adaptation. Firstly, it provides visualizations a uniform data model regardless of the data source. As an example, the data adapter may facilitate visualization for similar "ticket" data regardless of it being fetched from Jira or Trello. Secondly, data adapters may provide different data for different types of visualizations. However, there is an open research question concerning the data adapters:
  RQ3*: can this approach be implemented without performance problems? Naturally, some caching for optimization should be enabled.*
- **Data links** are used to link semantically connected data together. As an example, a content management system (CMS) commit may be linked with a ticket, repository, developers etc. Data links can be created automatically when new data is received or created as a result from various analysis tools.
- **Data broker** serves as a directory for the data, adapters and related metadata. It provides a query interface available to all components in the system. The data broker supports a publish-subscribe model where system components can subscribe to change notifications and receiving new data. The data broker has an open research question.
  RQ4*: the boundary between data broker and adapter is still a bit unclear – most probably the API provided by the data management system should be uniform.*

[1] See e.g., https://en.wikipedia.org/wiki/Real-time_business_intelligence

2.2 Visualizations

Conventional visualizations often present a single viewpoint to the underlying data. More advanced visualizations, such as the electrocardiogram (ECK/EKG), are pieces of code that define or create concrete visualizations from one or several data sources. These visualizations represent generic software components that are not limited to displaying data but may also include functionalities related to interactivity, for example zooming in on and investigating details, which allow the user to search for the root causes of various issues. Additionally, these visualizations can be configured for specific stakeholder needs and preferences, certain dashboard usage via for example certain color schemes, and for various logical data sources and adapters. The logical data sources may represent for example an issue management system without specifying a concrete tool like Jira. The correct adapter for each logical data source can be found with the query functionality provided by the data broker. These advanced features and forms of interaction should conform to a common specification for translating visualization outputs to a unified viewpoint.

From an architectural viewpoint, individual visualizations are treated as black boxes that each handle their own data retrieval, processing, and presentation. However, their outputs are optimized for dashboard usage and may be controlled by the dashboard composer. This approach lends itself well to features from the micro-frontend paradigm – the development of small, independent applications working together to create a larger frontend – where each team is responsible for the development of a micro-frontend from back to front. Thus, each team controls all required code, the building and deploying of their micro-frontend, and are not in any way reliant on the deployment schedule of other teams and their micro-frontends. Each team can thus also choose the technologies they need for their specific micro-frontend. However, each implementation of the reference architecture must weigh how heavily they invest towards the micro-frontend pattern as separating development activities with a micro-frontend-oriented approach may impose bottlenecks for data management.

There are several possible architectural considerations for utilizing a more-or-less centralized data management system. First, all new data sources would need to be compatible with the data management system. Unless modifications are allowed to the unified data model, the underlying data model must be defined once and for all for everyone at the very beginning and enforce its usage without modifications. In the latter case it might become problematic to isolate such modifications according to autonomous teams or specific micro-frontends. Second, the current depiction of the data broker suggests a system-wide uniform component that all micro-frontends would need to use. For independence of micro-frontends, there would need to be as many brokers as there are micro-frontends, which does not seem sensible, or there would need to be one unified data broker, which would serve all micro-frontends. In the case of one unified data broker we need to consider responsibility of developing the data broker - if we need to modify data broker along with addition of data sources, and addition of data sources translates to new micro-frontends to accommodate the new data sources, then the data broker needs to be modifiable by every micro-frontend team, which goes against the philosophy. Finally, it would be sensible to re-use data fetchers with very similar data sources even though they can be done independently per team in cases where micro-frontends are implemented based on individual data sources or tools. Moreover, the concept of push and pull approaches for data management workflows must be considered in relation to micro-frontends. Whichever approach is selected, the data fetchers, linkers, and adapters must also be implemented accordingly.

One possible solution to the centralization issue of data management may be addressed by dividing development teams or specific micro-frontends into collectives that focus in

specific visualization concepts or domains, and thus utilize a narrower selection of tools. This approach may utilize both different data models and a customized version of the data broker to serve a larger collection of visualizations without imposing restrictions on the whole organization. To reiterate, each implementation of the reference architecture must weigh how heavily they invest towards the micro-frontend pattern depending on how loosely individual visualizations can be coupled and whether it imposes restrictions on the data management system. This leaves an open research question ***RQ5****: To which extent does the micro-frontend pattern hinder advanced visualization functions and data management?*

2.3 Dashboards

The dashboards designed for different roles, needs and stakeholders are a key concept in VISDOM. A dashboard consists of one or more visualizations that together constitute a view that answers the visualization needs of the user. The proposed dashboard solution must be generic enough to support all activities in DevOps development in both stakeholder and company specific dashboards. This can be achieved by decoupling visualization logic from the dashboard itself. This approach provides the benefit of allowing dashboards to suit all stakeholder needs without imposing limitations on what is being visualized and allows customization for both stakeholder and company specific dashboards. Additionally, this approach supports both research and development activities by providing a common platform for deploying visualization services between development teams in an agile way.

The dashboard design is shown in the following figure. The leftmost boxes in yellow represent data retrieved through the previously described data management system. On their right, individual visualizations in red are implemented as visualization services by various teams, organizations and other entities interested in specific visualization concepts or domains. The dashboard composer in blue allows services to register their available visualizations for use within the dashboard composer. The composer provides a mechanism for selecting visualizations based on the current user's role, tasks, and other suitable factors. This approach makes it possible to provide a default dashboard based on stakeholder information, customize the provided views based on personal needs, and to create a feedback loop on collecting stakeholder information through dashboard usage. This data can be used to provide more accurate default dashboards, recommend visualizations for tasks, and to determine which visualizations are used in practice.

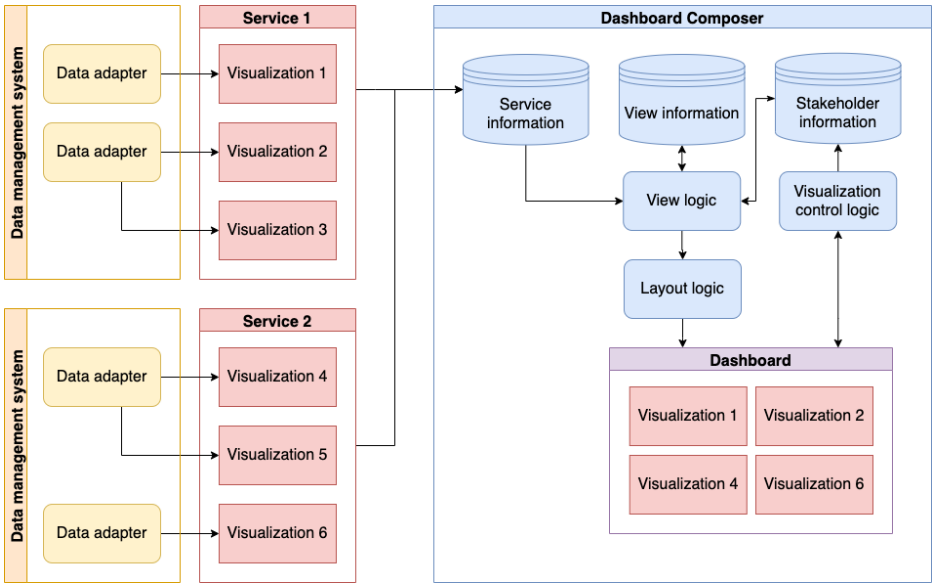

Figure 3 The dashboard composer architecture.

Composer creates a default dashboard from its registered applications, based on the stakeholder's role. Users may also customize and create new dashboard views that address their individual visualization needs. The composer consists of the following subcomponents, depicted in blue in figure 3, and their roles within the system are as follows:

- **Service information** is stored in the composer as a basis for creating varying views in the dashboard. Visualization services are provided by teams, organizations or other entities that focus on visualizing a specific concept or domain. Each service registers their available visualizations to the dashboard composer according to a shared specification. This specification contains metadata about the visualization's purpose, which is used to match available visualizations to the current dashboard user's needs, tasks and role. To put it simply, service information represents a collection of individual visualizations that can be used in the composer – including the metadata about what they visualize and how.
- **View information** is stored in the composer to represent groups of visualizations for a specific interest, task or other suitable concept for building individual dashboard views. In essence, view information includes a selection of visualizations, preconfigured perspectives, and layout information. These configurations are then rendered as individual dashboard views. Finally, users may configure new views to their dashboards based on their interests and tasks.
- **Stakeholder information** is stored in the composer as a basis for offering preconfigured views based on role and for improving existing preconfigured views. Stakeholder information includes an initial set of assumptions and educated guesses on which visualizations are relevant for different stakeholders, and an accumulated collection of dashboard usage data. The collected usage data helps to understand which visualizations remain relevant in daily usage and which roles and tasks are supported by visualizations.
- **View logic** implements the necessary functionality for combining the available visualization services to the user's needs while considering the available stakeholder information. The view logic component facilitates the creation,

management, and rendering of individual view configurations, and houses the logic used in selecting views and visualization perspectives for different roles.

- **Layout logic** implements the necessary functionality for producing a coherent layout for the dashboard in terms of laying out individual visualizations on a grid and providing a responsive display for screens of varying size. Views may contain information for content layout and visualization metadata may request specific element sizes in the dashboard but eventually the layout logic is responsible for determining the correct presentation for the current view.
- **Visualization control logic** implements the necessary functionality for inter-visualization communication and relaying of events, such as changes in visualization perspective, to all the visualizations present in the current view. **RQ6***: how to handle situations where different visualizations in a dashboard should work simultaneously, for instance scroll to the same position in time,*

2.4 Security and privacy concerns

Security and privacy are essential aspects for architectural consideration as the proposed system might allow unethical or illegal tracking of stakeholders and other malicious activities if left unchecked. Features such as access control might be omitted for research prototypes but should still be considered in the overarching architecture. The key aspects for further consideration include access control in the form of authentication and authorization, and anonymization of data. These issues present a need for some kind of access federation system that provides a cross-domain solution for bridging access rights. Alternatively, future versions of the architecture may offer alternative solutions for working around this issue by deploying multiple data management systems for specific uses. This approach will depend on the degree of centralization for each implementation of the reference architecture.

This leads to the following requirements:

- The data management system, including both data fetchers and adapters, shall include an anonymizing service. ***RQ7****: where should the anonymization service be?*
- The data management system shall include a user/access-token federation system. ***RQ8****: specification and design of the federation system. One of our earlier studies*[2] *may be used as a starting point.*
- The data adapters need to implement some kind of “cross-domain” solution that bridges access rights. ***RQ9****: specification and design of the cross-domain system.*

3 Standardizable components and interfaces

This section summarizes both the exploitable software components that can be utilized either independently or together, and the standardizable components that be pushed relevant standards for industry-scale interoperability. The previously introduced data fetchers, adapters, linkers, and visualizations form a baseline selection of the exploitable software components. Each of these components must addressed based on their structure, general behavior, and interfaces in order to present a unified architectural

[2] Anna Ruokonen, Otto Hylli, Kari Systä, Samuel Lahtinen, Service Composition for End-Users, 13th Symposium of Programming Languages and Software Tools (SPLST’13), Szeged, Hungary, 2013

solution. This architectural proposal is aimed to support both research and development in the VISDOM project.

Data fetchers represent a link between the VISDOM data platform and external data sources. The fetchers present an opportunity for creating a standardized software component where the data fetching strategies are implemented in a way that a) allow the creation of new data fetchers for tool vendors and independent collaborators, and b) supports the underlying data management architecture with a clear interface towards VISDOM elements. This includes specifying the strategy on when and how data fetching occurs and whether caching is implemented for performance reasons. The interface for the fetchers is another aspect for a standardized interface that corresponds to the requirements specified by the data adapter components.

Data adapters are responsible for providing visualizations a uniform data model regardless of the data source and for providing different data for varying types of visualizations. Data adapters are the key in combining data from several sources for VISDOM related visualizations. Thus, the adapters present a suitable opportunity for standardization on both interface and component levels. The interface for providing visualizations a uniform data model regardless of the data source may be specified in a form of a generic description that aids implementation and integration efforts. On a component level, the component's internal behavior may be suitable for standardization as it operates towards both raw data access and visualization related logic. Moreover, the solution should accommodate for the required workflows in data management, such as the push and pull approaches.

Data linkers are used to link semantically connected data together. In order to reach this goal in practice, the production of new data linkers must be as flexible as possible and the interface towards data storage is well defined. Currently, the interface is assumed to resemble, or be similar to, the data fetchers and linkers described previously.

Visualizations represent a more free-form selection of software components that may greatly vary between implementations depending on their intentions. However, each of the visualizations must still adhere to a common specification that allows the dashboard composer to control them in order to assume a uniform perspective within the dashboard. This requires both a common behavioral mode for software components that can be extended for individual implementations, and a uniform interface for the communication between visualizations and the dashboard composer. Furthermore, the visualizations must be able to communicate their capabilities and intent to the dashboard composer in order to be matched against specific user needs. This is a key requirement for supporting all DevOps activities for both stakeholder and company specific dashboards that are composed of new visualizations developed in VISDOM project.