International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 12 (2015) pp. 32387-32397
© Research India Publications
http://www.ripublication.com

Dashboard To Show Key Metrics For Teams Following Agile Methodology

Sangeeta S. and Malarvizhi V.

Dept. of Computer Science, Amrita School of Engineering
Bangalore, India
sangeetashivajirao@gmail.com
Dept. of Computer Science, Amrita School of Engineering
Coimbatore, India
malar.v1905@gmail.com

Abstract

Developer awareness and being well informed about the project status are important factors for successful product development. Understanding, analyzing and transforming results into a form that helps the team to track the progress and quality of a product will help in their efficiency, effectiveness and performance. There is a need to ease out this process of analyzing data and sharing them across members of a team. This paper talks about the implementation of an on premise information visualization tool or a dashboard that exposes key metrics for developers who are following an agile model. The tool involves a widget based presentation of metrics and allows creation of customizable dashboards. This dashboard is aimed to provide all the statistical data of a project running on Sonar & Jenkin by giving a comprehensive measure of risk to ship a product.

Keywords— dashboard, information management, agile methodology, decision making, metrics, component, analysis.

I. Introduction

Quality of any product in the market is the most critical factor for its acquisition [6]. Monitoring this quality during the development phase is an essential investment a development team makes. But, it is necessary to ensure that costs of assessing quality are not more than the benefits from the product. Thus, dashboard is one such visualization tool that is built keeping the cost factor in mind. This tool helps the development teams to measure the extent to which the project meets the defined quality specific goals. Dashboards are able to identify defects better and showcase changes by storing historical data. Individual dashboards target a project, so customizability of a dashboard plays an important role. This feature allows the dashboard to be project specific and to launch a custom-built dashboard.

Software development teams follow an agile model to cater to the changing needs of the customer [5]. It is an iterative process, allowing users to incorporate changes in the design of the project as and when there is a need. Requirements of the project evolve over a period of time, which requires continuous improvement and flexibility in development. With these requirements, the agile model needs development tools to increase awareness among developers. These tools help the teams to coordinate activities and realize important measures on the project. Agile teams follow development in sprints which requires the team to have working software at every point in time. This criterion of the model is met with the help of continuous integration of the code for new commits. Continuous integration is feasible with the use of automated tests written and run on the source code. A dashboard helps a development team to quickly learn about the status of the software. The dashboard gathers data from multiple sources and organizes them into valid information in the form of pie charts, numeric values, figures and icons.

Data-driven process of making decisions has become an obligation at most organizations and analytics on the data provides them an edge over any opinion [4]. It is a challenge for developers at the organization to maintain awareness of the status of the product and gain an understanding of their colleague's work. So, a dashboard is a tool that addresses this issue and is an impressive solution to this problem. Essential information for developers are often unavailable in the right format, however, a dashboard presents this information to make it worthwhile. Dashboards are now smart to provide a qualitative measure of how the quantitative measures have changed from the last time.

Providing status about any project is a challenging task as it does not involve giving information about its code base alone. Project awareness does not depend only on individual metrics, but requires a cumulated result. In this paper we present the implementation of a widget-based dashboard. The dashboard acts as a support system for development teams and stir healthy competition among one another. Maintaining awareness is time-consuming activity and the dashboard cuts down this amount of time spent on keeping oneself aware about important metrics. Our dashboard provides a single reporting web interface in the form of widgets for several key metrics. This dashboard is used as a tool to track performance and other important updates on the project. The dashboard shows a pie chart on the team's contribution in defects.

II. RELATED WORK

In [3], Awareness 2.0 uses a feed based system to provide awareness on projects, developers and tasks using IBM's jazz software development environment. It provides an insight into the workspaces of other developers and current activities in a project. Feeds are displayed on the dashboard as a viewlets; they are updated continually with current events. These viewlets can be configured to include event logs and build information. Developers can prioritize contents on the viewlets to personalize the application; this allows them to make tailor-made dashboards for their project. Dashboards also help in identifying bottlenecks during the development of the product; it helps managers to compare performance of different teams.

Using a widget-based dashboard [1] allows users to aggregate particular widgets to get meaningful results. The customers can enable communication between widgets such that output of one widget is the input for another widget, this facilitates inter-widget communication. This has been implemented as a web application with the main application logic on the client side using JavaScript, server-side interactions using AJAX and inter-widget communication using JavaScript API.

III. IMPLEMENTATION

This tool has been developed for an organization whose development teams use Jenkins for Continuous Integration and Sonar to examine code quality. A task run on Jenkins is also called a job. Fig. 1 shows the jobs run on Jenkins. The dashboard makes use of the Jenkins API in order to fetch the last build report for important jobs. The Fig. 2 shows the information provided by sonar, which the dashboard pulls using phantom.js. The Fig. 3 gives an idea about the metrics taken from Jenkins and Sonar.

In the following section we will see in detail the important metrics the dashboard displays.

A. *Metrics*

This tool contains metrics which are displayed on the widgets, also known as gadgets. Each of these metrics on the widget supports the tool in assessing the performance of the product. Widgets are small elements on the dashboard showing metrics from different data points. This section explains each of the metrics in detail.

1) *Continuous Integration Build:* The convention of merging individual developer branches into one master branch for a considerable amount of times during the day will require that the developers run integration tests on the master branch. The CI Build server runs these integration tests automatically when it detects a new commit. This practice makes it necessary for the developers to stay aware about the test results on a daily basis. Jenkins is used to run tests and perform repetitive tasks including nightly run on a project. The number of passed tests, time for the longest running test, number of failed tests and the time for the top 10 shortest tests are all the important metrics provided by the CI Build server. This tool enables effortless access of these metrics to the developers with the help of a CI Build widget. Fig. 3 shows the CI Build widget on the dashboard.

2) *Code Coverage:* Bugs in a product are always less when the source code is thoroughly tested. Engineers write test suites to ensure meticulous testing of the product's source code. And the process of running these suites assures quality of tested code. A widget on code coverage Fig. 3 shows the extent to which the source code is tested by the test suite. A product will have lesser

chances of finding bugs when the code coverage is high. Thus code coverage is an important factor to mark a product as bug free.

3) *Defect Count:* The quality of a product is always measured in terms of the number of defects. Total count of all the defects in a product is important to analyze and compare its competence with other products in the market. The number of defects is acceptable when they are below a particular threshold. While the defect count always provide a cumulative number of all the different defects, it is necessary that this number is always low. Fig. 3 shows the defect count widget on the dashboard.

4) *Static Violations:* The static violation gives the count of the number of dead lines of code. These lines are always executed in a program but their results are never used. Existing static violations in a code leads to a wastage of computation time, memory and CPU cycles. In order to avoid this, it is necessary that static violations are maintained as low as possible. It is also important that the static violations do not increase with continuous merging of commits. This widget shows the number of static violations and the decrease or increase of violations.

5) *Defect Statistics:* Certain defects in a product are reported by the customers and some defects need to be fixed within 28 days. The defect statistics widget gives the number of such customer found defects and the number of defects that have remained unfixed for more than 28 days. These statistics are important to track the number of bugs filed and to prioritize bug fixes in the product. The S1-S2 defects on the widget shows the number of severity-1 and severity-2 defects and these require an immediate fix.

6) *Load and Automation:* Load and automation widgets show the test statistics of performance and disruptive tests run on the source code. It gives a status of API and UI tests including IE and Firefox under UI. The status shows pending, aborted, failed or in-progress. If these tests have been completed, the number of failures with the total number of tests are shown.

7) *S1-S2 Defects:* S1-S2 defects are the severity defects in a product and this number is shown in the form of a pie-chart against all the scrum teams for a product. The severity of a defect calls for attention and developers prioritize this task over others. A product having a history of less S1-S2 defects are better and more stable for shipping.

8) *Risk to Ship:* An aggregated status of the overall risk in shipping the product is provided by this widget. The rules for calculating Risk to Ship requires that the code coverage has improved, static violations are lesser than a threshold and has not increased since the last run. Results from these widgets are taken for the computing risk to ship, as shown in Fig. 1. The CI Build has to have zero failed tests for a green signal in shipping the product. If the product fails to meet the above criteria, the risk to ship is high and a red signal is displayed on the widget. Hence, well informed decisions are made by the means of interwidget communication and business inteligence.

B. ***Detailed Design and Implemetation***

The dashboard tool contains an API, UI server and a data store. Fig. 2 shows the interactions of the API server with the UI server and the data store.

This tool has been developed using node.js as a development platform along with express.js framework to manage the front-end and the back-end efficiently. The front-end has been developed using AngularJS for making HTML dynamic. Phantom.js has been used to take screenshots of the webpages for data and log4js to write logs. No-SQL Cassandra database has been used for storage. With our implementation, we aim to provide an effective tool to allow developers and managers to make better informed decisions. A detailed list of all the node.js packages used by the front end and the backend is shown in Table I.

TABLE 1: PACKAGES USED

Serial No.	Node.js packages		
	Package	Front-end (UI server)	Back-end (API server)
1	Cassandra-driver	Yes	Yes
2	Express	Yes	Yes
3	Jenkins-api	No	Yes
4	Log4js	Yes	Yes
5	Phantom	No	Yes
6	Jsonfile	No	Yes

The front-end and the back-end are separated as UI and an API server respectively. The front-end experience is wholly controlled by the UI server, while the data is processed and business logic is implemented in the API. The UI server & the API server speak using the data. The API server collects data from different data points, analyses & processes them to transform it into a form usable by the UI server. The RESTful architecture has been deployed using an express.js framework for Node.js. Express.js supports HTTP verbs of post and update. While, AngularJS provides routes and binds the controller and the view enabling data to be accessed on the UI.

This overall design can be split up into three major application components- Database, Loading and Customizability.

1) *Database Component:* A key space in Cassandra database has been used by the tool to store data URLs of the metrics for each product. The product name uniquely identifies fields in the products table. Every product contains attributes for relevant metrics and the attribute values are stored as key-value pairs in Cassandra. A table called teams stores the details of every team member working on a product. The fields in the table include member's id, member's name, team name and product name. This table is used for the defect distribution metric which shows distribution of defects among several

teams in a product. The defect distribution metric shows results in the form of a pie-chart.

2) *Loading Component:* The primary component of the application is the loading component, and it is responsible for the entirety of the tool. The loading component comes into picture in every aspect of use in the application. It is responsible for interactions with the DB, ensuring API communicates with the UI and in allowing customizability of the tool.

The main page of the application is first loaded which allows us to choose a product from the list of products for which the dashboard exists. Each of these dashboards in the list has a dashboard-id. The AngularJS's $routeprovider uses a route with this dashboard-id to load the corresponding HTML page for the product when a particular dashboard is chosen. Every widget on the UI makes a request to the API server by calling specific functions. Each of these functions in the API server reads data URLs from the database using the product name. A phantom.js page for the URL read from the database will be created to take screenshot of the web page and to access the DOM elements. DOM elements are read using jQuery and these results are compared with the previous value of the metric. Previous values are stored in a text file and every time a function in the API is called, the new data will overwrite the previous value. This process happens in case of a widget taking data from the sonar servers. However, for a widget reading data from Jenkins, the API server calls Jenkins API using the server details taken from the database. The results returned as response from the API call is used for computation and compared with the previous values stored in the text file. The new data overwrites the text file. Hence, the new metrics obtained after computation is sent to the UI as a response to the request made by it.

3) *Customizability Component:* Along with viewing a dashboard, the tool is built to allow users to customize their own dashboards. The tool provides end users, the capability to launch a new dashboard, edit configurations for a dashboard, and delete an existing dashboard. It requires the UI server to perform operations on Cassandra to insert a record for a new dashboard that is launched, edit a record using the product name and delete the record from the products table to drop a dashboard. Upon hovering over each product name on the homepage, the end user will see a delete and edit icon appear for that product. Clicking on edit icon takes the user to a form page containing current configurations in the form fields. These fields can then be modified and saved back to the database. However, clicking on the delete icon will ask the user for a confirmation and removes the record from the database. In both these cases, the application is redirected back to the main page of the dashboard without the deleted product in case of delete and a modified database record in case of edit.

The dashboard loads all widgets by default. A check box that appears when hovered over these widgets will let the end-user check and uncheck the widget to show and hide respectively. Thus users can choose particular widgets to remain on the dashboard while the rest could be hidden.

IV. END USER VIEW OF THE DASHBOARD

From the end user's point of view, the tool allows him to first pick a particular dashboard from a list of products as shown in Fig 4. On choosing the product, a dashboard is launched with all default widgets as in Fig 3. User also has the privilege to choose the widget they want to show or hide as shown in Fig. 5. The user also has an option to edit the configurations from the edit page, shown in Fig 6. The tool also allows the end user to add a new dashboard from the new dashboard page, shown in Fig. 7. The user is also given an option to delete a dashboard from the main page.

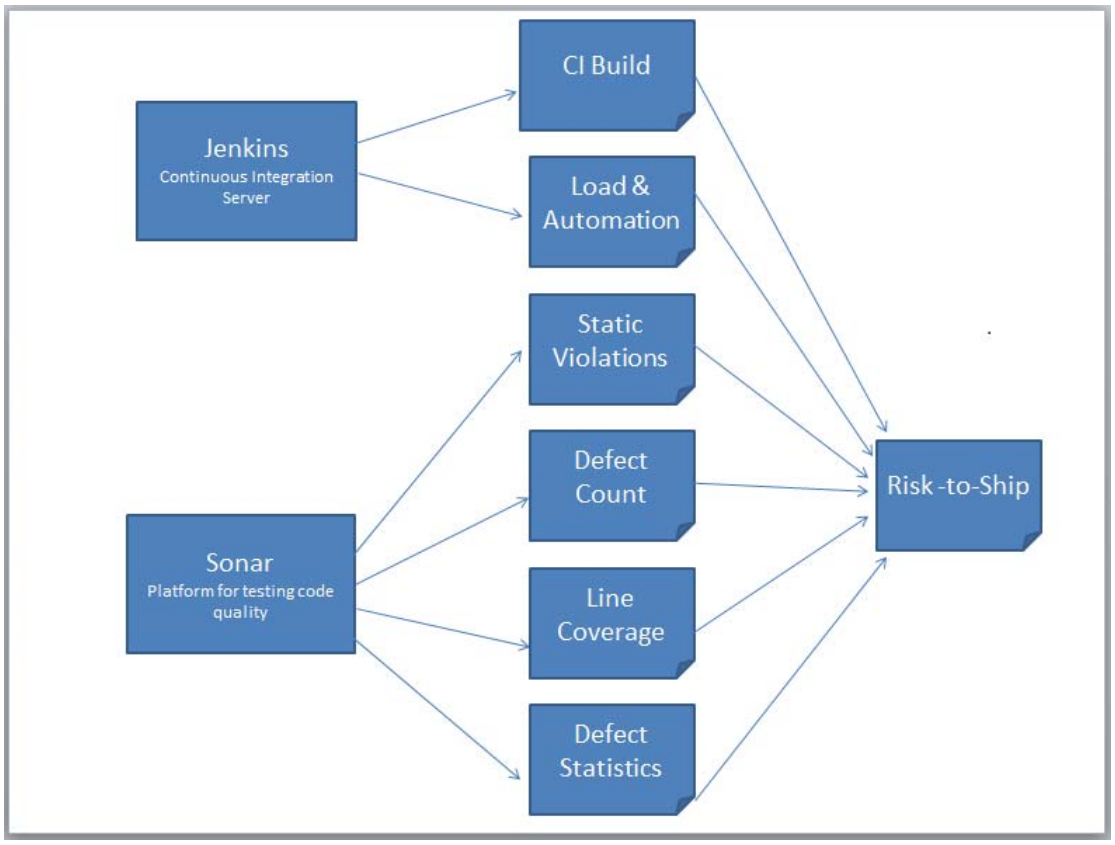


Figure 1: Metrics from Jenkins and Sonar

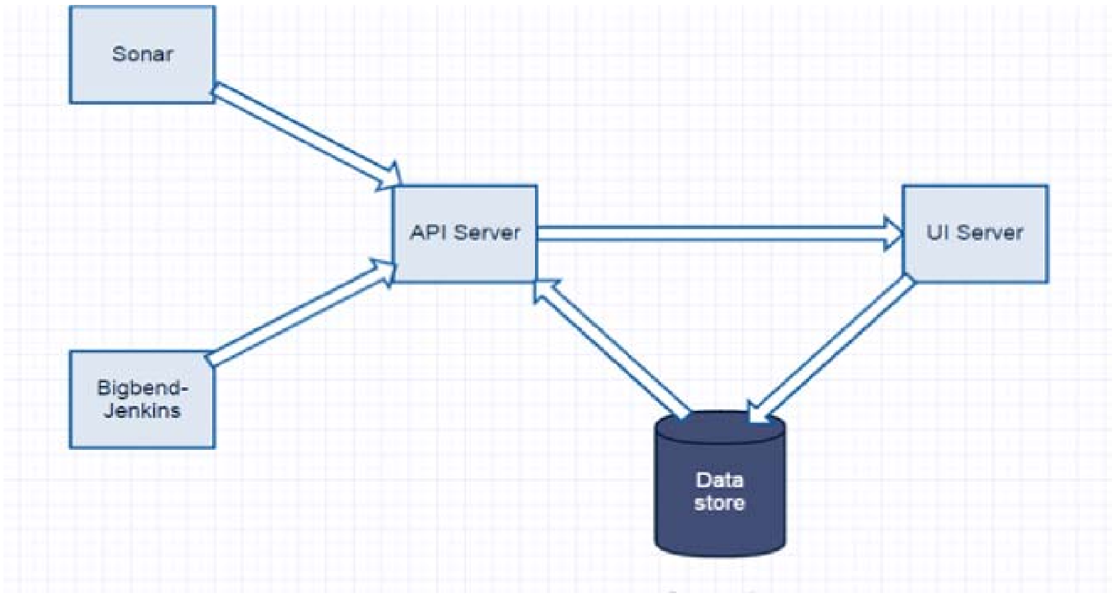
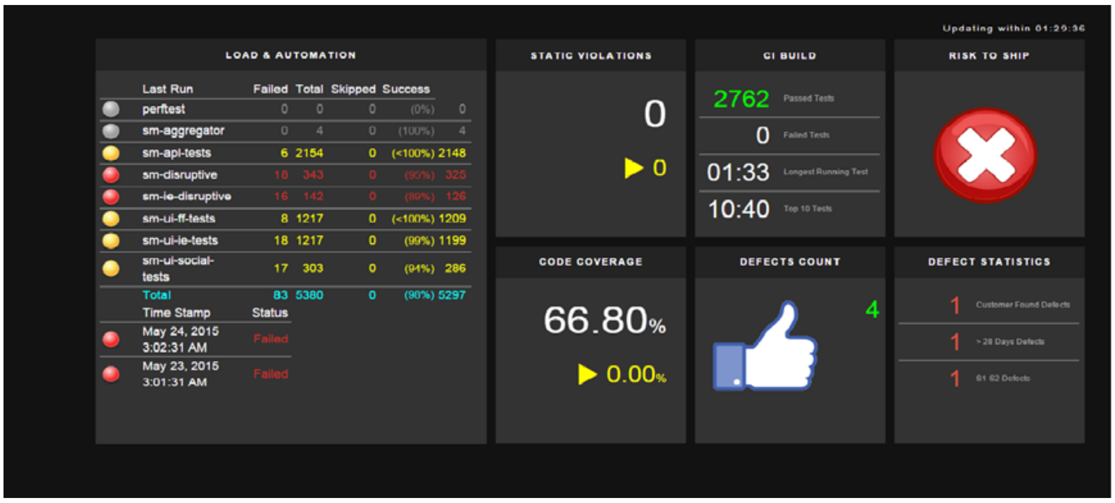


Fig. 2. Interactions of the API server with the UI server, Data store, sonar and Jenkins


Updating within 01:29:36

LOAD & AUTOMATION

	Last Run	Failed	Total	Skipped	Success	
●	perftest	0	0	0	(0%)	0
●	sm-aggregator	0	4	0	(100%)	4
●	sm-api-tests	6	2154	0	(<100%)	2148
●	sm-disruptive	18	343	0	(95%)	325
●	sm-ie-disruptive	16	142	0	(89%)	126
●	sm-ui-ff-tests	8	1217	0	(<100%)	1209
●	sm-ui-ie-tests	18	1217	0	(99%)	1199
●	sm-ui-social-tests	17	303	0	(94%)	286
	Total	83	5380	0	(98%)	5297

	Time Stamp	Status
●	May 24, 2015 3:02:31 AM	Failed
●	May 23, 2015 3:01:31 AM	Failed

STATIC VIOLATIONS

0
▶ 0

CI BUILD

2762	Passed Tests
0	Failed Tests
01:33	Longest Running Test
10:40	Top 10 Tests

RISK TO SHIP

CODE COVERAGE

66.80%
▶ 0.00%

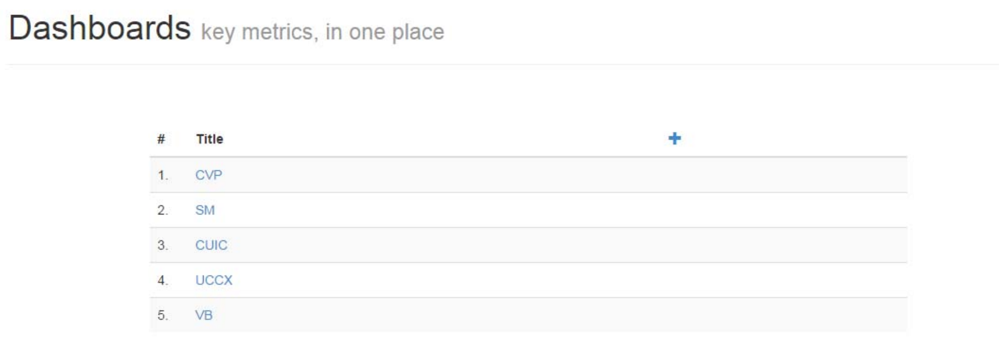
DEFECTS COUNT

4

DEFECT STATISTICS

1	Customer Found Defects
1	> 28 Days Defects
1	S1 S2 Defects

Fig. 3. Dashboard with default metrics



Dashboards key metrics, in one place

#	Title	+
1.	CVP	
2.	SM	
3.	CUIC	
4.	UCCX	
5.	VB	

Fig. 4. Main Page of the tool

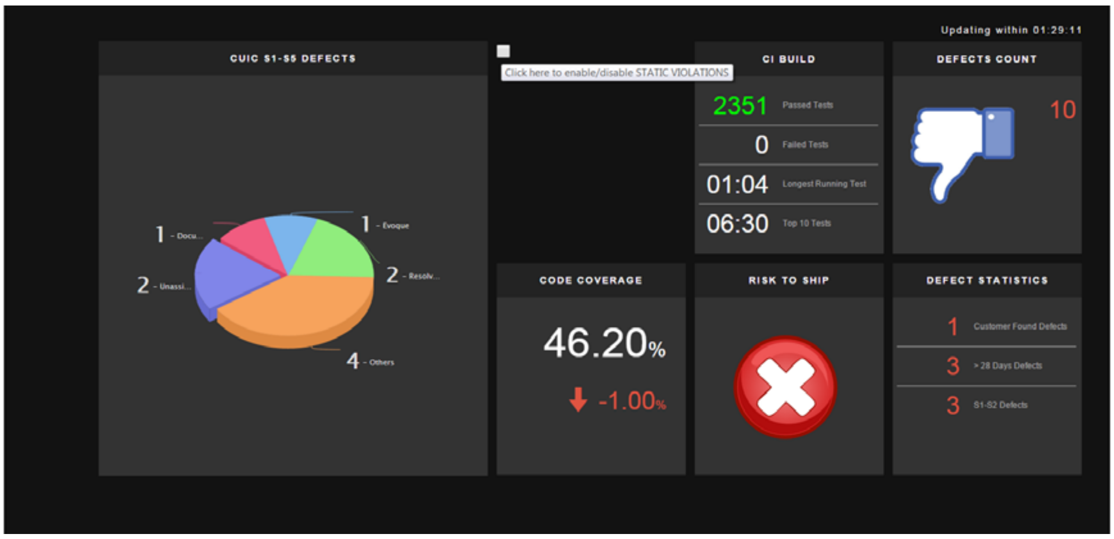
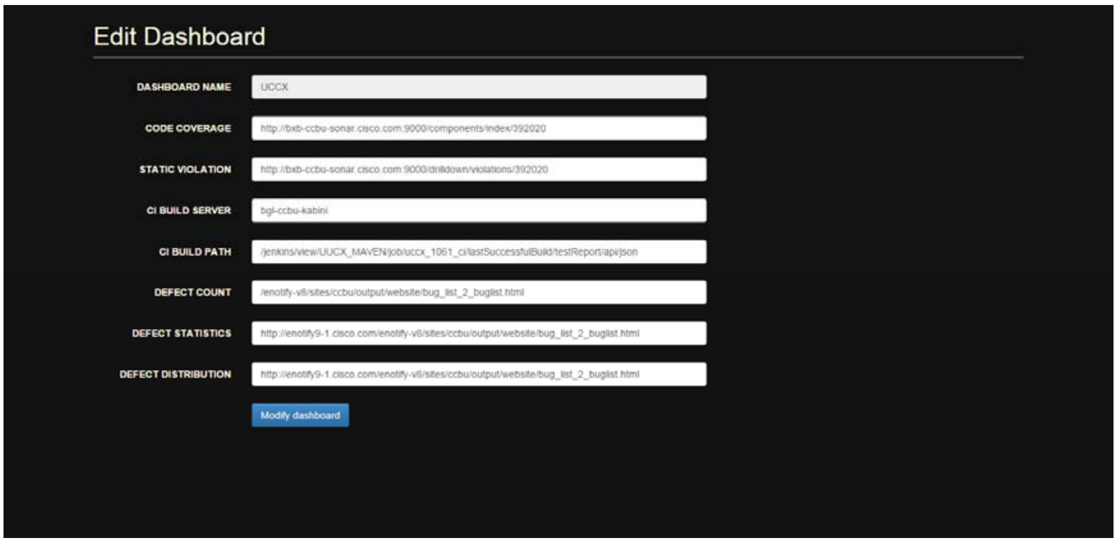
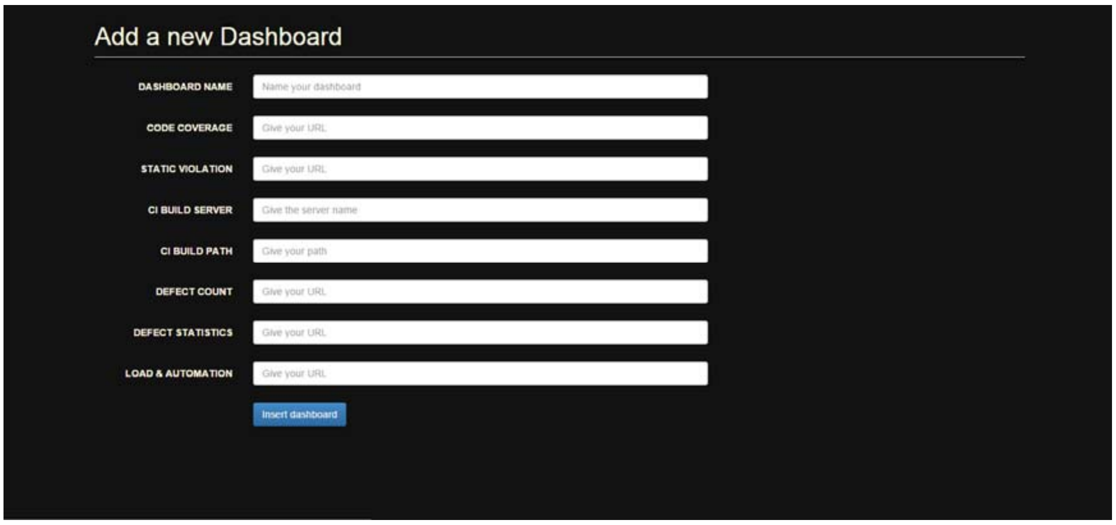


Fig. 5. Dashboard with static violations widget hidden


Edit Dashboard

DASHBOARD NAME	UCCX
CODE COVERAGE	http://bxb-ccbu-sonar.cisco.com:9000/components/index/392020
STATIC VIOLATION	http://bxb-ccbu-sonar.cisco.com:9000/drilldown/violations/392020
CI BUILD SERVER	bgl-ccbu-kabini
CI BUILD PATH	/jenkins/view/UUCX_MAVEN/job/uccx_1061_ci/lastSuccessfulBuild/testReport/api/json
DEFECT COUNT	/enotify-v8/sites/ccbu/output/website/bug_list_2_buglist.html
DEFECT STATISTICS	http://enotify9-1.cisco.com/enotify-v8/sites/ccbu/output/website/bug_list_2_buglist.html
DEFECT DISTRIBUTION	http://enotify9-1.cisco.com/enotify-v8/sites/ccbu/output/website/bug_list_2_buglist.html

Fig. 6. Page to edit dashboard


Add a new Dashboard

DASHBOARD NAME	Name your dashboard
CODE COVERAGE	Give your URL
STATIC VIOLATION	Give your URL
CI BUILD SERVER	Give the server name
CI BUILD PATH	Give your path
DEFECT COUNT	Give your URL
DEFECT STATISTICS	Give your URL
LOAD & AUTOMATION	Give your URL

Fig. 7. Page to add a new dashboard

V. CONCLUSION

In order to make well informed decisions and to stay aware about important metrics of a project, it is necessary that a single reporting interface is used. A tool that gathers data from multiple data points thus reducing the time and effort spent by individuals will always add value. In this paper, we have implemented a tool that collects data from Jenkins servers, performs certain computations on the data and displays results in a way that makes this data valuable. This dashboard helps developers prioritize their tasks and record their progress. Multiple dashboards can be launched for different projects running on Jenkins. This feature fosters healthy competition among different teams in an organization.

In our future work we aim at making the dashboard available on a mobile to provide easy accessibility. The dashboard can be enhanced to modify rules for inter-widget communication to compute risk to ship a product.

References

[1] Nassim Laga, Emmanuel Bertin, and Noel Crespi, "Building a user friendly service dashboard: Automatic and non-intrusive chaining between widgets," IEEE Computer Society, 2009.

[2] Jerzy Korczak, Helena Dudycz, and Mirosław Dyczkowski, "Intelligent Dashboard for SME Managers. Architecture and functions," Federated Conference on Computer Science and Information Systems, IEEE.

[3] Christoph Treude, and Margaret-Anne Storey, "Awareness 2.0: Staying aware of projects, developers, & tasks using dashboards and feeds," IEEE .

[4] Olga Baysal, Reid Holmes, and Michael W. Godfrey, "Developer Dashboard: A need for qualitative analysis," IEEE, vol 30, issue 4, April 2013.

[5] Julia Paredes, Craig Anslow, and Frank Maurer, "Information visualization for agile software development teams," Second IEEE Working Conference on Software Visualization, 2014.

[6] Florian Deissenboeck, Elmar Juergens, Benjamin Hummel, and Stefan Wagner, "Tool support for continuous quality control," IEEE, vol 25, issue 5, September 2008.