1. Introduction

Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently. Modern CI-tools such as Jenkins-CI, SonarQube or GitHub generate a bulk of data with each commit [1]. One of the main problems is that relevant information about the quality and health of a software system is both scattered across those stand-alone CI tools and across multiple views.

Therefore stakeholders need more time to come up with answers for the questions regarding the software quality and an integration of CI-tools lead to more accurate answers in less time compared to the standalone use of CI-tools [2]. For example let us consider developer working on multiple projects at the same time, what have my coworkers been doing between start date and end date?

We address this problem by concerning the quality aspects of the Software and customize this information to any of stakeholders, such as developers, project managers, testers etc. For that we present a piping framework for software quality data which includes service combination techniques, integration of information from the entire CI-toolchain and giving customizable or dynamic view using pipelining techniques

2. Background and Motivation

Research Question: How easy for different stakeholders for answering questions about software quality in different perspective with integrated pipelined CI-tools compared to the use of available standalone CI-tools? The combination of modern CI-tools within the development process of a software project is fully automated, and its execution is triggered after every commit. Developers or testers perceive the CI process most often only in case of a build break or test failure. In such a case, they get an automatically generated notification that is via email. Developers can then fix the problem using this information [3]. This kind of exception-driven behavior helps to detect and fix problems as early as possible during integration runs.

Modern CI-tools can generate a bulk of data with each commit, Before the build or test failures,. This data is available across the entire CI-toolchain and analyzing it, for instance, to monitor quality of a system is a time consuming task [4]. This can delay the rapid feedback cycles of CI and one of its major benefits is then not realized.

Software quality is hard to find with the stand alone CI tools. Let’s find the change between two builds and who has changed it? Answering these kinds of questions require two steps. First, we need to know the dates of the respective builds. Then, these build dates can then be used to answer the question itself by investigating, For examples Commit logs, file-diff data, build details, issue details etc. However, to obtain the relevant information a developer must access several different tools and navigate through multiple views [5].

We implement a pipelined integration framework for combining data from various tools. We allow every stakeholder to customize the view according to their need. Here views are created based on the pipes, the stakeholder use. Every pipe includes a quality measurement from various

CI tools. We also provide a default overview on the software quality which includes a set of often needed quality information on the source code. The default overview will be generated based on the composition of the quality measurements by using our newly developed algorithm.

3. Problem in Brief

Even Though the most of the quality information on software are available across the various CI stand-alone tools, there is much time and effort needed to monitor the most relevant quality of the software out of that scattered quality information. The current approaches focus on the information of stand-alone CI–tools rather than not taking the advantage of combined information from CI–toolchain.

4. Aim and Objectives

4.1 Aim

The aim of our research is to develop a piping framework for software quality data to combine the information across the CI-tooling landscape to address the time delay in measuring accurate software quality.

4.2 Objectives

- Gather software information related to evaluate the quality of the software from different CI-tools
- Identify the quality matrices which are required to evaluate the software quality
- Extract quality matrices from the available information sources
- Implement web services to gather the software quality information from the software quality metrics which are not covered in current CI-tools.
- Combine all software quality matrices services together in order to provide a common application programming interface
- Process the data acquired through the APIs using pipe lining techniques
- Provide a default view and customized views about the quality of the software

5. Our Proposed Solution

As described in the figure 01 we use the APIs of the CI-tools to gather data from the CI tools, we also implement a new module and take the information from the API. We process the data and transfer to a common data format. In the pipeline integration we bind the data to pipes. The stakeholders can drag and drop the pipes so that they can tailor their needed information. In the data extraction process we will use a composition algorithm to generate the information needed to generate the default view of any stakeholders.

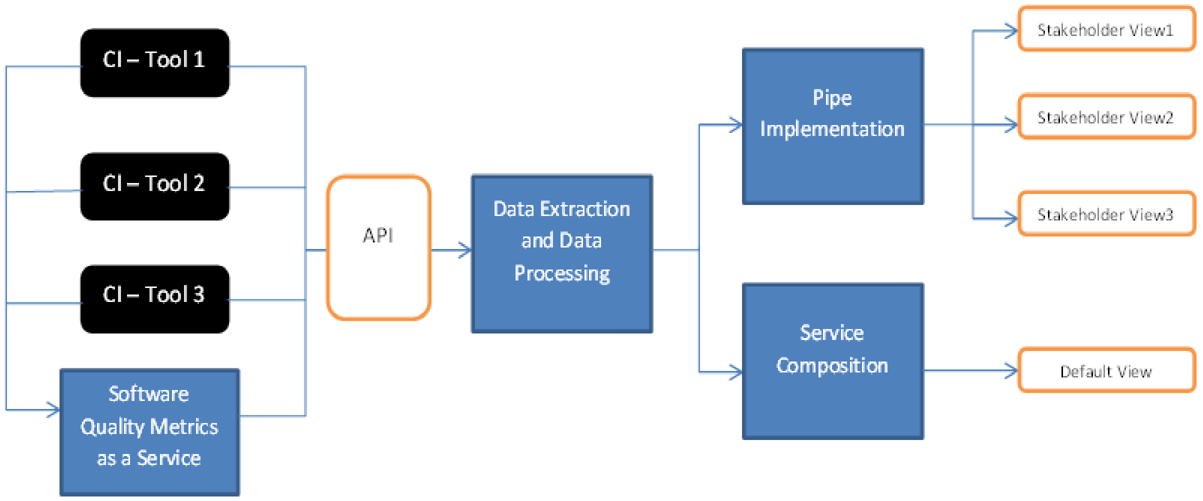

Figure 01

Here we address this problem using different technologies. The solution consists of four major modules.

5.1. Data Extraction and Pre-processing

Quality information will be gathered from various CI tools via Application Programming Interfaces. This is a bit challenging part as CI tools will provide data in different formats. We need to gather information from the CI tools and process them, so that we can come up with the common data format to make easy the data processing.

5.2. Software Quality Metrics as a Service

Certain quality information about the software will be required to evaluate a particular software quality. The above mentioned CI tools have provided some of the often used quality measurements. There are some quality measurements not available in those CI tools, Therefore we need to add them to evaluate the software quality. For an example let us consider sonarQube. It tracks only the issues but it does not have any details about the person who did it.

There are number of software quality metrics which are introduced by various researches. They are not yet implemented, we are planning to implement those software metrics and use its quality information in our data extraction process. We implement those software quality metrics as a service so that any third party can consume.

5.3. Pipe Implementation

We hope to use the pipeline implementation to make the job easy. Quality information from various CI tools is bound to pipes. Piping technology allows a set of pipes to combine in a way so that the bound data in the pipes can deliver useful information for example yahoopipes and etc. Piping technology provides combining any number of pipes. We use this feature to allow stakeholders to customize according to their information needs to the software quality.

5.4. Quality Information Service Composition

There are certain quality measures stakeholders often need. We planned to provide those quality measures in a default view. Here we use different services from the CI tools and use a better service composition to determine the software quality. We will create the composition algorithm for the above purpose.

6. Resource Requirements

Before a project can be authorized it is vital that a corporation consider the resources needed to support it. Resources include needed personnel, equipment, facilities, processes, and funding. Should insufficient resources be available it may be necessary to either outsource portions of the project, or reduce its scope to fall within available resources. The below points describe the required resources for this system to run.

6.1. Software Requirements

- Eclipse
- MySQL Community Server
- GitHub
- SonarQube
- Jenkins
- Jersey Framework

6.2. Platform Requirements

- OS - Windows / Linux
- Apache tomcat
- Java Development Kit
- OSGi Service Platform

7. References

[1] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, “Challenges in software evolution,” ser. IWPSE, Lisbon, 2005, pp. 13–22.
[2] O. Nierstrasz, S. Ducasse, and T. Gˇırba, “The story of moose: an agile reengineering environment,” SIGSOFT Softw. Eng. Notes, vol. 30, no. 5, pp. 1–10, 2005.
[3] Y.-F. Li and H. Zhang, “Integrating software engineering data using semantic web technologies,” ser. MSR, Waikiki, 2011, pp. 211–214.
[4] J. Aranda and G. Venolia, “The secret life of bugs: Going past the errors and omissions in software repositories,” ser. ICSE, Washington, 2009, pp. 298–308.
[5] Martin Brandtner, Emanuel Giger, and Harald Gall, “Supporting Continuous Integration by Mashing-Up Software Quality Information”
[6] P. M. Duvall, S. Matyas, and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, 2007.

Appendix - Plan of Actions

Actions	**Dec**	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**
Study about the continuous integration tools	▨								
Learn CI integration approaches	▨								
Study on restful web services	▨								
Use API of CI tools		▨							
Learn web service composition		▨							
Learn software metrics		▨	▨						
Learn prevailing software metrics			▨						
Create a software metrics				▨					
Data extraction from CI tools				▨					
Learn pipelining technology				▨					
Binding data to pipeline					▨				
Implement software metrics as a web service					▨	▨			
Create default views							▨		
integration							▨	▨	
Testing and re assessing the works								▨	
Reporting, Preparing presentation and demonstration								▨	▨