MÄLARDALEN UNIVERSITY

Evolvability Analysis Method for Open Source Software Systems

Author:
Muhammad Aufeef Chauhan
M.Sc. Software Engineering
School of Innovation, Design and Engineering
Malardalen University, Västerås, Sweden.
mcn10003@student.mdh.se

Supervisors:
Ms. Hongyu Pei Breivold, Industrial Software Systems
ABB Corporate Research, Västerås, Sweden.
hongyu.pei-breivold@se.abb.com

Dr. Ali Babar
IT University of Copenhagen, Denmark.
malibaba@itu.dk

Prof. Ivica Crnkovic, School of Innovation, Design and Engineering
Malardalen University, Västerås, Sweden.
ivica.crnkovic@mdh.se

Acknowledgements

I am very grateful to Prof. Ivica Crnkovic at Malardalen University and Dr. Muhammar Ali Babar at IT University of Copenhagen for their guidance to design the research plan, supervision of my thesis as well as my overall research work during my study at Malardalen University and for their generous support in terms of time and knowledge.

I am also very thankful to my supervisor, Ms. Hongyu Pei Breivold for her overwhelming guidance and time to advise me to conduct the research and laying out the draft of this report. I am also very thankful to her for her contributions to my other research courses.

I also acknowledge the EURECA project to provide funding for my study at Malardalen University and support of EURECA project's coordinator Prof. Sasikumar Punnekkat.

At the end I would like to thank Prof. Philip Johnson at University of Hawaii for providing source code repositories of the old releases and to make older version of the project's website available for analysis.

Muhammad Aufeef Chauhan
February, 2011.

Abstract

Software systems evolve over the life span to accommodate changes in order to meet technical and business requirements. Evolution of open source software (OSS) is challenging because of involvement from a large number of independent teams and developers who make modifications in the systems according to their own requirements. It is required to evaluate these changes as these are being incorporated into the system against the long term evolvability objectives. This paper presents the analysis of the Hackystat, an OSS framework; against analyzability, changeability, extensibility, testability domain specific quality attributes. The analysis of the processes used during the development of the OSS systems is also discussed. On the basis of the analysis and the early research conducted to evaluate software evolvability, an evolvability analysis method for OSS evolution is presented in this report. Guidelines of the model suggest that the requirements identification and analysis, identification of the system components that are to be affected as a result of the change, identification and prioritization of the potential solutions, evaluation of the potential solutions with respect to evolvability characteristics, use of test driven development and automated build tools are the important steps that should be performed to evaluate system changes. Evolvability analysis model also suggests that the team which is responsible to for system overall architecture (project control group) should also evaluate changes submitted by different teams. A case study to modify a service oriented architecture bases system into software as a service cloud model following the guidelines of evolvability analysis model is also presented.

Table of Contents

List of Tables and Figures

Introduction

Software evolution is referred as the ability of a software system to incorporate changes over its life span. It is defined as "*a process of progressive change in the attributes of the evolving entity or that of one or more of its constituent elements*" [35]. Changes to accommodate technology enhancements is mandatory to increase the life of the software systems whereas business related changes help software systems to meet the changing requirements of the customers and target business domain of the software systems. The class of the software systems that are free to modify, use and redistribute is referred as open source software (OSS) [36]. The evolution of OSS systems is challenging because the development of such systems is not only associated with a large number of independent teams but also different development processes.

The research work presented in this section was conducted to identify the process by which evolvability of OSS can be analyzed. It is quite challenging to keep the development on the right track to make sure that it is being evolved in the right direction. Research objective involve the identification of the evolvability characteristics of OSS, defining guidelines for evolvability analysis process that can be used to keep the changes on the right track and to evaluate the effectiveness of the proposed guidelines for the migration of OSS to more reliable software as a service models. One of the sub goal was also to identify the steps that should be taken to migrate service oriented systems into software as a service model.

This research work is an extension to the research performed by Breivold et al. to investigate the software evolution of industrial automation systems [1, 2, 3]. The architecture evolvability analysis method presented in their work to systematically analyze evolvability is evaluated against selected OSS system and tailored version is provided that can be adopted to meet the evolvability requirements of OSS systems. Proposed guidelines for the tailored version of evolvability analysis method begin with identification and analysis of the requirements. In next step, components are identified that can be effected as a result of the new requirements. Potential solutions are also analyzed at this stage. In last stages, potential solutions are evaluated against evolvability characteristics (analyzability, architectural integrity, extensibility, modifiability, portability etc) and solution that best suits the evaluation criteria is selected. Tailored version of evolvability analysis method also provides guidelines for the members of OSS system's change control group. These guidelines suggest the use of build tools, test driven development and evaluation of the commit requests for the main source code repository against evolvability characteristics. A case study for Hackystat migration to software as a service platform suggests the scalability and portability are the key characteristics of such systems.

The research work that is presented in this thesis is conducted in multiple steps. In first step, an OSS system the Hackystat [5] is investigated to analyze that how evolvability characteristics are addressed during the evolution of OSS systems. Analysability, changeability, portability and

testability are most important evolvability characteristics. The evolvability characteristics are significantly affected by the modularity and complexity of the systems. In later stage, the evolution of the open source software is analyzed against the tailored version of evolvability analysis method. An OSS system, the Hackystat framework is selected to investigate evolvability in a systematic manner from two perspectives: (i) the systematic evaluation of OSS evolution while focusing on Heckystat in particular and OSS systems in general (ii) evolvability analysis using the same evaluation method for the evolution of service oriented systems into cloud aware software systems.

The organization of rest of the thesis is as follows. First section presents research background and research objectives. A brief overview of the research results and research methodology is also discussed in this section. Section 2 elaborates the evolvability analysis of Hackystat software. This sections begins with an overview of Hackystat framework. Then metrics for modularity and complexity are presented that are computed against different releases of the framework. This follows by a section on evaluation of the evolvability characteristics over different releases and their interpretation in terms of modularity and complexity of the system. Other characteristics like architectural integrity, testability and complexity of interfaces are also addressed. The role of quality assessment, testing frameworks, automated build tools and source code repositories is highlighted as well. Last part of the section 2 provides the guidelines for evolvability analysis following by summary of the section. Section 3 presents a case study that was conducted to transform Hackystat framework into software as a service cloud. This section begins with an overview of the related technologies. Proceeding section provides the application of evolvability analysis method on software as a service migration. Third part provides the implementation and architecture overview of the modified version. Part 4 and 5 presents the future work and conclusion respectively.

1. Research Background, Objectives and Overview of Results

The ability of a software system to accommodate stakeholders' requirements in referred as software evolution [35]. OSS systems are evolved after repeated modifications and results in increasing complexity. If the systems are not designed to easily accommodate changes, the complexity may lead to huge modification cost. A challenging aspect of the evolution of open source software (OSS) systems is that such systems are evolved without strict development processes and with involvement of independent individuals or teams. Breivold et al. have identified the characteristics necessary for evolution and how evolvability can be addressed in a systematic manner in context of industrial automation systems [1, 2, 3]. The research presented in this thesis is conducted as a continuation of their research in context of OSS software systems.

In this thesis two case studies are presented as a study of OSS evolution. In first case study, evolvability characteristics and evolvability analysis method identified in [1, 2, 3] is analyzed in context of OSS systems. Original evolvability analysis model presented by Breivold et al. is also modified to analyze the evolvability of OSS systems. This case study is based on the analysis of an OSS system, the Hackystat [5] along with published literature on OSS evolution [10]. Hackystat is chosen for analysis because it is under development since 2001 and evolved from a server side web application into an application built on service oriented architecture. Selection of Hackystat has provided the opportunity to study evolution not only in context of traditional software architectures but also how a successful transformation can be achieved for modern technology paradigms like service oriented architectures. The second case study is an experiment conducted on the service oriented version of the Hackystat to modify it to meet the requirements of the software as a service model. During this experimental transformation, evolvability analysis guidelines defined for OSS system are also followed to verify for validity.

1.1 Research Objectives

The main objective of the research presented in this thesis is to analyze the evolvability characteristics and evolvability analysis method presented by Breivold et al. in context of the OSS system. One of the main objectives of the research is also to analyze the transformation of a service oriented systems into a more reliable and scalable software as a service systems. To analyze the OSS evolvability, it is also important to identify and analyze the evolvabiliy characteristics of such systems. Following research objectives are derived to address the main theme of study:

- ***What are different quality attributes (evolvability characteristics) that should be present in an OSS system for its evolution?***

- ***How evolvability of OSS systems can be analyzed by development teams as well as team responsible to keep the overall development of the system on right track (OSS control group)?***
- ***How effectively evolvability analysis method can be applied on software evolution for emerging paradigms like software as a service model?***
- ***How service oriented open source systems can be evolved into more robust and reliable software as a service model based systems?***

1.2 Overview of Research Results

In order to evaluate the evolvability characteristics and to address the analysis of OSS systems different releases as well as the development process of the Hackystat is investigated. Information from the published literature is also incorporated to have a more in depth view of the OSS development process. After the analysis it is verified that analyzability, portability, extensibility and testability play their role in the evolution of OSS system in the similar fashion as they do in case of proprietary software systems. Domain specific quality attributes are also critical for the successful evolution of such systems. Analysis of the Hackystat and the published literature has suggested that analysis of the OSS software by the member of the control group is equally important as it is to be performed by the members of the development teams.

Results of the investigation of Hackystat for its migration to software as a service system have shown that all the components of the systems should be carefully analyzed for scalability. It is also found that in order to take use of the cloud infrastructure; system components that are responsible for persistence management may need to be re-factored to make use of external storage clouds. Moreover, some new components may also need to be introduced to provide the consolidated view of the system to the outside world.

1.3 Research Methodology

The research process was initiated with the main goal of analyzing the evolvability of the OSS system. The main research objective was addressed at two levels. At first stage, analysis was conducted on a selected open source software for identification of evolvability characteristics and to perform evolvability analysis. At second stage the identified evolvability analysis method was applied on a case study to verify that the selected method resulted in successful evolution. To keep the research on the right track, research objectives were defined as mentioned in section 1.1. The second step was about the selection of the target system and the data sources for analysis. Heckystat was selected because of its long evolvability history of ten years, as it is being developed since 2001. As it is open source software so the code base against different releases was available for analysis. Information was extracted from the data on the basis of research objectives.

2. Evolvability Analysis of the Hackystat

This section provides the evolvability analysis of the Hackystat software over its different releases. Metrics for modularity and complexity are calculated from source of the selected releases and then the data from these metrics are interpreted in terms of the evolvability characteristics. Some process related aspects like change evaluation process, role of testing frameworks, importance of source code and documentation repositories as well as the significance of feedback to improve quality is also addressed. This section also provides the evolvability analysis guidelines for development teams and member of the OSS control group.

2.1. Hackystat's Overview:

Hakystat is an OSS used to collect the process and product related data about development of software projects [4]. Hackystat is being developed since 2001 and has undergone 8 releases since then. From architectural perspective, different release of the Hackystat framework can be categories into two categories. From release 1 till release 7, the software was built on the thin client server architecture. In release 8, it is transformed into service oriented architecture. Figure 1 shows the high level architecture of the Hackystat release 8 [5].

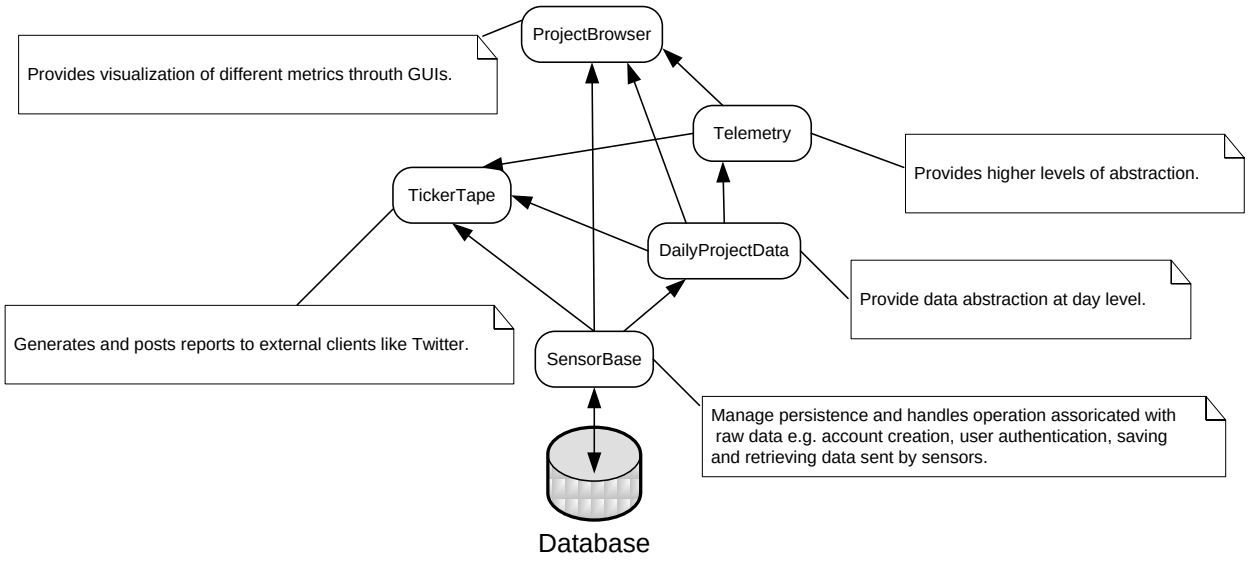

Figure 1: REST Components of Hackystat

The components of the hackystat framework can be classified into two categories: data collection components and data processing components [5]. Data collection components are client side components, also referred as plugins that can be installed with the target data sources like integrated development environments (IDE), source code repositories, XML data sources and word processing suite. The plugins are also named as sensors or sensor shells.

Other set of components is server side components that reside on server and are responsible to perform the business logic. This set of components receives the data from the sensors, persist this data and present it to the user at different levels of granularity through a web based application.

In Hackystat release 7 and earlier; business logic is handled at server side with more than one independent server side applications which perform processing on the data. Sensors act a client of the simple object access protocol (SOAP) [6] based web services and sends data to the server using SOAP protocol. Applications at the server side present this data in form of different process and product metrics.

In release 8, the framework is transformed into a service oriented system based on representational state transfer (REST) [7] principles. Different data input sensors use to work in the same way except that the data is transmitted using REST protocols instead of SOAP. However, server side architecture is completely changed. Different server side components are transformed into REST based services and interfaces of the services are exposed through REST application programmable interfaces (APIs). These components interact with each other through the APIs and act as a processing pipeline.

SensorBase is the first service and acts as the root node of the processing pipeline. This service receives the information from data sensors and sensor shells. It is responsible to persist the data into the database and acts as an access point for the data retrieval. Whenever other services need to have access to the persisted data, it is always done through SensorBase. Its REST APIs can be accessed by not only other services of the Hackystat but also by the external client applications that want to access data.

There are also different high level services responsible to present data at higher levels of abstraction. One of such services is DailyProjectData. It provides abstraction of the data associated with a single project for one day. An example of the product metrics which abstraction is provided by this service is Development Time Metric. This presents the number of minutes in a single working day for which the developers were actively interacting with the source code of the specific project through integrated development environment. Telemetry is another service that provides the abstraction of raw data at higher levels of abstraction than a single day. Finally the data is presented to the users through a web application component called Project Browser. Hackystat also has some client services that are used to post data to external clients. One of the client services is the TickerTape. It interacts with SensorBase and daily project data service to get information for the changes made in the source code of the specific project and generates a status report. This report can be sent to different external applications like Nabaztag Rabbit [8] and Twitter [9].

The different services of the framework act as a pipeline. For example if the ProjectBrowser have to fetch some data from database it sends request to the Telemetry service. Telemetry

service then calls the corresponding interface of the DailyProjectData service and it finally fetches data using interface of SensorBase. Fetched information is sent back to the ProjectBrowser through same hierarchy. This pipeline based approach not only provide loose coupling by separating the business logic into separate services but also provides interfaces to clients to access data at different levels of granularity.

2.2. Metrics to Analyze Evolvability Characteristics and Architecture over Releases:

In order to analyze the evolvability, source code from the different releases of the Hackystat is investigated. The final version of the source code is selected against releases for analysis. Only those releases that are associated with some major enhancement in the system are selected for computing modularity as well as complexity metrics. Source code from release number 2, 6, 7 and 8 is selected for analysis.

Modularity is referred as a property of a piece of software when it consists of distinct and logically cohesive units; and presents it functionality to the outside world through well defined interfaces [14]. Breivold et al. have identified that modularity of OSS systems is measured in terms of number of sub systems, number of modules and number of source code files [10]. At an abstract level when modularity is computed in terms of number of sub systems in the OSS, each sub system is regarded as a logically cohesive unit of functionality and is referred as module. If modularity is analyzed in more detailed level, each module or in each package (in case of java language) is referred as a module. In object oriented programming languages, each class encapsulated properties and behavior, so classes are also referred as modules. In the analysis of the Hackystat, modularity at all three levels is computed to see its impact on quality attributes.

Functional complexity of the system is defined in terms of the complexity of the logic contained in the system [10, 37]. It is also computed at different levels in terms of complexity of classes, complexity of modules and complexity of sub systems.

2.2.1. Metrics for Modularity

When modularity is measured in terms of number of files in the system, total number of source code files in a release is referred a number of modules. Hackystat framework is implemented in Java EE framework, so each java source file is considered as an executable file. Table 1 shows the modularity of different releases of the Hackystat in terms of number of files in the system. First column of the table shows the release number and second columns shows the total number of files in the corresponding release. It is clear from the table that as the system evolves; degree of modularity also tends to increase.

Table 1: Modularity in Terms of Number of Files

Release Number	Total Number of Files
2	206
6	503
7	922
8	861

As the Hackystat is implemented in Java EE, so a package is considered as a module. Total number of packages in a release is referred as the degree of modularity of the system in terms of number of modules. Table 2 lists the number of packages against each release of the system. It is clear from the table that the degree of modularity in terms of number of packages is also increasing in every new release of the system.

Table 2: Modularity in Terms of Packages

Release Number	Total Number of Packages
2	22
6	71
7	185
8	171

It is explained in Section 2.1 that Hackystat architecture is changed in release 8. So, the sub systems are computed differently in versions till release 7 and in release 8. Till release 7, different sensors and subs systems are implemented as separate source code projects. So, different project are treated as sub systems. However, in release 8, different components of the system are implemented as REST services. So, each REST service component is treated as a separate project although sensors continue to be implemented as separate projects and are computed in the same manner. Table 3 shows the number sub system against releases.

Table 3: Modularity in Terms of Number of Sub Systems

Release Number	Total Number of Sub Systems
2	4
6	4
7	40
8	10

Data in table 1, 2 and 3 shows that modularity in terms of number of files, packages and sub system tend to increase till release 7. In release 8, there is a small decrease in degree of modularity in terms of number of files and number of packages but number of sub systems has decreased significantly. This is result of system re-factoring that is performed in release 8.

2.2.2. Metrics for Complexity

Metrics for complexity are also calculated at different levels: at class level, at module level and at sub-system level. First, complexity is computed at class level. Then the average complexity of classes in a module and sub system is used to determine the complexity at higher levels of granularity. The metric used to computer complexity at class level is weighted method per class (WMC) [11, 12]. There are two variants of this metric. One variant determines the complexity as the sum of Cyclomatic Complexities of all the methods within a class. Other one determines the complexity simply as the total number of methods present in the class. As the Hackystat is written in java programming language, so the simple version of the weighted method per class is used and the complexity is computed in terms of total number of methods present in a class. Classes are marked with the complexity of scale low, medium and high. Research has shown that average number of functions in class ranges between 5 to 10 [13]. The classes with less than or equal to 10 methods are assumed of low complexity. The higher levels of complexity are marked relative to this base and classes between 11 and 20 methods are considered of medium complexity whereas with greater than 20 methods are assumed of high complexity. Table 4 elaborated the classification criteria.

Table 4: Classification of WMC Metric

Complexity	**No of functions**
1: Low	<=10
2: Medium	> 10 and <=20

3: High	>20

The complexity of modules and sub systems is computed in terms of the average of complexities of classes in the packages and sub systems respectively. Category with the largest occurring frequency is assigned as the complexity of the corresponding release. Table 5 summarizes the complexity of different classes with respect to releases. To compute the complexity of all the classes in a release, complexity of each class in terms of number of functions was calculated. Then the frequency of the low, medium and high complexity classes is calculated and the complexity of a release is associated with most frequent complexity class.

Table 5: Complexity of Classes

Release Number	Complexity of Classes
2	Low
6	Low
7	Low
8	Low

It is clear from the table that the complexity of the system's classes is low. Table 6 and 7 shows the complexity of Hackystat in at module and sub system level. The complexity of the modules in the release is calculated in terms of complexity of the classes in a release. The complexity of a module is associated with the most frequently occurring complexity type in a module and most frequently occurring complexity type of modules determined the complexity of the release. Complexity of the subsystems is also calculated in the same way.

Table 6: Complexity of Modules

Release Number	Complexity of Modules
2	Medium
6	Low
7	Low
8	Low

Table 7: Complexity of Sub Systems

Release Number	Complexity of Sub Systems
2	Medium
6	Low
7	Low
8	Low

Table 6 and 7 shows that the complexity of modules and sub systems was medium in release 2 but it also decreased in proceeding releases because of as a result of increase in number of modules and number of sub system.

2.2.3 High Level Architecture over Releases

Hackystat components are classified into two classes, server side components that perform the business logic and client side components that are used to collect data to calculate metrics. Over different releases of the framework, number of both server side and client side components tend to increase. In version till release 7, different server side components interact with each other through java API interfaces. As the number of server side components tend to increase, the communication between components becomes complex. Different components also required residing on the same physical machine. This resulted in decreased performance as more computing resources were required. It also increased time to set up development and testing infrastructure. Figure 2 shows the high level architecture in release 2, 6 and 7. It is clear from the diagram that number of subsystems increased with each new release. In release 8, the framework was rewritten and transformed into a REST based service oriented architecture as described in section 2.1.

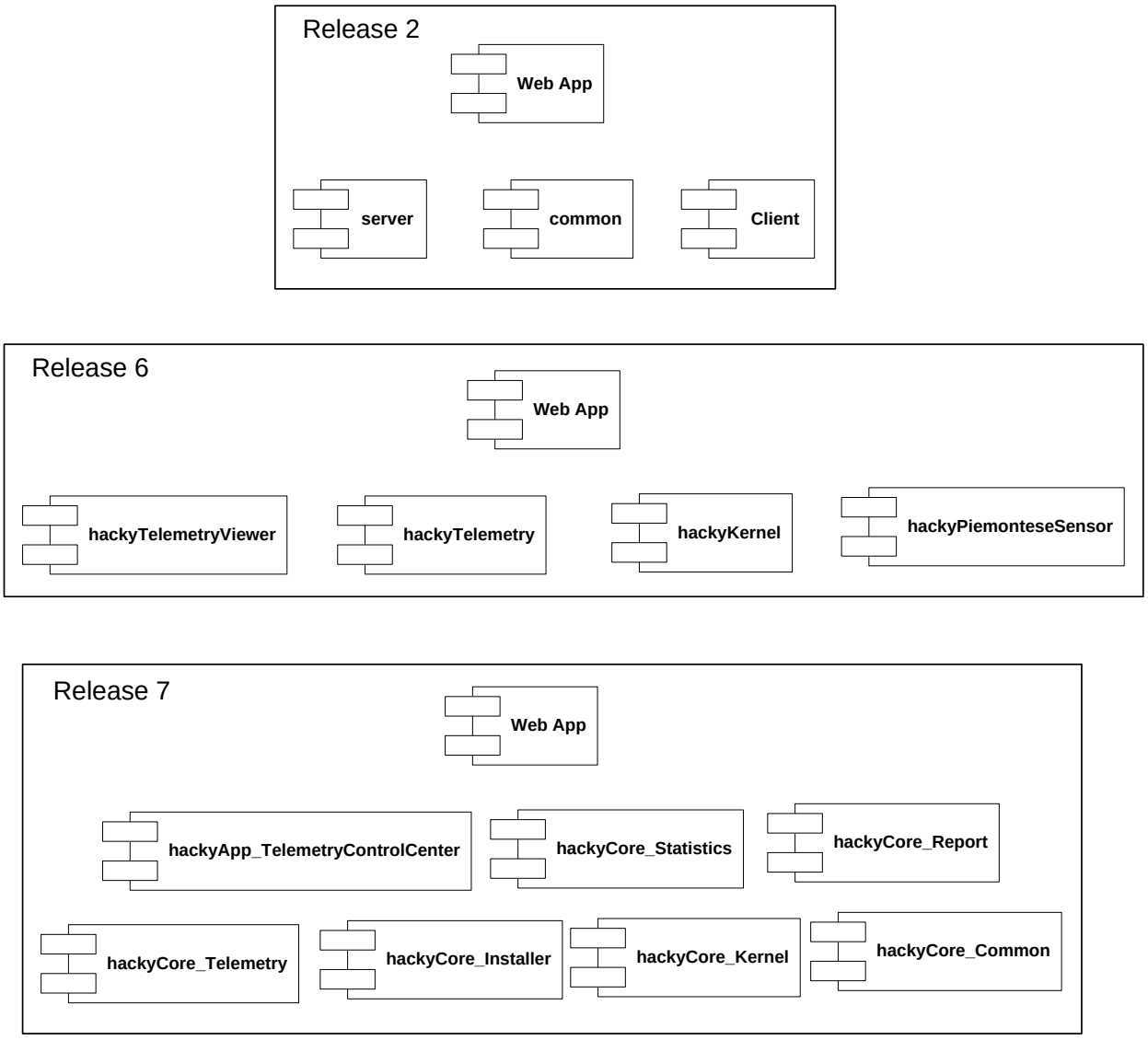

Figure 2: Hackystat architectire over releases

2.3. Evaluation of Evolvability Characteristics:

Breivold et al. has identified important characteristics that are necessary for the evolution of a software system [2]. The evolvability characteristics are: analyzability, changeability, extensibility, portability, testability and quality attributes that are related with a particular domain being addressed by the software system. In this research, different releases of the Hackystat are evaluated to determine the role of these characteristics in the evolution of OSS system. The evolvability characteristics are evaluated in terms of different measuring attributes. Table 8 lists the measuring attributes used to evaluate the corresponding evolvability characteristic identified by Breivold et al. in their study of industrial automation systems [3].

Table 8: Measuring Attributes with Corresponding Evolvability Characteristic Identified in [3]

Evolvability Characteristics	Measuring Attributes
Analyzability	Modularity, complexity and documentation
Architectural Integrity	Architectural documentation
Changeability	Modularity, complexity, coupling, change impact, encapsulation and code reuse.
Extensibility	Modularity, coupling, encapsulation and change impact.
Portability	Techniques to incorporate features that support adaption to multiple environments.
Testability	Modularity and complexity.
Domain Specific Attributes	Attributes related to specific business domain.

It is clear from Table 8 that modularity and complexity are key measuring attributes of many evolvability characteristics. ***Analyzability*** is the first evolvability characteristic mentioned in the table 8 and is defined as "the capability of the software system to enable the identification of influenced parts due to change stimuli." [2]. Modularity is the key attribute to increase the evolvability of a software systems because it is easy to analyze loosely coupled modules of a software system as compared to the system in which different components are tightly coupled and highly dependent on each other. Complexity of the software is also an important factor because less complex systems are easy to analyze.

"The capability of the software system to enable a specified modification to be implemented and avoid unexpected effects" is referred as ***changeability*** [2]. Modularity also plays a vital role in achieving changeability because it allows making modification in one module and keeping other intact as well as it reduces the possibility of unexpected behavior as a result of modification [14]. Similarly, system with manageable complexity is more likely to accommodate changes with manageable cost as compared to the system with high complexity.

"The capability of the software system to enable the implementation of extensions to expand or enhance the system with new capabilities and features with minimal impact to the existing system" is referred as ***extensibility*** [2]. As modularity supports separation of concerns and provide the extension points for the addition of new modules on the principles of coupling and cohesion [15]; thus make it a key requirement for extensible software systems. It is hard to identify the points of change in systems with higher dependencies between components.

"The capability of the software system to be transferred from one environment to another" is identified as ***portability*** [2]. Modularity ensures that only platform independent interfaces are exposed to external world and abstracts the platform specific implementation. Modularized systems are easy to modify for the portability on different platform because the modification done for this purpose will be hidden from other modules and do not require modifications in the clients' components. The advantages of the modularity make it suitable to achieve portability.

The verification and validation of the modified piece of software is covered under ***testability*** [16]. Modularity reduces the testing effort. The testing scope is only confined to the changed module because of intact interfaces that results in reduced testing effort. Along with above specific characteristics there are some additional domain specific quality attributes that are to be achieved in order to make software more evolvable. Similarly, less complex systems are more easily testable.

Following sections describes the analysis of the different evolvability characteristics including analyzability, architectural integrity, changeability, extensibility, testability and domain specific characteristics is presented in Table 8 along with measuring. Following sections describe how these characteristics are addressed in different releases of Hackystat.

2.3.1. Impact of Modularity and Complexity

The different releases of Hackystat were analyzed to have a look at the role of modularity in the evolution of the OSS system. The modularity of OSS systems is addresses at different levels: file, module and subsystem; because each one encapsulates a part of system functionality and provides certain level of abstraction. The hackystat software is implemented in Java EE technologies so in the analysis of the source code repositories, a java file is treated as an executable file. The different java packages are treated as modules and parts of the system that implement a specific feature for example handle telemetry analysis [4] is treated as a subsystem.

Table 1, 2 and 3 shows the number of modules in key releases of the hackystat software. It shows that number of modules is increasing from release 2 to release 7 at file, module and subsystem level. If numbers of files are considered as a measurement of modularity; there are four times more files in release 7 as compared to release 2. Similarly number of modules and subsystems have increased eight and ten times respectively. However, statistics from release 8 show a slight decrease in number of files and number of modules, whereas number of subsystems has reduces significantly [4]. In release 8, complete system is rewritten and transformed from a web based

application into a service oriented system to meet some domain specific requirements like visualization of product and process metrics at different levels along with scalability of the software.

Table 5, 6 and 7 shows the summary of the functional complexity at different levels of granularity. It is clear from the statistics that complexity is manageable in all the releases of the hackystat. In release 2, modules and subsystem are of medium complexity; but complexity of modules and subsystems is also low in later releases as in case of complexity of classes.

Statistical analysis of the different releases of the Hackystat OSS framework has confirmed that modularity and complexity play a vital role in improving the evolvability of the system because increased modularity and reduced complexity ensures the analyzability, changeability, extensibility and testability. Hence it can be deduced that the OSS systems with modularized architecture and manageable complexity as more probable to achieve high degree of evolvability characteristics as compared to the systems those are less modularized and complex.

2.3.2. Architectural Integrity

Research on proprietary software suggests that the architectural documentation as a primary attribute to ensure architectural integrity [3]. However, the analysis of the hackystat and other studies [10] suggest that OSS system often don't have detailed design artifacts. The study of the Hackystat has suggested that high level documentation for architecture as well as other design artifacts are available at the project's website but detailed design artifacts are not available [5]. Hackstat website is hosted on google project hosting system [17]. This website contains the documentation as well as source code of different subsystems and components of the Hackystat. Hackystat documentation is accessible in form of web tutorials, video tutorials and online wiki. Like in proprietary software systems, core architecture team is responsible to make major design decision as it was made to transform Hackystat into service oriented system [4].

The development of OSS systems involves large number of independent teams and individuals who participate in the development activities to modify software according to their requirements and usually don't know about development activities being done by other teams [38]. This highlights the need for a centralized place where different development teams can share their ideas and documents about their development activities. Then, the team of core architects can decide about which of these implementations can be incorporated into original software and make available the relevant documentation. A wiki based solution for this purpose can be helpful.

2.3.3. Testability

Testability is an important quality attribute to maintain the quality of software systems during the evolution. For OSS system, it is hard to ensure that each development team commits code to the base repository after proper testing. Test driven development (TDD) has proved to be the

significant development methodology to improve software quality around 40% [18]. TDD can play an important role in improving the quality of an OSS system. With the help of some automated tool like Zorro, development activities of different team can be monitored to ensure that they follow TDD methodology in a desired manner. In different releases of the Hackystat software, the modules are accompanied by corresponding test classes. In different releases of the Hackystat, source code modules are associated with test classes. Investigation of the existing literature as well as the development process of the selected OSS has confirmed that TDD can be significantly used to improve quality of OSS systems and in turn can play a vital role in long term evolution of the systems [18].

2.3.4. Complexity of Interfaces

Reusability of the code has positive effect on the evolvabilty of the software system [39]. Reusable components not only save the development effort but also improve the quality of the software system as reusable components are not required to be tested again and again over different releases. Different components of the system interact with each other through their interfaces. For a component to be reusable its interfaces should expose underlying functionality by simple and standard interfaces.

Up till release 7, interfaces of the different components of Hackystat system are exposed using Java interfaces and public functions. Different sensors and sensor shells sends process and product data through web service interfaces using SOAP protocol. Different server side components were accessing information from each other through java interfaces. For developers to set up a development environment, source code from all the components was compulsory to be downloaded on development machines to make components compile. Development activity can only be initiated only after source code from all the components is configured. This results in more effort in development environment setup. Apart from this initial effort, whenever there was modification in interfaces, the developers need to get the latest code to avoid anomalies in later stages. For a running instance of the software, this type of tight coupling also forced different components of the framework to be deployed on single machine. This dependency played an important role that lead to the re-factoring of the framework as it was getting very difficult to incorporate new features into the system [4].

In release 8, system architecture is modified and different components of the system are implemented as REST web services. Interfaces to the REST services are exposed in terms of HTTP GET, PUT, POST and DELETE methods. With this type of implementation, developers can work on the modules independently because there is no compile time or run time binding between components. Components access a particular interface only when they need to access its functionality. Other than development, simple REST interfaces also added feature in the framework to deploy components on different machines. It is the feature of the REST architecture that different components can interact with each other through HTTP protocol. In addition to modularity and deployment features, REST interfaces also make client modules

language and platform neutral. REST interfaces of the Hackystat components also provide an opportunity to write client programs to access functionality of each component and have access to the data at different levels of abstraction. It was not possible in previous versions. It also increases the possibility of code reuse. Hence, it is evident from the analysis that reduced complexity of interfaces increases the evolvability of the OSS systems. Requirements for the modifications are initiated by the project development control group lead by Hackystat project lead.

2.3.5. Domain Specific Characteristics

Other than general characteristics of evolution, every OSS system has a set of key quality attributes that are associated with the core functionality of the system like efficiency, security, reliability and scalability. These key quality attributes are covered in domain specific evolvability characteristics. Analysis of the different releases of the Hackystat has confirmed that the domain specific quality attributes also play a vital role in the long term evolution of the system. Like every server side application that is supposed to handle huge volume of requests from different clients, scalability is key characteristic of the Hackystat system. Hakystat system is dealing with process and product metrics; so, it is also important for this kind of system to provide data abstraction at multiple levels of granularity. Development of the Hackystat framework was initiated with the objective that system had been able to accommodate new requirements as they were generated. At early stages of the system development, domain specific evolvability characteristics were not considered. This brings the system at a point where it was impossible to meet the performance requirements. As a result of this bottleneck, complete system was redeveloped from scratch in release 8 [4]. During the redevelopment phase, scalability of the system to meet efficiency requirements and different levels of abstraction of development metrics were of primary importance. This resulted in the REST based service oriented architecture of the system. Hence, the analysis has confirmed that ignoring domain specific characteristics also have a negative impact on the evolution of the OSS systems.

2.4. Evolvability Analysis Process of Open Source Software Systems

The development process of OSS systems is characterized by the involvement of large number of independent teams and developers. The independent teams and developers make changes into the system according to their specific requirements. In OSS, the requirements may be initiated as a result of business objectives, by stakeholders and in some cases by the development control group. For the control group of the OSS, it is important to make sure that modification and enhancements that are to be incorporated in base source code repository is in line with existing architecture and are of high quality. Evolvability of OSS systems can be analyzed in terms of quality assessment using feedback, using automated testing frameworks, analyzing role of automated build tools and analyzing the influence of code as well as documentation repositories.

2.4.1 Quality Assessment using Feedback

Bouktif et al. have presented an approach that is for remote and continuous analysis of OSS systems [19]. The goal of this approach is to provide feedback driven communication service after analyzing the available data sources. For OSS systems, information is generally exchanged in form of email. The emails used for communication between development teams as well as between developers and members of the control group is an important source of information. Source code repositories and commit log files are also important sources of information. Metrics for growth, complexity and quality are computed after analyzing different sources. This information should be presented to developers by using some kind of dashboard service like a Wiki based solution to mitigate software degradation and risks.

2.4.2 Use of Automated Testing Frameworks

Testing is very important to ensure the quality of the software system. For proprietary software, testing is usually performed at the organization that develops and maintains that particular system. To ensure the quality of OSS system, testing also needs to be done by or under the supervision of control group. It is not possible to perform thorough testing on changes committed by different development teams. So, it should be the part of development process of the OSS system to use automated testing framework like JUnit [20]. Every commit request of the source code should include the corresponding test classes as well. It would help the control group to test and validate the changes before making that a part of the main code repository. Research on test driven development has also shown the significant improvement and claims from 40% to 80% improvement in quality [18].

2.4.3 Role of Automated Build Tools

Use of automated build tools like Apache Ant is also helpful to drive process described in build files and to ensure the quality of code committed by different development teams [21]. Every request for code commit in the main code repository should contain the build script so that the code can be build and tested with the help of testing framework. As build tools can also be used to drive development processes; the control group can provide guidelines for the build scripts and to certain extend control the development process of the individual team of OSS systems.

2.4.4 Influence of Source Code and Documentation Repositories

The source code repositories in which code is retained acts as exogenous factor. These repositories play their role as a differentiating factor in the evolution of OSS [22]. A study by Beecher et al. investigates the large number of repositories and results of the study strengthened the statement about the role of repositories in the evolution. Different releases of Hackystat have also confirmed this claim. Throughout the development history of the framework, its code is being hosted on source code repositories like CVS [23] and SVN [24], and can be accessed from remote location. Documentation about the different artifacts of the system is also available through project website. For the latest release of the Hackystat, project code and website is

hosted at google project hosting solution [5, 17]. All these findings have confirmed the significance of the source code and documentation repositories for the evolution of OSS system.

2.5. Evolvability Analysis Method for OSS

In this section, an evolvability analysis method is proposed that can be used to access evolvability of the OSS system in a systematic manner. This model is based on the discussion made in section 2.4 and tailored version of steps proposes in study by Breivold et al. [1]. The proposed method consists of two parts. Part one contains the guidelines for independent teams and developers working on the OSS system. Second part lists the guidelines for the control group of the particular software to ensure the quality of the OSS in the long term and keep the enhancements on the right track. The guidelines for evolvability analysis of independent teams and developers are as follows:

- *Requirements identification and analysis:* Analyze the impact of proposed change on the overall software and to its external client components. This involves identification, analysis and prioritization of the requirements.

- *Evaluation of change impact, potential solutions and test cases:* Once the key set of requirements is identified, next step is to evaluate the impact of the change on the overall system architecture and prepare the proposed solution. This involves the identification of the architectural constructs if available, identification of the system components that need to be modified to accommodate change, identification and prioritization of the potential solutions, and defining the test cases.

- *Evaluation of potential solutions against evolvability characteristics:* The potential solutions are evaluated against the selected set of evolvability characteristics described in Table 8. The solution that best suits the evaluation criteria is selected.

- *Testing:* Write the test cases using automated testing framework for example JUnit [18, 20] so that test cases can be executed by member of the control group.

- *Build process:* Write the build scripts by using some build toll like Ant [21] or Maven [25] to compile code and run test cases on the modified parts of the OSS system.

The guidelines for the members of the control group to analyze and ensure long term evolvability of the system are given below:

- Enforce the use of the build tool using the specified script guidelines so that only error free code is checked in the branches of the core repositories.

- Enforce the use of testing framework to test the modified part of the code. Classes for the test code should be submitted along with the build script to make the modified code testable by members of the control group.

- Identify the components and modules that each commit can change with the help of build script and testing framework.
- Evaluate the changes in terms of the evolvability characteristics of the system. Also mark the degree of eligibility in quantitative measures.
- If a change request fulfills the evaluation, add it to the candidate list of the commit requests eligible to be incorporated into the base source code repository.
- As there may be more than one development teams that made modifications on the overlapping portion of the code; so, the commit request with most high degree of eligibility should be made part of the base code repository.
- Information about the accepted and rejected commit requests along with evaluation criteria should be made available on the OSS system Wiki.

2.6. Summary and Concluding Remarks

This part presented the analysis of the different releases of the hackystat software against evolvability characteristics. Modularity and complexity are key measuring attributes to analyze many of the evolvability characteristics including: analyzability, changeability, extensibility and testability. Through study of different releases of the Hackystat framework has confirmed that high modularity and low complexity are key factors for long terms evolution of the OSS systems.

It is a fact that architectural documentation is considered an important factor for the evolution of the software systems; but Hackystat, like many other OSS system continues to evolve successfully without the availability of detailed architecture documentation and other design artifacts. Project web site serves as a major source of information and since the beginning of the project, it has an associated website for information retrieval. Testing plays a vital role to ensure quality of the Hackystat and other OSS system and use of testing framework is a good way to ensure quality of such systems.

Evolvability analysis of the OSS systems for compliance with architecture and evolvability characteristics also needs to be performed by both the developers as well as members of the control group. Initially, independent development teams can perform this task to ensure that their development is in line with the overall system architecture. However, the analysis of the architecture evolvability by OSS development control group is of significant importance. The use of automated testing frameworks and automated build tool can be very helpful to govern the software development process as well as to verify the quality of code commit requests.

Following points summarize the steps of evolvability analysis method of OSS systems.

- Identification of the components and modules that are to be affected as a result of addition or modification of a change.
- Evaluation of the added or modified sections of the code from technical as well as business perspective.
- Evaluation of the test cases according to the standards of selected testing framework that are submitted by the development teams along with code related to actual functionality.
- Compare the evaluation results with other commit request affecting the overlapping sections of code.
- Selection of the commit request for evolvability analysis if it fulfills the selection criteria.

- Once a particular commit request fulfills the evaluation criteria, it should be investigated to see the impact of the changes in terms of evolvability characteristics. If a commit request satisfies the acceptable threshold level of the evolvability characteristic it can be make part of the main code repository.

3. Hackystat Migration to Software as a Service Cloud: A Case Study

This section presents the work that is conducted to transform the Hackystat software into a cloud aware OSS system. Migration was performed using the evolvability analysis methods described in the previous section to judge its effectiveness for new generation applications build on the principle of Service Oriented Architecture and cloud aware software systems. A brief overview of the related technologies is also provided in this section.

3.1. Technology Overview

Service oriented architecture (SOA) [26] is gaining momentum because of its inherent characteristics of modularity. As a result of modularized architecture, such systems are easy to maintain and scale up or down according to the requirements. Web services are an effective and easy solution to implement service oriented architecture [27]. On the basis of the technology, web services are broadly classified into two categories: i) services in which basic unit of communication is a message instead of operation, ii) services that provide the interface to operation through a set of well defined protocols such as HTTP. Second type of services is referred as REpresentational State Transfer (REST). Release 8 of the Hackystat framework is implemented following RESTful architecture standards [4]. We picked this release and investigated it as a case study to migrate REST based SOA systems into cloud aware platform. In this section, a brief overview of the RESTful web services and cloud computing environment is provided.

3.1.1. RESTful Web Services

RESTful web services are based on the concept of resource, its representation and its state [28]. Resource is an implementation of a functionality that is stored on the server and can be exposed to the external world. A representation of the resource is information about its state. A resource can have more than one representation. In REST services, a resource has two different types of states. Information about the resource is referred as a resource state and information about how the resource is accessed by its clients is referred as application state. State of the resource is handled by the server where the resource is hosted whereas application state is maintained by the clients of the resource. Resources are exposed to the clients with the help of URIs. Each resource has a unique URI representation. The resources always expose the stateless information to the clients. Communication between resources and clients is done through HTTP protocols.

In REST web services, communication between resources and their respective clients is performed using four http operation: GET, POST, PUT and DELETE. GET is used to retrieve the state of the resource, PUT and POST are used to create a new resource or its new state and DELETE removes an existing resource from the server.

3.1.2. Cloud Computing

Cloud computing has emerged as an attractive area of research because it provides the flexibility to scale up or scale down software and hardware infrastructure without huge upfront investments [6]. Cloud aware infrastructure should have three characteristics: ability to acquire transactional resource on demand, resource publication through a single provider, and mechanisms to bill users on the basis of resource utilization. As this research is focusing only on OSS systems, so only first two properties of the cloud computing environment are addressed in this case study.

From the ownership perspective, cloud computing infrastructure is classified into three categories: private clouds, public clouds and hybrid clouds. Private cloud is a collection of computing resources, storage resources and cloud technologies owned by an organization itself. The organization has control of all the resources and technologies; and itself is responsible for the maintenance of the infrastructure. Public cloud infrastructure is a collection of resources maintained by different organizations and the resources are offered to public users. A hybrid cloud is a category of cloud resources in which part of the infrastructure is maintained by the organization itself whereas it also acquires the services from public clouds. The different models of the cloud environments have its advantages and disadvantages. The advantage for maintaining the private cloud is that the respective organizations have control over all the resources. The disadvantage of such a system is that the respective organization not only has to invest in computing and storage resources but also on the related software and maintenance activities. The advantage of using the public cloud is that the organizations itself does not have to take care of infrastructure and operational activities. The disadvantage of utilizing services from public cloud is total dependency on the firm that is offering resources through public cloud. Security of the data that is stored on the public cloud can be another issue as some critical organizational data should be prevented from every time of accidental access. In such cases the tradeoff between public and private cloud is made and a hybrid approach is adopted. In such cases, organizations maintained their critical resources as private cloud and rely on public clouds for non critical business operations.

Taking advantage of the cloud infrastructure, organizations offer different types of services. Software as a Service (SaaS) infrastructure offers software applications to its customers. In this type of services, developers usually don't have any option to customize the applications. However, using the available customization option, users of the systems can modify the respective software according to their needs. Google email service [31] is an example of SaaS model. In Platform as a Service (PaaS) infrastructure, developers have access to a development platform through its APIs. These platforms support a specific set of programming languages. Google AppEngine [30] is an example of such platform. Infrastructure as a Service (IaaS) model offers infrastructure for computing and storage resources. This infrastructure is used to host applications. IaaS models often provide automatic support for scalability of computing and storage resources.

3.2. Application of Evolvability Analysis Method on SaaS Migration

Evolvability analysis of the Hackystat for its migration from a simple SOA architecture to SaaS model was conducted to evaluate the effectiveness of evolvablility analysis method presented in Section 2.5 for the evolution of the existing systems into SaaS models. One of the objective of this transformation was also to take use of the IaaS model and evaluate the steps that needs to be performed during this transformation. All the evolvability characteristics including analyzability, architectural integrity, changeability, portability, extensibility, testability and domain specific attributes like efficiency and presentation of development metrics on different levels of abstractions were considered during the migration.

3.2.1. Requirements Identification and Analysis

The migration activity was initiated as a result of the requirement to enhance Hackystat software as a SaaS model and with the additional capability to deploy this model on IaaS clouds. The identification of steps required to be taken for this migration was also an important stimuli for the migration activity. To achieve this high level business requirement, it needed to be analyzed and to be broken down in more concrete requirements that can be implemented. For this purpose, properties of SaaS clouds were investigated and following key characteristics had been identified.

- SaaS clouds have the capability to scale up or scale down the computing and storage resrouces on demand basis.
- This scalability is offered in a transparent way and users of the system access the services of the system in seamless fashion.

The target software system is expected to be migrated on a IaaS cloud , so the properties of such infrastructure were also investigated and the followings characteristics had been identified.

- IaaS clouds offer environments to host applications to utilize computing resources of the clouds.
- IaaS clouds offer storage solutions to utilize storage capacity of the clouds.

After analyzing the characteristics of the SaaS and IaaS clouds, following requirements for the system were identified.

R1: Different components of the systems should be able to be replicated to scale up or scale down the system according to performance requirements and should be deployable on computing resources offered by IaaS providers.

Activities for R1 include:

a. Investigation of the dependency between different components of the system.

b. Investigation of whether different components of the system save information relevant to state of the requested operation.

R2: Components of the system that are responsible for persistence handling should be able to take use of the storage resources provided by IaaS providers.

Activities for R2 include:

a. Identification of the components that are performing persisting related processing as well as handle the business logic operations.

b. If such components exist, split them into separate components.

R3: End users of the system and external clients should be able to the system in the same fashion as they used to used before the initiated migration.

Activities for R3 include:

a. Introduce new components that will distribute requests between replicated components.

b. These components should have the flexibility to add new routing algorithms in future without any change in system architecture.

R4: Once the transformation of the system is completed, a list of migration steps should be available so that it can be served as a reference guide for future migration activities.

Activity for R4 includes:

a. Throughout the implementation, diary of the decisions made during transformation should be maintained.

Some of the above requirements are related with the evolvability characteristics. Table 9 shows the association of the requirements with the evolvability characteristics.

Table 9: Association between Requirements and Evolvability Characteristics

Requirement	Evolvability Characteristic
R1: Different components of the systems should be able to be replicated to scale up or scale down the system according to performance requirements and should be deployable on computing resources offered by IaaS providers.	Extensibility
R2: Components of the system that are responsible for persistence handling should be able to take use of the storage resources provided by IaaS providers.	Changeability

R1: Different components of the systems should be able to be replicated to scale up or scale down the system according to performance requirements and should be deployable on computing resources offered by IaaS providers. **R3:** End users of the system and external clients should be able to the system in the same fashion as they used to used before the initiated migration. **R4:** Once the transformation of the system is completed, a list of migration steps should be available so that it can be served as a reference guide for future migration activities.	Domain Specific Requirements: Scalability, Backward Compatibility and Knowledge management.

3.2.2. Evaluation of Change Impact, Potential Solutions and Test Cases

The Hackystat framework is an OSS system, so the detailed architectural artifacts are not available. High level system architecture diagram shown in Figure 1, along with the description of components shown in the diagram as well the APIs and documentation of the features supported by each component is available. Identification of the components that are to be impacted as a result of the proposed change was done with the help of available documentation as well as the analysis of the code.

Requirement 1 is associated with the domain specific requirement of SaaS model and deals with the scalability features. Scalability is addressed in two different dimensions. First dimension corresponds to the scalability in terms of the computing power whereas second one deals with the scalability of storage resources to handle high volume of persistence data. In order for software applications to offer features of SaaS cloud, components of such systems should be able to be deployed on physical or virtual nodes of public/private clouds so that mode computing resources should be added. The Hackystat is implemented using RESTful web services model and its components are state independent so they can be replicated unless they have some associated persistence handling. But to enable replication, there must be some kind of mechanism that replications is transparent for the clients of such components and clients have a uniform view of the services; irrespective of the fact that how many physical nodes are there. This highlight the need of a new wrapper component on top of the each group of replicated services.

In the next step, all of the different components were investigates to see their dependency on database. It was turned out that SensorBase was the only component that had been dealing with persistence and all the other components interact with SensorBase to handle persistence related issues. Figure 1 shows the interaction of SensorBase with database and with other services of the framework.

Following were the findings after the analysis of the Hackystat against SaaS model:

- New components should be introduced for each group of replicated components of the same type to provide a uniform interface to clients.
- SensorBase component should be re-factored and database related functionality should be moved out into a separate component so that SensorBase can also have replicated deployments.
- Once database layer is separated into a new service, it can easily be programmed to user database and persistence features offered by external storage clouds.

The strategy to perform the testing along with test cases was also decided.

3.2.2.1 Identification and Evaluation of Potential Solution

From technology point of view, solution of two set of components was required to be investigated. One was about the set of components that would serve as a wrapper to the replicated services and other one was about the new database service component.

For the wrapper component of the replicated services, there were following two options:

- To write a RESTful wrapper service for each set of the replicated services with detailed API same provided by the replicated service and route the request to individual services on the basis of the routing algorithm.
- To create a generic server side service component that route the request to the routing service depending on routing algorithm without providing the detail API and delegates the responsibility of request interpretation to the target service.

These two options were investigated against the pros and cons of each one. If first option was chosen, it would require implementing the detailed wrapper API for all the features offered by the wrapped services. It would also require writing the separate wrapper for each cluster of the replicated services as each type of service has different APIs. If second option was to be opted, it would not have required writing the detailed API, hence had provided the possibility to replicate the same wrapper service for different clusters of service and have saves he programming effort. Second option had one more advantage. If there is any change in the API of wrapped services; wrapper services would not need to be changed. Second option was selected for the implementation of wrapper service because it required less programming effort and had more advantaged.

To write a database service, all it was required to take the database related implementation out of the SensroBase service and move it to a new database service with the REST based API for its operations. Only issue for the development team was to get familiarize themselves with the related technologies. Jersey [32] and Restlet [33] were two available frameworks for

implementation using open source Java based technologies. Jersey framework is quite simple and works with annotation based parameters to associate URIs with corresponding functions. Restlet framework is more complicated as compared to Jersey and it requires to define a new class to delegate request from a URI to corresponding functions. However, Restlet framework was chosen to make this new service built on the same framework as used by other services.

3.2.3. Evaluation of Potential Solutions and Changes against Evolvability Characteristics

Table 9 shows that extensibility, changeability, scalability and backward compatibility were the major evolvability characteristics associated with migration of the Hackystat framework to a SaaS platform. To achieve the requirements, two new components are introduced into the system. Hence, modularity of the system is increases as well as splitting of SensorBase component into two components to handle business logic and database related operations separately has reduced the complexity of the framework. Table 8 shows that increased modularity and reduced complexity of the systems have positive impact on analyzability, changeability, extensibility and testability. Scalability and backward compatibility were domain specific requirements associated with enhancements of Hackystat into SaaS platform. These two evolvability characteristics were also positively addressed during the migration phase. Maintaining the knowledgebase of the migration activity so that it can serve as a reference guide for migration of other SOA projects into cloud was a process related requirement and it had been dealt by maintaining the diary of events and technical decisions made during the process of migration. As the strategy adopted during the change activity have positive impact on the evolvability characteristics, so it can be inferred that Hackystat would be able to accommodate technology and business specific changes in future as well.

3.2.4. Testing and build process

Use of automated testing frameworks and build tools is suggested in guidelines of evolvability analysis method. After the development of the new components for the project, system was build using the build scripts mentioned on the Hackystat wiki. Unit testing was also performed with the help of test classes.

3.3. Implementation and Architecture Overview of Hackystat as SaaS Model

Section 3.2 describes the detail of the activities and relational behind the change process. This sections provides a brief overview of the implementation and architecture of the Hackystat SaaS model.

3.3.1. Implementation Overview

One of the two services that are written for SaaS migration is responsible to route the requests between services present in the cluster of replicated nodes. The component that was developed

for that purpose read the information about the URI of the replicated services from a property file. The URIs of the services can be mentioned in the properties file separated by commas as shown in Figure 3.

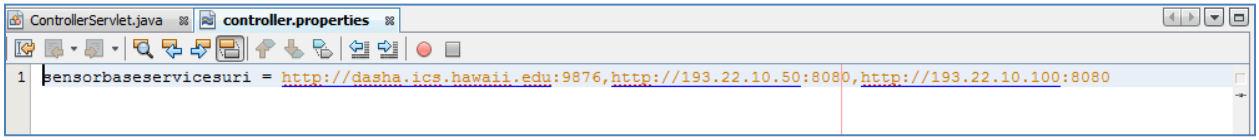

Figure 3: Comma separated URIs of the replicated services in cluster in controller.properties file

In the first version of the component, round robin algorithm is implemented to equally balance load among all the services in cluster. This algorithm reads the address of individual services from the properties file and then returns their addresses to the main functions responsible for routing. Figure 4 shows the snapshot of the main routing functions of this service.

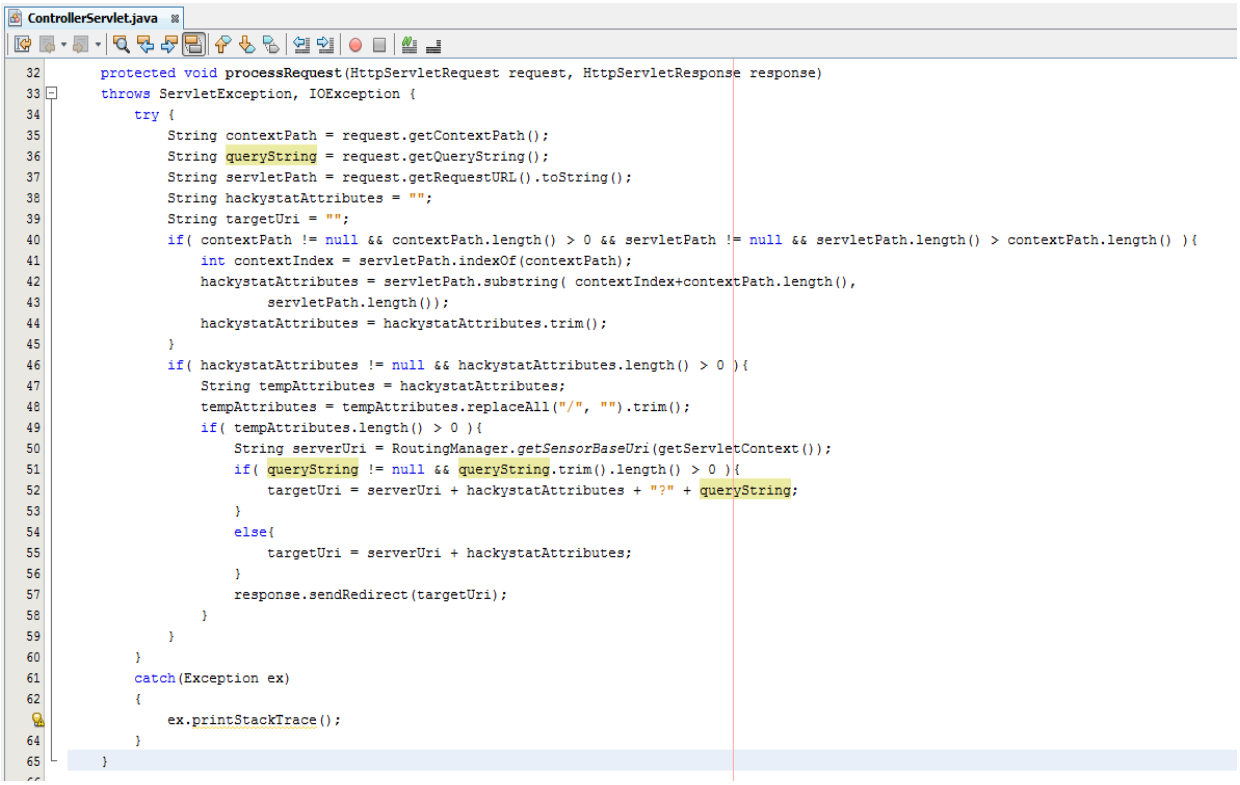

Figure 4: Function to process and route requests among services

Client services of the cluster, which may be other components of the Hackystat framework, data input sensors or third party components, access the cluster through the URI of the controller service and this cluster would appear as a single service to them.

The second service that was written to handle persistence, had been implemented using the Restlet framework and same programming style that was used in other services of the Hackystat

framework. Each method of this services is exposed by a unique URI. Table 10 shows the list of the APIs in terms of URIs. Terms between curly brackets represent variables and can have variable values, whereas the words that are written without curly brackets are part of the REST API URI and cannot be changed. Clients of the database service can access its functionality through these URIs.

Table 10: REST API of database service

REST APIs in terms of URI
/getUserIndex
/getUser/{user}
/deleteUser/{user}
/storeUser/{xmlUser}/{xmlUserRef}
/dbmanager/{xmlSensorData}/{xmlSensorDataRef}
/dbmanager/{xmlGregorian}
/deleteSensorData/
/deleteSensorData/{xmlGregorian}
/getSensorDataTypeIndex
/deleteSensorDataType/{dSdtName}
/getSensorDataType/{sdtName}
/storeSensorDataType/{xmlSensorDataType}/{xmlSensorDataTypeRef}
/getProjectIndex
/getProject/{user}/{projectname}
/deleteProject/{user}/{projectName}
/storeProject/{xmlProject}/{xmlProjectRef}
/getSensorDataIndex
/getSensorDataIndex/{user}
/getSensorDataIndex/{user}/{sdtName}
/getSensorDataIndex/{ownerString}/{startTime}/{endTime}/{uriPatternString}/{sdt}

/getSensorDataIndex/{ownerString}/{startTime}/{endTime}/{uriPatternString}/{sdt}/{tool}
/getProjectSensorDataSnapshot/{ownerString}/{startTime}/{endTime}/{uriPatternString}/{sdt}/{tool}
/getSensorDataIndex/{ownerString}/{startTime}/{endTime}/{uriPatternString}/{startIndex}/{maxInstances}

3.3.2. Overview of the SaaS Architecture

Figure 4 represents the high level architecture of the SaaS implementation. Different ovals in the figure show the cluster of replicated services. Between every interaction point of clusters and between clusters and client services or project a controller service is introduced. This service play its role as explained in section 3.2. Different clients of these services like Project Browser, Tickertape and data input sensors can only have access to the cluster of services through its controller.

The process of service method invocation is as follows. Data input sensors send process and product related data to the SensorBase services through its controller. The controller of SensorBase services delegated requests to one of the SensorBase service in the cluster depending on the routing algorithm. Replicated SensorBase services interact with database service to deal with persistence. Whenever data is to be stored or retrieved from the database it is done through this service.

Functions of the framework are invoked through URIs in top down fashion. Information between different levels of hierarchy is always accessed via controller service of the corresponding hierarchy. Project Browser is a web application that interacts with different services and presents the metrics to the end users through web interfaces. This application resides at the most upper level of hierarchy. For example if a user wants to have the consolidated view of the project specific data, Project Browser has to access the component that is responsible to present the abstraction of the metrics. Telemetry service is the component that deals with this level of abstraction. In a SaaS model, there are multiple telemetry services, so the Project Browser requests the telemetry controller service to delegate request to any of the telemetry instance. When the telemetry instance receive the request, it needs to have access to the daily project data to present the consolidated view of the whole month. DailyProjectData service is responsible to provide the abstraction of metrics on day level. Again there are multiple replicated instances of the DailyProjectData service, so the telemetry service has to request through controller service that encapsulates the cluster of the DailyProjectData services. Similarly, DailyProjectData needs to interact with SensorBase service to get raw data and this request is also handled through the controller of the SensorBase cluster. In this way, the whole process of dealing with replicated services is completed. External clients of the services also require interacting via controller for example in case of Tickertape as shown in Figure 5.

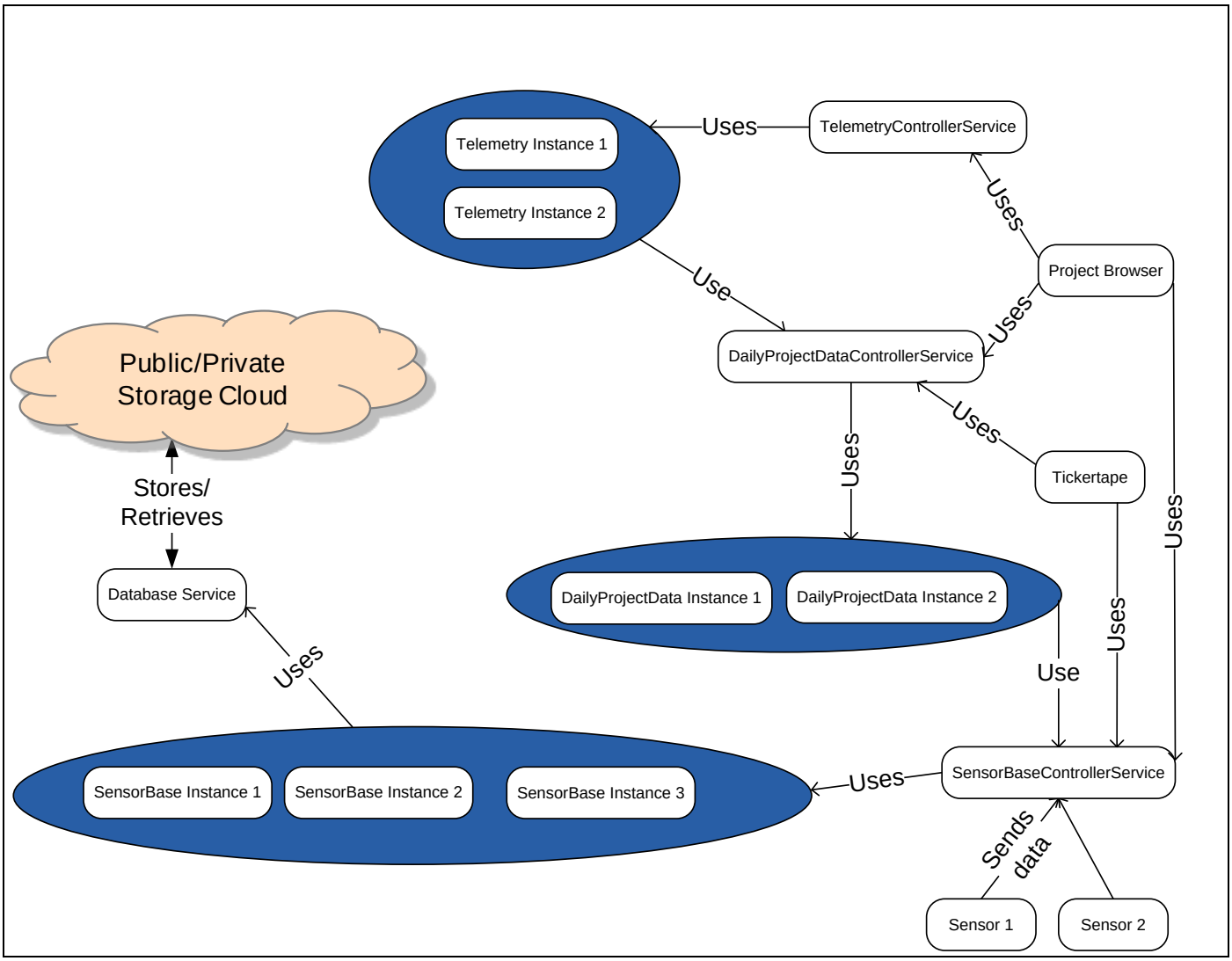

Figure 5: Hackystat SaaS Architecture

3.3.3 Summary of the Migration steps

This section presents the summary of the migrations steps that were taken in order to transform the SOA based framework into a SaaS system. These guidelines can be adopted as recommendations for transforming systems to clouds infrastructure to take use of reliable cost effective solutions offered by cloud infrastructures. The guidelines are:

- Evaluate different system components against business requirements that initiate the migration activity. Scalability, reliability and cost effective infrastructure offered by the cloud infrastructure may be the main objective of the migration activity. Different system components should be evaluated how these can take use of the quality attributes offered by cloud infrastructure. It is to be noted that business objectives that initiate the migration activity may be conflicting with the extra functional or non functional requirements of the

system. A detailed analysis of the business objectives against extra functional requirements of the system should be performed.

- Investigate the target platforms against support for the proposed re-factoring solutions. It is important especially for applications that are focusing on public SaaS and PaaS infrastructures. For software systems targeting cloud infrastructure portability seems hard to achieve because only a limited set of software and frameworks are supported by public cloud providers. Applications developed on PaaS platforms would result in even more tight bound.

- Provide orchestration at service level to ensure that applications can be seamlessly deployed on public, private or hybrid clouds. The orchestration layer would ensure the migration of the components on different cloud environments.

4. Future Work

The evolvability model, presented in section 2.6 consists of two set of guidelines. One consists of recommendations for individual teams and developers working on OSS system. These guidelines suggests that how they can perform evolvability analysis during the enhancements in the system. These guidelines are followed during migration of Hackystat to SaaS model and activities of our experiment show the significant of guidelines to evaluate evolvability. The second set of guidelines is recommended for the member of the control group of the software systems. These guidelines are based on the literature review of the OSS process as well the process that is followed by control group of Hackystat software. However, the complete set of roles needs to be verified in an experimental or real project to identity the practical issues that may be associated with the suggested process.

Section 3 of the thesis presents the work conducted for the migration of the Hackystat to a SaaS model. This migration has added features into the Hackystat to act as a SaaS model while taking advantage of IaaS clouds. This modified framework needs to be tested on a real public or private cloud to see the impact of network delays on efficiency of the system. Some advances features of REST services like computational REST (CREST) also need to be investigated for its impact on efficiency of the system [34].

For SaaS model, database layer of the framework is separated from the corresponding services to take use of storage clouds. This has introduces the new security concern in the REST based architecture of the framework. Service of the database service are exposed through URIs are accessible to everyone on the internet. To enable a secured access to the database service, a subscription mechanism needs to be implemented so that only subscribed and verified services can access the REST based API, and any breach of security can be avoided.

5. Conclusion

The research presented in this thesis is conducted to analyze the evolution in context of open source software systems. Different releases of the Hackystat software are analyzed to see how different evolvability characteristics like analyzability, architectural integrity, postability, changeability, extensibility and testability are addressed during its evolutions. Analysis shows that increase in modularity and reductions in complexity over different releases has played a supporting role throughout its evolution.

Although architecture documentation and detailed design artifacts are considered important for the evolution of the software systems, but hackystat like many other OSS systems continue to evolve without extensive architecture related documentation. Project's website remains a major source of information for developers and users of the systems. It provides the information about high level architecture, video tutorials, information about the API of different components along with java docs and description of functionality.

Unlike proprietary software systems, control group cannot perform the evolvability analysis against architecture before the implementation is already done. This is because of the reason that in OSS systems, different teams work on these systems and make modifications according to their own requirements, often unaware what features are developed by other teams. This highlights the importance of collaborative environment as well as the need to perform evolvability analysis by control group when commit requests are received by independent teams and developers. The quality of OSS systems can be improved by using automated testing and test driven development. Automated build tools and testing frameworks can play a vital role to control the development process.

Experiment on Hackystat framework to migrate it to a SaaS model has shown that systems built on using REST based service oriented architectures can be converted into SaaS models without any major hurdles. However to perform that kind of migration activity on OSS systems requires critical evaluation of the system's components for their dependency on each other as well as on persistence related modules. The detailed analysis of the components is also important because of absence of system design artifacts.

References

[1] Breivold, H.P.; Crnkovic, I.; Land, R.; Larsson, M.; , "Analyzing Software Evolvability of an Industrial Automation Control System: A Case Study," *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on* , vol., no., pp.205-213, 26-31 Oct. 2008.

[2] Breivold, H.P.; Crnkovic, I.; Eriksson, P.J.; , "Analyzing Software Evolvability," Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International , vol., no., pp.327-330, July 28 2008-Aug. 1 2008.

[3] Breivold, H.P.; Crnkovic, I.; Land, R.; Larsson, S.; , "Using dependency model to support software architecture evolution," Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on , vol., no., pp.82-91, 15-16 Sept. 2008.

[4] P. M. Johnson, S. Zhang, and P. Senin, "Experiences with Hackystat as a service-oriented architecture," Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, Tech. Rep. CSDL-09-07, February 2009. [Online]. Available: http://csdl.ics.hawaii.edu/techreports/09-07/09-07.pdf.

[5] Hackystat, http://code.google.com/p/hackystat/ (December, 2010).

[6] Davis, A.; Du Zhang; , "A comparative study of DCOM and SOAP," Multimedia Software Engineering, 2002. Proceedings. Fourth International Symposium on , vol., no., pp. 48- 55, 2002.

[7] Xinyang Feng; Jianjing Shen; Ying Fan; , "REST: An alternative to RPC for Web services architecture," *Future Information Networks, 2009. ICFIN 2009. First International Conference on* , vol., no., pp.7-10, 14-17 Oct. 2009.

[8] Nabaztag, www.nabaztag.com [January , 2011]

[9] Twitter, www.twitter.com [January , 2011]

[10] Breivold H.P., Chauhan M.A., Babar M.A., A Systematic Review of Studies of Open Source Software Evolution, 17th Asia Pacific Software Engineering Conference (APSEC), IEEE, Sydney, Australia.

[11] Software Metrics, http://dmst.aueb.gr/dds/sw/ckjm/doc/metric.html [January, 2011]

[12] Java Metrics, http://javaboutique.internet.com/tutorials/metrics/index-2.html [January, 2011]

[13] Chidamber, S.R.; Kemerer, C.F.; , "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on* , vol.20, no.6, pp.476-493, Jun 1994.

[14] Developing Architecture Views, http://www.opengroup.org/architecture/togaf8-doc/arch/chap31.html. (January, 2010).

[15] Clements, P., Bachmann, F., Bass, L. et al.: Documenting Software Architectures – Views and Beyond. (2007).

[16] Pierce, P.; , "Software verification and validation," Northcon/96 , vol., no., pp.265-268, 4-6 Nov 1996.

[17] Google project hosting, http://code.google.com/projecthosting/ (December, 2010)

[18] Williams, L.; Maximilien, E.M.; Vouk, M.; , "Test-driven development as a defect-reduction practice," Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on , vol., no., pp. 34- 45, 17-20 Nov. 2003.

[19] Bouktif, S., Antoniol, G., and Merlo, E.: 'A feedback based quality assessment to support open source software evolution: the grass case study', International Conference on Software Maintenance, pp. 155-165, 2006.

[20] JUnit, http://www.junit.org/ (December, 2010)

[21] Apache Ant, http://ant.apache.org/ (December, 2010)

[22] Capiluppi, A., and Beecher, K.: 'Structural Complexity and Decay in FLOSS Systems: An Inter-Repository Study', 13th European Conference on Software Maintenance and Reengineering (CSMR), 2009.

[23] Version Control Systems, http://en.wikipedia.org/wiki/Concurrent_Versions_System [January, 2011]

[24] Subversion, http://en.wikipedia.org/wiki/Apache_Subversion [January, 2011]

[25] Maven, http://maven.apache.org/ [January, 2011]

[26] Anand, S.; Padmanabhuni, S.; Ganesh, J.; , "Perspectives on service oriented architecture," *Services Computing, 2005 IEEE International Conference on* , vol.2, no., pp. xvii vol.2, 11-15 July 2005.

[27] Web Services, http://en.wikipedia.org/wiki/Web_service [January, 2011]

[28] Xinyang Feng; Jianjing Shen; Ying Fan; , "REST: An alternative to RPC for Web services architecture," Future Information Networks, 2009. ICFIN 2009. First International Conference on , vol., no., pp.7-10, 14-17 Oct. 2009.

[29] Louridas, P.; , "Up in the Air: Moving Your Applications to the Cloud," Software, IEEE , vol.27, no.4, pp.6-11, July-Aug. 2010.

[30] Google Application Engine, http://code.google.com/appengine/ [January, 2011]
[31] Gmail, http://en.wikipedia.org/wiki/Gmail#Twenty-four_hour_lockdowns [January, 2011]
[32] Jersey, http://jersey.java.net/ [January, 2011]
[33] Restlet, http://www.restlet.org/ [January, 2011]
[34] Erenkrantz, J. R., Gorlick, M., Suryanarayana, G., and Taylor, R. N. 2007. From representations to computations: the evolution of web architectures. In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (Dubrovnik, Croatia, September 03 - 07, 2007). ESEC-FSE '07. ACM, New York, NY, 255-264.
[35] Madhavji, N.H., Fernandez-Ramil, J., and Perry, D.: 'Software Evolution and Feedback: Theory and Practice', John Wiley & Sons, 2006.
[36] OSS, http://en.wikipedia.org/wiki/Open-source_software [January, 2011]
[37] Yingxu Wang; Jingqiu Shao; , "Measurement of the cognitive functional complexity of software," Cognitive Informatics, 2003. Proceedings. The Second IEEE International Conference on , vol., no., pp. 67- 74, 18-20 Aug. 2003.
[38] Grottke, M.; Karg, L.M.; Beckhaus, A.; , "Team Factors and Failure Processing Efficiency: An Exploratory Study of Closed and Open Source Software Development," Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual , vol., no., pp.188-197, 19-23 July 2010.
[39] Khoshgoftaar T. M., Allen E. B., Kalaichelvan, K. S., Goel N., "The impact of software evolution and reuse on software quality." Empirical Software Engineering 1(1): 31-44. 1996.