Architectural Evolution of Open Source Software Systems: An Empirical Study

Submitted by
Muhammad Waseem
(191-FBAS / MSSE/ S08)

Supervised by
Mr. Muhammad Usman
Assistant Professor
Department of Computer Science and software engineering
International Islamic

Department of Computer Science and Software Engineering
Faculty of Basic and Applied Sciences
International Islamic University Islamabad
2012

16/2 Sy

MS
005.3
MUA

Open Source Software-

INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD
DEPARTMENT OF COMPUTER SCIENCE & SOFTWARE ENGINEERING

Dated: 12-11-2012

FINAL APPROVAL

It is certified that we have examined the thesis titled "**Architectural Evolution of Open Source Software Systems: An Empirical Study**" submitted by **Muhammad Waseem, Registration No. 191-FBAS/MSSE/S08,** and found as per standard. In our judgment, this research project is sufficient to warrant it as acceptance by the International Islamic University, Islamabad for the award of MS Degree in Software Engineering.

PROJECT EVALUATION COMMITTEE

External Examiner

Dr. Rizwab Bin Faiz | Assistant Professor

Riphah International University Islamabad

Internal Examiner

Syed Muhammad Saqlain | Assistant Professor

International Islamic university Islamabad

Supervisor:

Mr. Muhammad Usman | Assistant Professor

International Islamic university Islamabad

DECLARATION

I hereby declare that this thesis **"Architectural Evolution of Open Source Software Systems: An Empirical Study"** neither as a whole nor as a part has been copied out from any source. It is further declared that I have done this research with the accompanied report entirely on the basis of my personal efforts, under the proficient guidance of my supervisor and my friends especially my supervisor **Mr. Muhammad Usman** Assistant Professor at IIU Islamabad. If any of the system is proved to be copied out of any source or found to be reproduction of any project from any of the training or educational institutions, I shall stand by the consequences.

Muhammad Waseem

Dedicated to my parents for their support, love and encouragement

Abstract

The success of long-lived software systems depends on the systematic and controlled evolution. The term "Evolution" refers toward the gradual changes that are implemented in a system. Software evolution is an essential part of strategic decisions, features, and financial value of the long lived software projects. Previous work has pointed that there is lack of empirical studies that report the software architecture evolution. Software architecture plays key role to implement changes in efficient manner. Changes that come after successful deployment of any software product affect architectural components; these components are module, packages and classes of any system, depending on their size.

This thesis presents a study on architecture evolution trends of the open sources ERP at system and subs system level. The studied systems are Adempiere, Openbravo and Apache OFBiz. The architectural evolution trends are identified through release wise comparison of the source code of each ERP. This comparison gives modified, added and removed packages or classes. These packages or classes basically act as evolving component.

The study found several interesting results: First, major growth is limited to few subsystems. Second, overall packages and classes increase. Third, dominating change factor is "Modification" of the architectural components. Furthermore, we develop a tool that has two significant uses. First, it can analyze and visualize the architectural evolution. Second, it can recover the architecture diagram in the form of packages from source code.

Table of Contents

Table of Contents

Chapter 1

INTRODUCATION

CHAPTER 1: INTRODUCATION

Free/open source software is famous because of their accessibility to code and functionality. This opportunity provides a great edge to a researcher for evaluating the software from different perspectives. One of these perspectives is software evolution.

Success of long-lived industrial project depends on the successful evolution of the software systems. Software evolution leads towards software requirement change from user or system end in open and closed source software. Success of system depends on how effectively these changes are managed and controlled. According to [10] major part of the software cost increase to meet the changed requirements in long-run.

For better explanation of the term evolution, we see the definition of [62] where he described "Software evolution is an attribute that bears on the ability of a system to accommodate changes in its requirements throughout the system's lifespan with the least possible cost while maintaining architectural integrity".

"Software evolution is the phenomenon of software change over years and releases" [25].Some people would prefer to describe such phenomenon as software maintenance. The two terms refer to the same overall phenomenon, but with different emphasis. The term "evolution" focuses on the gradual changes that implemented into the system.

Lehman also contributes for software evolution by defining two thoughts. First, investigates the driving factors that impacts on evolving software system and the nature of the software evolutionary phenomena; it is called as a noun or "what and why perspective". Second, addresses as a verb or "how" perspective, that is about the practical aspects of the software nature by using that software evolution can accommodate. This may be tools methods and technology [39]. This study deals with both perspectives through observing the software architectural evolution and by developing the tool.

According to (SEI) software architecture is "the structure or structures of the systems, which comprise software elements, the externally visible properties of those elements and the relationships among them". Basically architecture of the system model the high level view of overall system, run time behavior of the system and allocation of the different module. Structure /Structures of the system also show different software element and relation among them.

Modeling the software architecture before the development is important for mutual communication, early design decision and transferable abstraction of the system [57]. Architecture can predict different dimension of the software system that may evolve, these changes usually come from the user nonfunctional requirements and it is refer as software architecture evolution [26] according to [45] software architecture provides basis for software evolutions.

The focus of this research is to analyze the architectural evolution patterns/trends in large open source system. The rest of this chapter describe the overview of open source software, software evolution , software architectural evolution, research motivation, research question , research conducting methodology and overview of the thesis.

1.1. Open Source Software

The idea proposed for sharing source code introduced in 1970's by Richard Stallman, who was one of the Unix OS developer. For this purpose general public license introduce under the GNU. The aim of GPL, to ensure and make possible source code would available to everyone.

Motivation behind the usage of open source software system usually found because of its freely availability of source code to everyone. Open source project organized through common online storage repository. These repositories host hundreds and thousands of free online projects, for example source forage contains approximately 300.590 open source projects and fresh meat host about 44000 projects. Some system also contain there separate repository system that is normally available on their site. These OSS cover the wide range of software system, its start from the simple email internet software to complex internet server like Apache, different operating system such as FreeBSD and Linux. This list leads to huge number of Java based software. A significant number of Enterprise Resource Planning packages are also available with their source code on these repositories. Many reasons are identified and describe in literature for working on Open source projects. Following four reasons make highly adoption of OSS in the world of software development [10, 57, 58].

Openness: Freely availability of source code for download, modification, usage and for distribution purpose.

Flexibility: Provide great opportunity to customize and integrate with other projects for both commercial and research usage

Speed: Due to distributed naturel it can be developed quickly

Standards: Most of successful international standards are used for ensuring long term readability and usability purpose.

1.2. Software Architecture

Software architecture of the system presents high level view, behavior, structure of the system with different software element and relations among them. It is important for mutual communication, early design decision and transferable abstraction of the system [57]. These elements of the software called components as well, when the components held together with connector, it gives the relationship between components. Software architecture delivers the basis to software engineer to work on at the further level of abstraction.

There have been many different definitions reported in literature about software architecture and still is a need for definition that describes complete and overall process of the software architecture and agreed upon by one and all.

We know, software architecture show the relationship among components through connector, in addition we also see that, software architecture has the external agents that called the system stakeholders [37, 40, 47, 67]. These agents may look for different requirements and expectations form same system. It is still a challenge for the software architect that take into account all these expectations and requirements while he/she is creating the architecture for the system. One of the most common used definition that has been given by [8], that states "the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of these components, and the relationships among them".

From this definition we can understand, that architecture defines components and interaction between them, its present detail about components, effect of these components on other, usage, and interaction with other components.

The term "*externally visible* "refers to the properties of components that helps to reminds to the software architect on that level of abstraction where concept of architecture become true. This

definition also show "*internal aspects*" of the architectural components (it is not important for architectural description), where some time all system stakeholder not aware of it.

This definition also show that every system has architecture, because every system consist on the element and relationship between them, in only few cases we have seen software system is itself a single element.

The Perry and Wolf definition of software architecture mention merits that define software architecture is all about Elements, Forms and some Rationales [58]. This mean software architecture is set of Design element with specific shape.

- **Elements:** Software architecture elements can be classified in three types. First, *"processing element* "that performed the processing functionality.it takes some input and generating output after perfuming transformation on considering data. Second type is, *"Data Element"* that contains information that has to use during the transformation. Third and last is, "*connecting element*" that create the relation among communicating element.
- **Architectural Forms:** It's about the weighted property and relationship. Perry and Wolf definition show, each architectural components has some constraints and relationship that decided by the Architect. Basically these properties define constrain over the architectural elements and weighted elements.
- **The Rationale**: when architect decided to design architecture for the software system, he/she has different choices and decisions. Basically rational describe the phenomenon for choices. For example, why particular form selected; which architectural style is more important.it is secured the requirement from stakeholder, different views of the architecture and role of stakeholder.

1.3. Software Evolution

Currently, most of the organization highly depends on software system. Maintaining and achieving the dynamic need of the organization in software system is a serious issue regarding planning and development. Basically, modification and getting some extra functionality for particular existing software system lead to the evolutionary phenomena. Lehman present eight laws for defining what the software evolution is? These laws are formulated by observing

FEAST and IBM OS/360 operating system. Pioneered of these laws use Term E. Type system almost in every law that's mean a real world program that must be evolved [39, 40].

The definition of the [53] relates the term evolution with progressive change in different evolving entity. They also give some benefits in term of value or fitness increasing in a specific time period for a specific project. They suggest progressive change need to determine in each context [44]. Software evolution al can relate, a way of evolving system in an over period of time [79]. This term describe also software changes occur during the system's life cycle. These changes refer to evolution of software, and software evolves within some laws of evolution. Lehman has identified the laws of software evolution that are about the continuing changes, increasing complexity self-regulation, conservation of organizational stability, conservation to familiarity, continuing growth, declining quality and about the feedback system.

Software Architecture is also expected to evolve, especially in case of *Continuing change*, *increasing complexity*, *continuing growth*' and *declining quality*.

This research observe large open source ERP system's architecture, According to [14] following properties of large software system are noted.

- Every large system must be *complex* that leads to
    - Understanding all programs' states problem
    - Extension of existence module or program
    - Communications problem
    - Delaying conceptual integrity problem
- Second essential property is *conformity* that deals with human institutions and systems
- *Changeability* each software must be accept changes
- *Invisibility* there is no geometrics demonstration ,that can visualize software instead of interacting graphs that show different features of the system

These properties of the software system confirm the software evolution phenomena and present the increased need of evolvable software system that accommodates deviations in a cost effective way.

1.4. Software Architecture Evolution

Architecture of the software system represents high level view, behavior, structure or structures of the system with different software element and relations among them [57]. It is important for mutual communication, early design decision and transferable abstraction of the system. Architecture can predict different dimension of the software system that may evolve. Hence we can relate architecture of the system with evolution [26]. In literature we see two types of evolution with respect to architecture, i.e. internal evolution and external evolution [5].

Internal evolution deals with the topology for different components, that may be created deleted or modified during the execution of system life cycle whereas **External Evolution** represent the component's specification that change due to the stakeholder requirements. The focus of this research is studying the internal architectural evolution.

We see a number of approaches in a literature that describe and deal with the evolving nature software architecture. The cost metrics proposed by the [5] that deals with the change operation for the evolving software architecture. Aoyama also proposed metrics for software evolution continuity and discontinuity

Bennett et al described nature of software architecture. They say that the evolvable software system must be changed in controlled way without negotiating on system reliability and promised features [11]. Software architecture evolution also welcomes the integrating crosscutting concerns. That's why; architectural integrity is one of the important aspects. One of the reasons for inconsistency issue is crosscutting architectural integrity is. To handle this inconsistency issue [9] developed the TranSAT Framework. This framework work on the behalf of architectural aspect; that describes new concerns and their integration into the existing available architecture

The study of Jansen et al reports the key and essential concept for architecture evolution. They considered design decision as a key concept capturing of these decisions is important for architectural knowledge, in software evolution design of software architecture and it play vital role. Furthermore changing in architecture of the software may create the possible reason for earlier design decision that can increase the rate of design erosion [26].

1.5. Research Motivation:

Software systems evolve with the passage of time; it brings change that cause increasing complexity. When these changes and complexity not deal rigorously than it may lead huge development and maintenance cost or sometime system failure .Software evolution is an essential component for cumulative strategic decisions, features, and financial value of the software [73].

Evolution can be classified on two levels; it may be external or internal, the former perspective purely concern with system output and the latter address architectural or structure changes. Software architecture evolution comes to be acute part of the software lifecycle.

The focus of this research to understand evolution from architectural point of view, according to [8] software architecture play key role for implementing changes in efficient manner. Changes that com later stage of the any software product effect on the architectural element/components; these elements may be subsystem, module, packages and classes depending on the system size. The aim of the this study is to analyze Open Source Enterprise Resource Planning (ERP) system that often have life time from 3 to 5 year and because this life span must undergo a significant amount of changes for different levels of the system ,e.g. at package level, at class level ,at method level and so on . As the selected systems are large; so evaluating the evolvable architectural pattern on package and class level changes are enough.

During the thoroughly literature review this has been observed there is no case study or empirical study done on architectural evolution of Open /closed source ERP. This study describes and makes contributions to following aspects.

- How architecture grow up in term of packages and classes?
- Identification of most Evolving modules/subsystem.
- Identifying and reporting the dominate change factors

1.6. Research Question

This section presents the topic of research in term of research question and summary of possible answer.

1) **What are the architectural evolution patterns/trends in open source enterprise resource planning systems?**

Software architecture evolves with the passage of time. The architecture of software consists of different components and relation among them. The basic concern of this research is to analyzing and reporting the evolving trends of the architecture in open source enterprise resource planning systems.

The architectural evolution depends on the changes in structural elements of the software. These structural elements are packages and classes. The changes are addition, deletion and modification of the architectural elements.

This research study intends to find following trends of the architectural evolution of the open source enterprise resource planning systems.

- Architectural growth patterns: This involves studying, how architecture of the software grows in terms of packages and classes.
- Identification of most evolving modules/subsystems: This involves identifying and studying those parts of the software that are frequently changed. It will highlight the more growing modules where most of the evolution activities are done.
- Reporting the dominate change factors: There are multiple factors that can be happened with single module. This study also investigated what are the dominate change factor that are playing critical role in evolution.

1.7. Thesis Overview

The organization of the thesis describe as bellow.

Chapter 1 : Chpter 1 provide the overview on open source software , software archtecure , software evolution, motvation for doing research work and research question.

Chapter 2: This chapter provides the detail information about research methdolodgy, system that are being used and data extraction tools and techniques.

Chpater 3: This chpater reports emperical research work on the software evolution ,grwoth evoltuion , structure evioluiton and complexity evoluion etc according to defiend criteria.

Chapter 4: Chapter 4 reports the result that compiled from three open source ERP on system and subsystem level.

Chapter 5: This chapter provide the overview of data extraction tools that are mostly used in emperical studies.

Chpater 6: This chpater describe the archtecure of the developed tools, use case and resulted snaphop.

Chpater 7: This chapter sumarized the contrubutions and future work of thr thesis.

1.8. Summary

This thesis reports the architectural evolution trends in open source enterprise resource planning systems. Previous works have pointed out that there is lack of empirical studies that deals with evolution of software architecture. Software architecture play key role to implement changes in efficient manner. Changes that coms after the successfully deployment of any software product effect on the architectural components; these components may be module, packages and classes of the any selected system, depending on the system size.

This thesis presents a study on architecture evolution trends of the three open sources ERP at system and subs system level. The subject systems are Adempiere, Openbravo and Apache Ofbiz. The reason for java base ERP selection is popularity of the java with mature and established development platform for development of open source products. As result we see a large amount of open source products that has not any objection for their usage. In this thesis we are studying those systems that have 36 month development history and at least have 5 stable releases. The literature reported only 15% of open source projects have a development history greater than 2 years, with only 2% of the projects surviving more than 3 years [11]." Therefore our selection criteria only meet with three open source ERP projects that have long development history and success.

The architectural evolution trends are identified through release wise comparison of the source code of each ERP. This comparison gives modified, added and removed packages or classes. These packages or classes basically act as evolving component. The study found following interesting result.

- Major growth is limited to few subsystems.
- Overall packages and classes increase.
- Dominate change factor is "Modification/Updating" of the architectural components.

As a supplementary research contribution, we also developed a tool that can

- Analyze and visualize the architectural evolution
- Recover the software architecture from source code.

Chapter 2

Research Methodology

Chapter 2: Research Methodology

Introduction

Most of empirical study results based on quantitative data; in this study we extract data from three Open sources Enterprise Planning Systems. This part provides an overview of different source of information that can be helpful for studying the architectural evolution. Software evolution based on historical data. This historical information can be classified in three types [69].

1. Release history that contains software artifact and releases on regular base
2. Revision history typically consist on "version control logs and issue/defect records"
3. Project history made up with project documentation, process information and logs

Typically all above mentioned information stored on many plate forms in a systematic way. There are numbers of tools available that can be used for extraction this information from open source repository as well as from closed source.

2.1 Systems Selection Criteria

The selection of targeted systems play critical role in quality of empirical study; i have selected those systems that must meet the following norms.

1. The system must be Enterprise Resource Planning (ERP)
2. Must be Developed on Java platform
3. Source code of each release must be available
4. The ERP has been actively developed and used since at least three years
5. Further , at least three ERP will require along with minimum 5 stable releases availability

We mined also following repository that are hosted as open source Projects

1. Source forge - http://www.sourceforge.com
2. OW2 Consortium - http://www.objectweb.org
3. Apache Software Foundation - http://www.apache.org

4. Java Community Projects - http://community.java.net/projects/

The reason for java base ERP selection is popularity of the java with mature and established development platform for development of open source products. As result we see a large amount of open source products that has not any objection for their usage.

In this thesis we are studying those systems that have 36 month development history and at least have 5 stable releases. The literature reported only 15% of open source projects have a development history greater than 2 years, with only 2% of the projects surviving more than 3 years [11]." Therefore our selection criteria only affect few in the case of open source Enterprise Resource Planning projects that have long development history and success.

Following section describe the system overview that are being used with their features list

2.2. System Used

Initially number of ERP projects has been that arc open source and developed on Java platform, after applying thc decided criteria; following three ERP being shortlisted for studying the architectural growth patterns

1. Adempiere
2. Open Bravo
3. Apache OFBiz

Above mentioned system briefly explain with their feature list as below

2.2.1 Adempiere [3]

Adempiere is one of the top rated open sources ERP that hosted on sourcforge.net; currently it has 16 major releases, feature list of Adempiere generally classify in three categories that are Admin, ERP Feature, OLAP and Reporting feature. Table 1.3 present the list of Adempiere as per category.

Table 1.3 Adempiere Feature

S.NO	Category Name	Features List
1	Admin	Deals with system admin, Application Dictionary ,partner Relations, Human Resource and Payroll etc
2	ERP Features	• Handle with Sales module, Purchasing Module • Dealing with Manufacturing and material Managements. • Control all type of Finance and Human Resource related activities • Provide a control on Maintenance Management • Deals with Project Management, customer Relationships, POS and Ecommerce Activates
3	OLAP and Reporting	• Business Intelligence, Generate following types of reports • Financial • jasper • Exporting • Extending

2.2.2 Open Bravo [55].

Openbravo ERP is commercial open source web based ERP that has been developed by Openbravo Company. This is suitable for small and medium size companies. Open bravo construct under the java based architecture by using the two development models that are

- Model Driven development
- Model View Controller

By using open bravo ERP small and medium size organization can automate their routine process such as sales, purchase, Manufacturing, finance, and procurement etc. Some of key features are presented below that openbravo contain

- User-configurable home page with extensible widgets also called Role Based Workspaces
- Open bravo can provide the empowerment of multitasking for their users independently
- Data can be edit directly with automatic saving in a grid view
- Enable dynamic column filtering
- Manage different level of information on a single screen
- User-centered design with quick lunch
- Modular based ERP
- User with different roles can be easily created in Openbravo
- Open bravo can be deployed on single server and on network
- Provide the facilities of auditing

The code of open bravo hosted on https://dev.openbravo.com/ , we extracted round about 40 major and minor releases of open bravo with their source code by using tortoise SVN.

2.2.3 Apache OFBiz [6].

Another famous open source ERP that become the part of this Empirical study is apache for business process or Apache OFBiz. Apache OFBiz work under the license version 2.0, apache OFBiz is provide the comprehensive set of feature, few of them listed below

- Advanced e-commerce and different type of catalog management
- Promotion of different product
- Price ,sales & purchase management
- Customer management (part of general party management)
- Stock and warehouse management
- Accounting with invoice, payment & billing accounts, fixed assets etc
- Goods manufacturing management
- "General work effort management (events, tasks, projects, requests, etc)"

- Content management
- Provide mature Point out sale system

For this project we have extracted the source code of 8 major releases of apache OFBiz fromhttp://svn.apache.org/repos/asf/ofbiz/.

2.3. Tools Used

Research in a software evolution basically relies on some historical quantitative measures. Generically we can divide these histories in three types, in which first is release history, second is project history and third is reversion history. In our case we have selected release histories of the selected ERPs and download the source code through tools. Following tools are being used for extracting the empirical data.

2.3.1. Tortoise SVN [68].

Tortoise SVN is a free open source client for managing changes into information in organized way. These tortoises SVN manage files folder and directory with the passage of time.

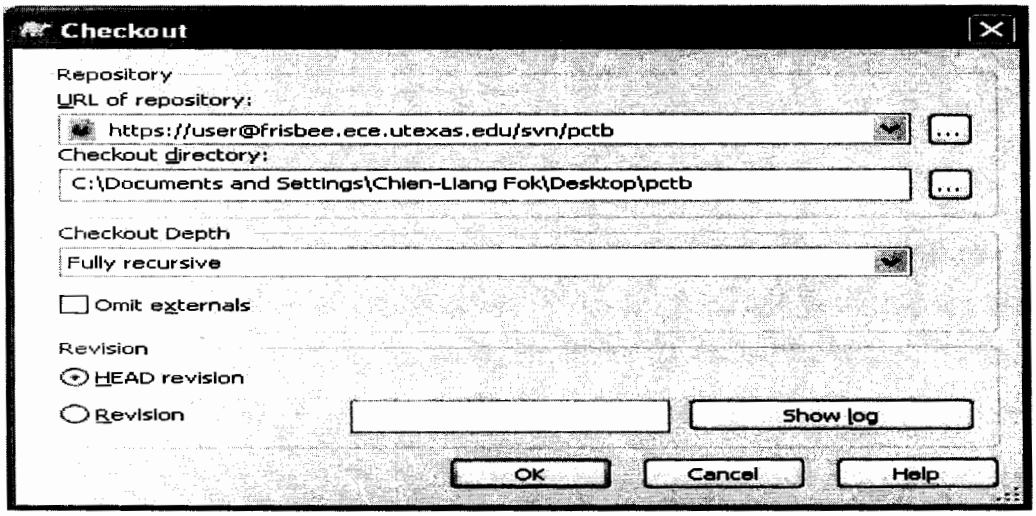


Figure 2.2 Tortoise SVN Checkouts

Usually files are stored in central place that also called central repository. Here we are presenting some feature of Tortoise SVN that make different from other version control tool from other version control tools.

- Shell integration
- Icon overlay
- Easy access to sub versioning commands
- Directory versioning
- Atomic Comments
- Versioned metadata
- Efficient branching and Tagging
- Consistent data handling

Our concern is with tagged releases of selected ERPs that hold the source code of released version of ERP. We are using Tortoise SVN client for windows that available with GUI. We extract the source code of ERP by using SVN Checkout command by providing desired ERP source code URL.

2.3.2. JDiff [33].

Typically understanding the evolving nature of software has become more difficult when we don't have any appropriate tool, Information about changes between different releases of software is a useful for a different software task.

JDiff work on the base of CalcDiff algorithm given in Figure 1.2, CalcDiff perfumes its comparison at packages, classes, method and filed level.

Our concern is only with the extracting evolving nature of architectural elements (packages and classes) from different version of the selected system.

I will extract the following architectural elements from open source system by using JDiff.

- Total number of packages deleted between subsequent releases
- Total number of packages added between subsequent releases
- Total number of packages modified subsequent releases
- Total number of Classless and interfaces deleted between subsequent releases
- Total number of Classless and interfaces added between subsequent releases

```
Input. original program P
        modified program P
        maximum lookahead LH
        hammock similarity threshold S
Output. set of class,class C
        set of interface,interface I
        set of method,method M
        set of node,node ,status N
        Declare. Node n, n
Begin. CalcDiff
        1.compare classes in P and P ; add matched class pairs to C
        2.compare interfaces in P and P ; add matched interface pairs to I
         3.for each pair c, c in C or I do
        4. compare methods; add matched method pairs to M
        5. for each pair m,m in M do
        6. create ECFGs G and G for methods m and m
        7. identify, collapse hammocks in G until one node n left
        8. identify, collapse hammocks in G until one node n left
         9.N = N∪ HmMatch(n,n ,LH,S)
         10. end for
         11.end for
         12. return C, I, M, N
end CalcDiff
```

Figure 2.2 JDiff Algorithm

- Total number of Classless and interfaces modified in subsequent releases

2.3. Methodology

We selected adempiere, open bravo and apache ofbiz as study subjects because these system meet the system selection criteria. Data is extracted through comparison of subsequent release of each system though JDIFF tool. This gives information about the removed, added and modified packages.

2.3.1. Study metrics

The following two metrics are used for evaluating the architecture evolution of open source software.

i. **Java Packages:**

Java packages provide a way in which classes and other members of classes organized under the unique name space for particular module, usually it stored in compressed JAR file. During the development of open or closed source applications. Packages are used to organize classes in a

similar group. The source code of the product maintain via classes, interfaces and methods under the packages. Another, important benefit identified in a literature, the access protection from other application by using concept of separate code library [3], in this study we have to find out the difference of removed, added and modified packages for all available release of every selected ERP.

ii. **Java Classes and interfaces**

Second most important architectural element is classes and interfaces for any system, classes provide the way in which similar kind of stuff organized in same place. Classes consist of public, private and abstract method that are called in outside the class by using object, usability of classes guide toward the some easiness of the object oriented programming. We have to extract the difference of removed, added and modified classes for all available release of every selected ERP as well.

2.3.2. Step by Step Study Approach

To track the trend of open source ERP architecture, the following steps are used.

1. Selection of three open sources ERP.
2. Downloading source code repositories through sub SVN tools. The source code of all repositories can take approximately 10 GB desk space.
3. For Adempiere ERP use the SVN Check Out Command ,entire releases based on tagging mechanism available on www.sourceforage.net
4. For Open Bravo, we used the SVN Check out Command ,all-inclusive release based on tagging mechanism available on https://dev.openbravo.com/
5. For Apache OFBiz, we used the SVN Check out Command as well. Releases are organized on tagging mechanism that are available on http://svn.apache.org/repos/asf/ofbiz/ and
6. For extracting architectural Elements on system and subsystem level, the JDiff tool is used.
7. Information about architectural difference of two consecutive releases on system or subsystem level obtained through this equation.

$$\Delta n = \text{Release}_{(n)} - \text{Release}_{(n+1)}$$

Where the value of Δn obtained from the difference between two subsequent releases.

8. All extracted information stored and analyzed through MS Excel 2010 spreadsheets.
9. The data plotted as in following format
    - System level comparison
    - Sub System level comparison
    - Comparison for major subsystem
10. We analyzed the collected information to determine if either or all ERP show evidence for
    - Where the major growth of system is located?
    - Identification and reporting of most evolving module?
    - Identification of dominate change factors in evolution.

Chapter 3

Literature Review

CHAPTER 3: LITERATURE REVIEW

Introduction

A significant research work has been done on open source software development in last decade; the researchers still have opportunity to study and explore the open source system from different perspective for understanding and reporting the evolutionary phenomena of the nature of OSS. This chapter describes the current state of the art in the development of a theory of software evolution, and where and how it applies to open source software evolution

The literature review of the thesis will covers those important empirical evolution studies of the open source system that deals with the source code or architecture on system or subsystem The literature review is divided into two sections; First Section introduce the proposed criteria for reviewing literature, the motivation behind introducing review criteria is doing studies in concept centric way ,testifying and analyzing easily the reviewed studies, this has been done before in some studies[77] and second part is reported those studies that has been reviewed according to proposed criteria.

3.1. Review Criteria

This section describe the literature review criteria, the selected studies literature review with respect to proposed criteria, followings are the criteria measures.

- **Research Question:** The research question is one of the first methodological steps the investigator has to take when undertaking research. Choosing a research question is the central element of both quantitative and qualitative research and in some cases it may precede construction of the conceptual framework of study. Research question are able to enhance objective of the study, formulation of theory and tell what data to collect and what not to collect. Research on open source study may or may not consist research question.
- **Study Method:** Research hypothesis or research questions validate through some methods, these methods provide step by step process of the way of doing. There are number of research method available for study of the open source software evolution, some are difficult to apply some can be implemented easily and some are implemented with modification. Researcher are presenting which method they are going to adopt in their

research, these method can be implemented with some adjustment in further study. I will report in this literature review which method is used for getting result and how it was implemented on specific open source software for exploring the different evolutionary aspects if the method explicitly defined in selected study.

- **Unit of Study :** Exploring all evolutionary aspects of OSS is difficult in a single empirical study, Researcher work on some Entities/dimensions of the open source software, they may investigate Evolution over Releases, System or program,(E-Type) application, Process ,Models of Process(),folder ,file and procedure . i will investigate systematically in our literature review where researcher focus among from these dimensions and also cheeked what other dimension being consider for investigating evolution of open source software .
- **Study Focus:** As described in previous sections a one particular study it is not possible cover all aspects that are related with evolution of system. Every study carries on some dimensions that might be growth rate, size, complexity, structural complexity, structural design principle, and quality and economics value of the systems. The level of exploring the level might be different for each study it may be folders, files, procedures on large and small system etc.
- **System Type:** Open source software provides good opportunity for observing software product in different stages of their evolution, growth and maturity. Our concern only with getting information from literature in which discuss evolution of open source system, we analyze what kind of open source system or discuss.
- **Number of System:** Literature show fascinating figures about number of open source system, these systems may be a part of one large system or hundreds of systems that was being study in one single empirical study .Under this criterion we just only report how much systems were study in a one particular study.

Study Span: Span of study does mater maturity and accuracy of result. Some researcher may take long time to study the evolution of system that is some time clearly define in there researcher papers. We will report if available how much time will take a particular researcher for particular study under this criterion.

- **Repository span:** A sequence of different versions or releases of an application program or software system implements changes in system quality, performance, function, etc. We will

report all repository age for specific open source system that was study in literature if available.

- **Tools:** Researchers and practitioners have developed a number of tools that has been play a vital role for developing, maintaining and extracting a repository of the open source system. These repositories have complete information of major and minor changes on day by day. We will report how and what type of tools are used for studding open source software evolution.
- **Result/Contribution:** The last evaluating criterion is about the collation of results that has been studied in particular study.

3.2. Reviewed Studies

This section consist on the open source software evolution related studies. The objective of this review is to obtain an overview of the existing studies in open source software evolution. These studies for this review were identified based on a pre-defined search strategy. We select the studies that discuss the open source software system evolution trends and patterns, evolution process support, evolvability characteristics and examining OSS at software architecture level. This study may further discuss the software growth, software maintenance and evolution economics prediction, Modularity, complexity and quality assessment of software evolution.

1. "Evolution in open source software: A case study Software [27]".

Research Question: This study investigates the Linux evolution. The investigation shows, Linux is a large system with two million line of code at the time of study. Size of Linux grows as a quadratic trend. Growth rate was "fully in line with Lehman's sixth law", while the" super linear rate contradict with some consequences of the second law", they also found the higher growth rate in Linux particular subsystem machine drivers. They did not set the any hypothesis explicitly

Study Method: Godfrey and Tu measured evolution form different perspective by using Linux source code with variety of tools and assumptions. They examined 96 kernel versions including stable and development releases by considering the "directory structure of each source release

Study Unit: This study presents the growth of Linux on the system and sub system level. This study has been done on ten available releases by taking ", Line of code and size of the source file as basic measurement unit.

Study Focus: The focus of this study is to validating the Lehman Laws of evolution, the author worked on studying the "Median and average size of implementation files, Median and average size of header files, Growth of the major subsystems and percentage of total system LOC for each major subsystem".

Name, Number and Type of the System

Table 3.1: System used in study 1

S.No	Name of System	Number of System	Type
1	Linux	1	C based open source system

Study Span: Almost six year evolved code has been used for this study, from March 1994 (version 1.0, 487 source code files comprising over 165,000 lines of code) to January 2000(stable kernel is version 2.2.14).

Tools: There is no information available about the tools or commands, which has beenused for extracting data from Linux kernel repository.

Result/Contribution: The result of this study shows, Size of Linux grow as a quadratic trend ,growth rate was "fully in line with Lehman's sixth law", while the" superliner rate contradict with some consequences of the second law", and researcher of this study found the higher growth rate in some subsystem of Linux like machine drivers".

2. "An empirical study of open-source and closed-source software products [56]"

Study Hypothesis: This study evaluates the different general theories about the open source software through empirical data colocation. The authors of this study set the following five hypotheses.

- *H1. "OSS grows more quickly than proprietary software"*
- *H2. "OSS projects foster more creativity "*

- *H3. "OSS is less complex than proprietary systems"*
- *H4. "OSS projects have fewer defects and find and fix defects more rapidly"*
- *H5. And last is "Open source projects are more modular than closed-source projects"*

Study Method: This study investigate there hypothesis through empirical research method, for successful completion of this research they, they use the total growth of the project in term of function, Line of code, complexity over a period of time, function added over a time, average complexity of the function and added function, function modified and there percentage in total function and correlation between function added and function modified.

Unit of Study: This study explore the answer of their hypothesis on system and major sub system level releases by setting line of code, total function, added function, complexity of the function, percentage of the modified function and correlation between function as study metric

Study Focus: The focus of this study to find the evidence for growth trends, creativity ratio, simplicity, defects and modularity amount both in open and closed source subsystem.

Name , Number and type of the system : Table 3.2 present the system that are used in this study

Table 3.2: System used in study 2

S.NO	Name	Number of system	Type of system
1	*OS-Linux*	6	C based open source
2	*OS-Linux*		
3	*OS-Apache*		
4	*OS-GCC*		
5	*CM-Project A*		Closed Source
6	*CM-Project B*		
7	*CM-Project C*		

Study span and repository span: There is no clue for the study and repository span of any project repository that is being used .No other information's also available in this study for

example; how much project is mature, how long projects are been running successfully and How much these project exact age have.

Tools: There is no information available for tools that are used for extracting metrics from selected open and close source systems.

Result/Contribution: The following result has been compiled from this study

About Growth: similar growth rate with respect to increase of the Line of code, complexity and number of function.

About Creativity: Results of this study show that that four out of the six projects a (OS Apache, CM Project A, CM Project B, and CM Project C) have a reducing growth rate. Where Linux and GCC both have increasing growth rate, none of the commercial projects from (Paulson and Succi) study show positive growth rate over time.

About Simplicity: The total complexity of each of the project releases is examined and an average of all the releases performed.

3. **"Opportunities and challenges applying functional data analysis to the study of open source software evolution [64]"**

Research Question: This study characterizes the complexity pattern with respect to coupling and cohesion in evolution of open source software projects. They did not define hypothesis explicitly.

Study Method: The authors of this study introduce the new approach for exploring the complexity pattern from empirical data that's called functional data analysis. This technique implemented through by calculating CplXLCoh and counting the number of Line of code for each release.

Unit of study: This study is on system and file Level of different releases by taking line of code and CplXLCoh measure

Study Focus: The focus of this study is on to finding the complexity pattern of open source software with respect to coupling and cohesion by using the empirical data of 105 open source project

Name, Number and Type of the System: in this study 105 systems are used but there the information about the name, study spans which can show the matureness of research and projects repository is not available.

Tools: No information available about tools that are being used for extracting metrics from selected open and close source systems

Result/Contribution: The reported result of [45] study are, initially they find the change pattern in actual complexity evolution value, on behalf of first part of the result they explore the evolution of complexity values.

4. "The Evolution of FreeBSD and Linux [18]"

Research Question: This study done with same parameters that has adopted in [20, 27, 28]. The results of this study do not support the hypothesis regarding, "OSS systems grow at rates that exceed in traditional systems". Basically this study did on the behalf of prior study that's work on proving or disproving the Lehman's laws of evolution. Hypothesis for this research also not define explicitly.

Study Method: The evolution of FreeBSD [18] done by dividing the stable release of the Linux and FreeBSD into file system structure, "LOCs (Lines of Code), number of directories, total size in Kbytes, average and median LOC for header (dot-h) and source (dot-c) files, and number of modules (files) for each sub-system and for the system as a whole "were the metrics of the empirical study that are used.

Unit of study: The researcher of this study takes the system and major sub system level releases as a unit of study for studying the evolution in open source system.

Study Focus: A major concern of this study is finding the growth patterns of the open source and closed source software

Name, Number and type of the system: Following two open source tool used in this study

Table 3.3: system used in study 4

S.no	Name	Number	Type
1	FreeBSD, Linux	2	Open source

Study Span: Information about study span is not available for this research.

Tools: Only CVS is used for extracting the require information from source code repository.

Result/Contribution: This study observes the evolution on system and subsystem level in which they found mostly the growth rate of FreeBSD and Linux are growing constantly or in a linear way.

5. "Analyze the Design Principles in Architectural Evolution [48]."

Research Question: Maintaining the architecture of large and complex project is serious issue for long time projects, the author of this research worked on the "structural design principles that are used in practice" and set the some question as a base for their study,

- *"Does the evolution of size indicate continuous growth? "*
- *Second is "Does complexity increase or was there any effort to maintain or reduce it?"*
- *Does coupling decrease and cohesion increase?*

Study Method.: This researcher validate their research questions by using Eclipse 47 major release, they set the number of internal static and dynamic dependency, number of plugins, number of external dependency for founding out complexity, size and coupling during the evolution.

Unit of Study: This research is about the evolutions of open source system for conforming weather design principal followed or not. For this purpose they take the 47 releases of Eclipse.

Study Focus: Major concerns of this study are studying the growth pattern of the open source and closed source software.

Name, Number and Type of the system: Following tool is used for this study

Table3.4: System used in study 5

S.no	Name	Number	Type
1	Eclipse	1 but with 47 release	Java based open source system

Tools: table 3.5 presents the tools that are used for extracting empirical data for this study.

Table3.5: Tools used in study 5

S.No	Name	functionality
1	Bash awk and XSLT	Use for reading the architectural elements and relations from XML file
2	Crocopat	For commuting relation
3	CVS	For extracting repository
4	FETCH	
5	Open Office's	For visualizing the analysis

Result/Contribution: The result of this study presents the size of the architecture increases more than four folder. The evolution follows a" segmented growth patterns, in which different segments have different growth rates".

6. **"The Evolution Of Source Folder Structure In Actively Evolved Open Source Systems [15]"**

Research Question: This study finds the patterns of software code structure with special consternation on relation between code size and folder depth, relation between system change and functional growth, and OSS source code tree evolvement during the evolution.

Study Method: Following five steps explain the process of performing empirical study on open source software system for studying the structural evolution.

- Selection of 25 projects among from 400 open source project
- Defining attribute
- Selecting parsing tools
- Pattern recognition from extracted data
- Interpretation of the results

Unit of Study: The basic concern of this study is to found the patterns of size and structural evolution, horizontal and vertical expansion with vertical shrinking in open source software.

Study Focus: Focus on to know the depth of the Code Size, Functional growth pattern and changes and there relation with some other aspects likes code structure,

Name, Number and Type of the system: This study done by using twenty five open source tools

Table 3.6: system used in study 6

Name	Number	Type
Arla ,Ganymede ,Gwydiondylan ,Ghemical ,Gimpprint ,Gist ,Grace ,Htdig ,Imlib ,Ksi ,Lcrzo ,Linuxconf ,Mit scheme ,Motion ,Mutt ,Nicestep ,Parted ,Pliant ,Quakeforge , Rblcheck ,Rrdtool ,Siagoffice ,Vovida ,Sip ,Stack ,Weasel ,Xfce	25	Open source

Study and Repository span: Information about study span or for the age of project that are under research in this study is not being defined clearly.

Tools: Following three tools are used for extracting information.

Table 3.7: Tools used in Study 6

S.No	Name	Functionality
1	XSCC,	Counting lines of code
2	Graphviz	Extracting source trees
3	PERL scripts	Quantifying the number of changes between releases.

Result/Contribution: This study analyzes the similar value regarding size from twenty five application program that are being focused; they did not found the apparent relation between size of the system and the average size of the folder. They also reporting about the some stable point for average size of the source file in KB that show average size of almost all projects are stable.

7. "Change Rate And Complexity In Software Evolution[12]"

Research Question: This work is about identifying the refactoring part of source code and prioritizing these for maximum impact on the system.in this study they finds the relation between

change during the evolution and gradually improvement in overall complexity during the life of project.

Study Method: This study is about the finding the change and complexity ratio in open source projects. Following steps are made for preceding this research

- Project selection
- Metrics selection
- Identifying Less stable part
- Measuring complexity for stable and unstable part of the software
- Analyzing the file and candidates for refactoring

Unit of Study: The Stable and unstable part of source code is target unit of this research.

Study Focus: This study focuses on complexity and changes rate in evolutionary life of the software.

Name, Number and type of the system: The following open source system is used for this study

Table 3.8: System used in study 7

Name	Number	Type
Arla	1	Open source

Study Span and repository Span: The age of the repository and study conducting duration is unknown for this study

Tools: Only Perl scripts are used for extracting desired information

Result/Contribution: This study results indicate the source files which are subject to the higher change rate include a large portion of highly complex functions. They argued refactoring of this less stable, more complex functions, is "likely to have an impact on evolvability. ".

8. **"Studying the Evolution of Open Source Systems at Different Levels of Granularity: Two Case Studies [17]"**

Study Hypothesis: This study evaluates the multiple hypotheses around the evolution, these studies are taken from several studies, and data for these hypotheses are taken from the long lived software on different stages. The text of Hypotheses given below as it;

- *"Staged evolution and discontinuities in trends"*
- *"Disciplined behavior of evolutionary attributes within stages of software evolution"*
- *"Ripples & feedback"*
- *"Anti-regressive work"*
- *"Increasing complexity"*
- *"Language heterogeneity growth"*

Study Method: Methodology that used in this study for obtaining the data and evaluating there hypothesis is as follow.

- Calculating the size
- Exploring the folder structure
- Measuring Complexity through McCabe and Halstead indexes
- Calculating the number of distinct files modified, added and deleted per release
- Finding anti-regressive work

Unit of Study: Source file, source folder, source tree, size, release sequence number, level number, file added, deleted, modified and cyclomatic complexity.

Study Focus: This work done on the folder for measuring the size, activity rate and complexity

Name, Number and type of the system: Following two system are used for this study

Table 3.9: Tool used in study 8

S.no	Name	Number	Type
1	Mozilla and Arlas	2	Open source

Study span: Considered 3.5 years of the Mozilla evolution

Tools: No information available about tools

Results: The result of this study shows the decreasing growth with constant complexity rate on growth and procedure level. ARLA's growth found as superliner. Furthermore Arla and Mozilla represent two related but distinct modes of system evolution.

9. **"Class movement and re-location: An empirical study of Java inheritance evolution [51]".**

Research Question

The basic and most important unit of object oriented paradigm is inheritance; this study identified the ratio of added and modified classes during the evolution of java base open source system. This study answer the research question how and where these classes were added and deleted in system. Furthermore this study also highlight where existing classes were moved across inheritance hierarchy.

Study Method: Following four steps that has been done for understanding the movements of classes in inheritance evolution.

- Selection four java base Project
- Following Data is collected
  - Total number of predecessor classes through depth of inheritance Tree(DIT)
  - Total number of method Measure in a class through Number of method metric(NOM)
  - Number of method calls from one class to another is extract through Message passing coupling method(MPC)
  - Methods, that access different field\ variables are collected through Lack of Cohesion Of Methods (LCOM)
- Data Analysis

Unit of Study: The Classes, total number of methods, message passing method and cohesion methods are used as a basic unit for this study.

Study Focus: Finding out where the new class added, deleted and modified across the inheritance with respect to inheritance evolution

Name, Number and type of the system: Following four systems are used

Table3.10: System used in study 9

S.No	Name	Number	Type
1	HSQLDB, JasperReports:, SwingWT, Tyrant,	4	Java base open source

Study span/Repository: The age of repository and study span is not available

Tools: Jhawk is used for extracting the information from selected systems.

Result/Contribution: Following results are compiled from existing study

- Classes are continuously moving across the inheritance hierarchy
- Most frequent changes were restricted to just one sub-part of the overall system.
- Maximum three levels of threshold has been observed when system using inheritance

10. "An Empirical Study of Evolution of Inheritance in Java OSS [51]"

Research Question: Some other studies of OO software have reported avoidance of the inheritance mechanism. Researcher of this study had claim about inheritance hierarchy's grow" breadth-wise" rather than "depth-wise" after studying the working behavior of the developer

Study Method: This empirical study conduct as follow

- Selection of seven open source system
- Collocating data with following parameter
    - Total number of predecessor classes through depth of inheritance Tree (DIT)
    - Specialization Ratio (SR) metric is used for calculating number of subclasses/number of super classes.
    - Via Reuse Ratio (RR) metric measures inheritance
    - Measuring the number of children Method
- Data analysis

Study Unit: Classes, specialization ration metric reuse ratio and children methods used as targeted study unit.

Study Focus: The basic focus of this research is to give the deep look to inheritance hierarchy.

Name, Number and type of the System: Following systems are used in this study

Table3.11: System used in study 10

S.no	Name	Number	Type
1	HSQLDB, JasperReports, EasyWay, SwingWT, JAG, JBoss and Tyrant:	7	Open source

Study Span: No information available about the life of projects that are being used.

Tools: Jam and Jhawk used for this study.

Result/Contribution: Empirical Analysis of (Nasseriet al,2008 b) evaluate the trends in inheritance evolution for number of version in OSS. The result of this study conform some earlier result as well.

- One of them is system will evolve 'width wise' rather than 'depth wise'
- Found the strong affinity\tendency for classes to be added.
- Result of this study also supported the ratio of specialization, Reusability and number of children metric.

11. "System evolution at the attribute level: an Empirical study of the three java system OSS and their Refactoring's [50]".

Research Question: This study focuses on the net changes of the classes with evolutionarily relationship between classes and attribute. "The research question addresses the evolutionary relationship between classes and attributes as well as the connection between those changes and their re-factoring nature".

Study Method: The following steps are used for conducting this study.

- System chosen from Sourceforge.net
- Extracting inheritance and size measure with jhawk Tool
- Data analysis
- Result reporting

Unit of Study: The targeted unit of this study is classes

Study Focus: The basic focus of this research is to explores the relationship between evolutionary attributes, classes and there refactoring nature.

Name , Number and type of the system : Following three systems repositories are used

Table3.12: System used in study 11

S.no	Name	Number	Type
1	HSQLDB, JasperReports, Tyrant,	3	Open source

Study Span: Information about time period of repository used is not available, so we can't say how much the project was matured

Tools: Jhawk used as metrics collection from selected system

Result/Contribution: The author of this study worked on the attribute of classes rather than focusing and finding the evolutionary trend of complete class, they found the some interesting pattern in attribute level. In this study three open source system was under observation for the analyzing attribute evolution. Results from these open source system suggest that looking only at 'class' evolution is a just short-sighted policy. We need to consider features inside the classes (i.e., attributes). They also evaluate, it might not be possible to predict complete evolutionary behavior only focusing on analysis of classes' evolution. It can also help developers predict changes in future versions. Finally, the study shows where remedial effort can be applied by developers, since we can identify behavior inside classes that may need to be re-engineered.

12. "Empirical Study of Evolution at the Method Level[52] "

Research Question: In this study Nasseri and Shepperd investigate how open source system evolves on method level. They take the empirical data from four java base open source software with the help of different tools.

Study Method: Following steps are followed

- Selection of four open-source systems
- Data Collected through Total number of predecessor classes through depth of inheritance Tree (DIT)
- Finding normalized percentage (Norm.) change

Unit of Study: Classes and there methods are being used as study unit

Study Focus: Exploring and identifying the evolving nature of software on Methods level

Name, Number and type of the System: Table 3.13 present the system used in this study

Table3.13: System used in study 12

S.no	Name	Number	Type
1	HSQLDB, Jasper Reports, Swing and Tyrant	4	Open source

Study span: Information about the age of repository used is not available,

Tools: No information available tools that are used for extracting information from selected systems.

Result/Contribution: The result of this study shows classes provide a tried and trusted base for measuring OO evolution studies , because most of changes may occur on different level that become as cause of evolution. They investigate four OO java systems. This study highlights few benefits for developer in a term of understanding corrective effort for future version. The results of this study also support the view that need for different level of Granularity.

13. "Metrics and Evolution in Open Source Software [38]"

Study Hypothesis: This study explain the affiliation among "fan in-fan out" coupling and classes. The result show quality of newly added classes improved instead of deleted classes. This study also justifies the Lehman 1st, 2nd and 6th laws are followed as per result come from study while Law about "Declining quality (7th law)" not had been satisfied. Following hypothesis take under observation for this study.

- "H 1: The class growth throughout all versions will be positively reflected in the fan-in/fan-out coupling metric values."
- "H2: The class growth throughout all versions of the program will not be positively reflected in the cohesion metric values. "
- "H3: The average fan-in coupling of group A will be higher than the average fan-in of group R."
- "H4: The average fan-out coupling of group A will be lower than the average fan-out of group R".
- "H5: The average cohesion of group A will be higher than the average cohesion of group R."

Study Method: The open source system that is used for this empirical study was *"JFreeChart", one of the best open source solutions for* charting library.

Following procedure ware adopted by the authors

- Selecting a "jarjar diff " tool for extracting data
- Extracting classes information with respect to size, coupling and cohesion from twenty two version of the target tool
- Finding out the removed and added classes from two adjacent version
- Comparison of the different version with respect to added and deleted classes.
- Performing analysis for coupling and cohesion on added and deleted classes.
- Identifying the relationship between number of classes and coupling and cohesion

Unit of Study: Young Lee et al select the classes with some other metrics like size, coupling and cohesion.

Study Focus: This study focus on the finding out the relationship between the newly added and deleted classes with respect to coupling and cohesion, they also conform where how the Lehman laws of software evolution are prove or not prove.

Name, Number and type of the system: Only one system is used for analyzing metrics and evolution in open source products.

Table3.14: System used in study 13

S.No	Name	Number	*Type*
1	JFreeChart	1	*Open source java base*

Study Span Information about projects repository not been disclosed in research.

Tools: Jarjar diff tool used as metric extractor.

Result/Contribution: This study present the behavior of the open source software system with respect to evolution, following result are compiled after observing the 'JfreeChart.

- Gradually increase classes with improved reusability
- Strong coupling
- New added classes has higher Fan in and lower Fan out coupling
- Follow the Lehman's 1st, 2nd and 6th law while reject the 7th law

14. "Measuring the evolutionary stability of software systems: case studies of Linux and FreeBSD. [80]"

Study Hypothesis: This study validate stability of the software during the evolution, following hypothesis are define.

- "**H01a:** There is no correlation between the measurement of source code distance and the measurement of structure distance of evolving Linux modules".
- "**H01b**: There is no correlation between the measurement of source code distance and the measurement of structure distance of evolving FreeBSD modules".
- "**H02a**: there is no significant difference between the means of the source code distance of Linux kernel modules and Linux network modules."
- "**H02b:** there is no significant difference between the means of the structure distance of Linux kernel modules and Linux network modules."
- "**H02c**: there is no significant difference between the means of the source code distance of FreeBSD kernel modules and FreeBSD network modules."
- "**H02d:** there is no significant difference between the means of the structure distance of FreeBSD kernel modules and FreeBSD network modules. We used one-way ANOVA to"
- "**H03a:** there is no significant difference between the means of the source code distance of Linux kernel modules and FreeBSD kernel modules."
- "**H03b:** there is no significant difference between the means of the structure distance of Linux kernel modules and FreeBSD kernel modules".
- "**H03c:** there is no significant difference between the means of the source code distance of Linux network modules and FreeBSD network modules."

Study Method: This case study done on two open source operating system, in which explore the evolutionary stability behavior of the Linux and FreeBSD C module via their own design program, that consist on two part. First, it concern with the diff algorithm that calculate the source code distance between two versions. Second, it is used the "LOC, number of functions, Number of loops, number of branches and number of included libraries". Basically these metrics helps to calculate the structural distance in two versions

Unit of Study: This study base on five study unit i.e. LOC, loops, libraries and branches and number of functions

Study Focus: This study inspect the evolutionary stability of the open source software system

Name, Number and type of the system: Linux and FreeBSD are used for this study

Table3.15: System used in study 14

S.no	Name	Number	Type
1	Linux ,FreeBSD	2	Open source

Study Span: This study used the two open source system Linux version from 2.1.114 to 2.4.20 having 51 month evolutionary time period and FreeBSD version 3.0 to 5.0(this as well have same time period).

Tools: In this study Jarjar diff used for extracting difference between two versions of the system w.r.t number of class.

Result/Contribution: This study is about the designing, developing, presenting and evaluating (according to the hypothesis) the model for evolutionary stability of the different components of the software. This model can measure the source code and structural distance among two version, as a case study they use Linux and FreeBSD.

15. "The Linux kernel as a case study in software evolution [36]"

Study Hypothesis\Research Question

Linux is a one of important open source operating systems that gives the huge amount of empirical data to researcher for testing and validates their hypothesis. Israel and Feitelson examine the data of 810 version of Linux kernel study. On the behalf of this data they evaluate the Lehman laws of Evolution.

Study Method: Consider the 840 full releases of the Linux kernel that has been lunch from 1994 to 2008. They consider the .h and .c file of the entire Linux code of the both production and

development version of the Linux. The evaluation of the Lehman laws become possible through different metrics those implicitly or explicitly define in a case study.

Unit of Study: Different releases of Linux Kernel are used.

Study Focus: Validation of the Lehman laws still remarkable, researcher come with hypothesis, arise a question on it, and then validate their hypothesis. like some other researcher, (Israel and Feitelson, 2009) also examine the Lehman laws of software evolution on Linux Kernel Study.

Name, Number and type of the system: Linux kernel is used with following main module in this study.

Table3.16: system used in study 15

S.no	Name	Number	Type
1	810Versions Linux kernel, main module are init, kernel, mm, fs, net, ipc	1	Open source

Study span: The age of repository of Linux is about 14 year that ware used for this research from March 1994 and ending on August 2008.

Tools: Information about tools that are being used as study metric extractor is not available.

Result/Contribution: This study evaluate the Lehman's laws of evolution by taking 800 version of Linux kernel, which consist on development, major releases and slightly updates within kernel results from this study.

- Support continuing growth and changes related laws,
- Occurring of Large changes support the Conservation of familiarity.
- "Self-regulation and feedback system" laws hard to justify.

16. "Impact of release intervals on empirical research into software evolution, with application to the maintainability of Linux [67]".

Research Question: Most of open source evolution study done with respect to release sequence number (RSN), the study of Thomas et al take the Linux system and done accordingly Release

date. Basically this study identify the location of the Linux module, where that grow linearly and also they identified the amount of Common coupling grow exponentially.

Study Method: Thomas et al doing explore the Linux for coupling and growth rate with by using release dates. This study also done on same way as other Linux related empirical study, only distinction between this and other studies are, it was getting through release date

Unit of Study: The basic unit for this study is used was Linux Series Number, different versions and release interval of that specifics version and series.

Study Focus: Analyzing and identifying the different impact on open source system during the evolution with respect to release date/intervals is the major focus of this study.

Name, Number and Type of the system: Linux is used for this study

Table3.17: System used in study 16

S.no	Name	Number	Type
1	Linux	1	C based open source

Study span: No information available for age of the repository being taken

Tools: No information available tor tools being used

Result/Contribution: This study analyzed with respect to date of releases rather than version, the result was reported after analyzing the Linux.

- Coupling grow exponentially
- Development of modern software is a complex process
- For finding growth rates of software evolution need to take great care about metric.
- Open source revolution bring changed working environment for tester and developer.

17. "A replicated and refined empirical study for use of friends in C++ software[24]"

Study Hypothesis: Research show there are few empirical studies that deals with roles of friend functions in object oriented paradigm, this stud, which was all about the C++ Friend functions.

Hypothesis that are explicitly define as it follows

- "**H1a**: Classes declared as friends of other classes have higher import coupling (NAS i) than classes which are not declared as friends"
- "**H1b:** Classes declaring friends have higher export coupling (NAS e) than classes which do not declare friends. "
- "**H3:** The more declared friends of a class, the more private and protected members in that class Classes that declare friends and that engage in inheritance appear lower in the inheritance hierarchy than other classes engaged in inheritance."
- "**H4**: Classes that declare friends and that engage in inheritance appear lower in the inheritance hierarchy than other classes engaged in inheritance."
- "**H5:** Classes that do not engage in any inheritance declare more friends than classes which do engage in inheritance."
- "**H6** Classes declared as friends are bigger than other system classes."
- "**H6a** Classes declared as friends have a higher NMC than other system classes."
- "**H6b** Classes declared as friends have a higher LOC than other system classes."
- "**H7** Classes declaring friends are bigger than other system classes. Again this hypothesis is evaluated based on both NMC and LOC."
- "**H7a** Classes declaring friends have a higher NMC than other system classes".
- "**H7b** Classes declaring friends have a higher LOC than other system classes".
- "**H8a** Classes declared as friends have higher import coupling than other classes when adjusting for size."
- "**H8b** Classes declaring friends have higher export coupling than other classes when adjusting for size."

Study Method: In order to enlarge the previous study and improve the external validity of this study, open-source software projects from "*www.sourceforge.net* "were analyzed. There are a number of reasons for choosing to evaluate open-source software. The projects analyzed in this study were taken from the top 100 most downloaded projects on sourceforge.net. In examining the class hypotheses, only systems which declare at least 5% friend class relationships and have at least 10 classes declared as friends and declaring friends will be considered

Study Unit: Classes are used for validating Hypothesis as basic units

Study Focus: Enlarging and improving the external validity of Pervious and Currant study

Name, Number and Type of the System: Following thirteen systems are used for this study

Table 3.18: System used in study 17

S.No	Name	Number	Type
1	ShareazaEmulemorph, Licq, Emule, Emuleplus ,Abiword ,Audacity, Winscp, Dc++, Bzflag,Ultravnc ,Scummvm, Wxwidgets	13	Open source

Study Span: Information about time period of repository used is not available, so we can't say how much the project was matured

Tools: There is no information available about data extraction tools for this study.

Result/Contribution: The result of this study highlight the different properties of classes that are associated with the usage of friends function in object oriented paradigm, it also evaluate the some design and language facilities trade-off for friends function constructor in practice . Moreover this research highlights link between the classes has not been established heretofore and "use of the friend mechanism is independent of other class design issues".

18. "The Evolution of Open Source Software using Eclipse Metrics[3]"

Study Hypothesis: Open source software is becoming widely adopted by commercial, public and academic organization; the study of ajlan al reported the evolution process in Eclipse, by using Eclipse metrics. The main aim of this research is to measure the evolution of OSS using.

Study Method: This study done on Guice Software where they examining package metrics by using fowling type of metrics.

Cyclomatic Complexity		
Package Version	V-1	V-2
com.google.inject	22	16
com.google.inject.tools.jmx	6	6
com.google.inject.util	6	10
com.google.inject.matcher	3	1
com.google.inject.name	3	2
com.google.inject.spi	3	11
com.google.inject.jndi	2	2
com.google.inject.internal	-	13
com.google.inject.commands.intercepting	-	3

Figure 3.1: Methodology used in study 18

Study Unit: Package and method used as basic unit for this study.

Study Focus: This study explore the evolution of OSS Guice using, eclipse metrics (EM)

Name, Number and Type of the System: Following systems are used for this research study

Table3.19: System used in study 18

S.no	Name	Number	Type
1	Guice Software,	1	Open source

Tools: Eclipse Metrics (EMs) are used as data extractor tool.

Result/Contribution: The following results are obtained from Gucci Software after applying Eclipse Metrics are observed in OSSs evolution context.

- Version 2 has 10 packages, 284 classes and 10029 lines
- V1 has just only 8 packages 148 classes and 4419 lines
- This study used EM and focused on five metrics and all of them prove that there is an evolution from V- 1 to V-2. Moreover, it means that version 2 has also added new features over Version 1.

19. "An automatic approach to identify class evolution discontinuities [4]."

Study Hypothesis: During the evolution of software system, some features come in system, some say good bye that and some are modified for improving the performance. This all may happen on different level of software; it may be on system, packages, classes or Function level. The work was inspired from vector space information retrieval to identify class evolution discontinuities and, therefore, cases of possible re-factoring.

Study Method: This study done through vector space approach that is inherited from linear algebra and vector composition i.e., vector sum. Following steps are being selected for completing this research.

- Finding data for Cases of Refactoring Considered, this done through collecting by following type of information
  - Class Replacement,
  - Class Extraction, Class Split,
  - Class Merge,

- Class Merge into a new Class,
- Recombining Classes,

- Calibrating the Threshold
- Approach Scalability
- Tool Support

Study Unit: Classes are used as targeted unit for this research

Study Focus: This study is done to know the classes' continuities and discontinuities in evolutionary phenomena.

Name, Number and Type of the System

Table3. 20: System used in study 19

S.no	Name	Number	Type
1	*Dnsjava*	1	Open source

Study Span: Information about time period of repository used is not available, so we can't say how much the project was matured

Tools: Tools that used for extracting empirical information in this study are listed in Table 3.21.

Table3.21: Tools used in study 19

S.no	Name	Number
1	Perl scripts, Java CC, java lexer	3

Result/Contribution This study present an approach that based on 'Vector Space .This approach aimed to identify the merge, replace, split and change classes for observing the evolution .according to antoniol and penta opinion, this can provide a useful support when a project manager is interested to study the evolution of a feature across all the releases of a software system.

20. "Defect Prediction using Combined Product and Project Metrics - A Case Study from the Open Source "Apache" MyFaces Project Family[71]".

Study Hypothesis: Predicting defect for carrying out quality assurances activity is one of the major tasks for maintaining the life of software successfully. This research investigates the defect predication with data from the open source software. In this case study following hypothesis are being evaluated.

- **Contribution of Project Metrics.*H01***

  There is no project metrics (pj) that has statistically significant impact to dependent variable Py compare to product metrics (pd). If r is a function to check whether there is a strong correlation between variables.

- **Accuracy of Defect Prediction using Combined Project and Product Metrics**

  A prediction model that used combination of project and product metrics have higher ARE value Compare to prediction model that used only product metrics.

Study Method: Predicting defect in open source software project is one of the initial case studies that have been done with following steps.

- Defining object as a case study unit for Apache and MyFaces project
- Defining depended and independent variable
- Formulating research hypothesis

Study Unit: The basic focus of this research is to identify the different types of defects on system level.

Study Focus: Measuring the quality assurance practices in open source software was the main focus of this study. Activities, that are binge analyzed in this study was analyzing issue related to design, code testing and "code peer review.

Name, Number and Type of the System

Table3.22: System used in study 20

S.no	Name	Number	Type
1	Apache. MyFaces Project Family	1	Open source

Study Span: Information about age of the repository and duration of the study is not being available.

Tools: Following tools are used in this study.

Table3.23: Tools used in study 20

S.No	Name
1	StatSVN tool, Jira query commands Eclipse metrics tool, plug- SPSS

Result/Contribution: This study was one of initial study that has been performed on OSS community projects for understanding the quality aspects, where they found strong relation between selected metric from different releases and defect growth.

21. "Evolution and growth in large libre software projects [42]"

Study Hypothesis: Studies of different researcher shows Some Lehman laws of evolution want re-thinking , as programming and management paradigm shaft with the passage of time , work of [42] also present the empirical study on Linux Kernel with Family of Libra software that give some interesting result about Lehman Laws of evolution.

Study Method: The methodology used in this work based on an open source code that is publically available on internet. Following steps are involved

- Selection of software
- Downloading there source code and storing in a database
- Measuring the size of source code by using SLOC metric
- Plotting these measures against the time

Study Unit: Measuring source line of code on system and subsystem level

Study Focus: The basic focus of this study is to evaluating the Lehman's laws of evolutions.

Name, Number and Type of the System: Following projects are evaluated in this research

Table3.24: System used in study 21

S.No	Name	Number	Type
1	Linux. Based kernel ,kdelibs, jakarta-commons, mcs, mono, koffice. Kdepim, gnumeric, gtk+,xml-xerces, galleon, httpd-2.0, xml-xalan, kdebase, kdenetwork, kdevelop, ant, evolution, gimp	20 Projects	Open source and Libra software

Study Span: Repository age For Linux and BSD kernel about 12 year and for Family of Libra projects is about 2 to 3 year.

Tools: Following tools are used for evaluating the selected system

Table3.25: Tools used in study 21

S.No	Name
1	SLOC counts, CVS Tools

Result/Contribution:

The results of this work are summarized follow

- "The Linux kernel exhibits a super-linear growth rate. Most of the growth is due to new functionality and added hardware support, not to bug fixing."
- "Much of the functionality (especially device drivers) is complex and extensive, but also relatively independent from each other and from the rest of the system".
- "External contributions (both for adding and maintaining code) were frequent in the devices and architecture subsystems. Maintenance is often done by third parties."
- "Large parts of the kernel (especially device drivers) do not require active maintenance, but are still shipped with Linux just in case the user needs them."
- "Fourth Lehman's law of software evolution is presumably not fulfilled in the case of Linux."

22. Mining Software Evolution to Predict Refactoring [59].

Study Hypothesis: Predicting about the location of changes on the behalf of development history during the evolution of open source product done through the empirical study by Ratzinger, J et al by determining following hypothesis,

- H1: Evolution data is a good predictor of future refactoring.
- H2: It is possible to predict refactoring on short time frames.
- H3: We can accurately predict the number of future refactoring for each file.
- H4: There is a common subset of features essential for predictions in different projects.

Study Method: Empirical study of Ratzinger, J et al about the predicting and detecting about the change location in future, this case study done through several steps that was

- Data understanding with leaning and target period
- Processing on evolutionary data by using CVS
- Information extracting from evolutionary data
- Appling data and analyzing result

Name, Number and Type of the System: Following projects are used for this research

Table3.26: System used in study 22

S.no	Name	Number	Type
1	Argo UML, Spring	2	Open source

Tools

Table3.27: Tools used in study 22

s.no	Name
1	Pearl ,Java Scripts and CVS

Result/Contribution: This study contributes for understanding the refactoring nature of the software in open source software. Following major contribution has been identified.

- Development of the refactoring classification model.
- Proving refactoring is an essential propriety for quality of the projects.

- Demonstrating lines activity rate and number of lines altered per commit provide much information for the assessment of refactoring.

23. "Monitoring software evolution using multiple types of changes[2]"

Study Hypothesis\Research Question: Studying the software evolution is debatable topic, where researcher studies the evolution from different perspective. Basically software evolution is all about the changes; it may be modification, deletion and addition of the something that can help the survival of the software product. This experimental Study focuses on three type of changes (modification, deletion, addition) and confirmation of the Lehman 5th law.

Study Method: This experiment consists of following steps.

- Extracting changes that ware addition, modification and deletion of source code module.
- Performing experiment
- Concluding results

Study Focus: Monitoring the trend of software evolution with respect to counting module and three type of changes, and additionally conformation of Lehman's 5th law was the basic focus of the this study.

Name, Number and Type of the System

Table3.28: System used in study 23

S.no	Name	Number	Type
1	KTorrent(45 releases), GNOME(168 releases), Konversation(13 versions), Evince(46 releases)	3	Open source

Tools: No information available about tools that are being used as extractor

Results: This study based on two sets of experiments that are related with counting modules from existing evolutionary measure and second experiment is about proposing the way of monitoring software evolution. The authors of this study suggest that all changes activities "addition, deletion and modification "in successive releases of software system should be monitored for better evolutionary picture.

24. "The evolution of Eclipse[47]"

Research Question: This research presents the study on evolution of Eclipse, they investigate whether the laws of software evolution supported by the data or not, main focus of these researcher on Law about increasing complexity, counting growth, and continuing changes.

Study Method: This empirical study investigates the Eclipse SDK's data from seven major releases that are number from 1.0 to 3.3. some minor release are also consider as a research data for this study, .zip, number of classes , LOC .java and jar files are the major focus for quantifying the Lehman's Laws of Evolution. These all data extract through their own developed Python scripts to count files and file sizes, and to compute differences (additions, deletions, modifications) between releases using various Unix commands ("diff, find, grep, sed").

Study Unit: Classes and line of code are used as basic unit of this study

Study Focus: Exploring the evolutionary pattern in ECLIPSE.

Name, Number and Type of the System

Table3.29: System used in study 24

S.No	Name	Number	Type
1	Eclipse	*java*	*Open source*

Study Span: Six year data of Eclipse From release 1.0 in November 2001 to release 3.3.1 in September 2007.

Tools

Table3.30: Tools used in study 24

S.no	Name
	Python scripts ,Unix commands ("diff, find, grep, sed"), STAN, Eclipse's defect reports and Microsoft Excel

Result/Contribution: In this study considered the Eclipse IDE on global and "namespace" or subproject level for finding out the evidence about three of the laws of software evolution,

- On global level, Eclipse's data supports laws 1st and 6th. The 2nd Law of evolution partially supported.

- At subproject level, a negative exponential model seems to capture well the relationship between size and the size ranking. And also they identified a variety of size and complexity behaviors at subproject level.
- Suggestion for extending this study
- Can be extended for remaining five laws (laws 3rd, 4th, 5th, 7th and 8th).
- Designing a comparative analysis of different indicators and approaches due to lack of generally accepted set of measurement approaches
- Studying the external Eclipse plugins in terms of lifetime, quality, popularity and effort etc.

25. A replication case study on FreeBSD [15]

Study Hypothesis: This study validate different claims about producing high quality product in OSS development within low cost, following hypothesis are formulated as Research millstone,

- H1. "Open source developments will have a core of developers who control the code base, and will create approximately 80 percent or more of the new functionality. If this core group uses only informal ad hoc means of coordinating their work, the group will be no larger than 10 to 15 people."
- H2. "If a project is so large that more than 10 to 15 people are required to complete 80 percent of the code in the desired time frame, then other mechanisms, rather than just informal ad hoc arrangements, will be required in order to coordinate the work. These mechanisms may include one or more of the following: explicit development processes, individual or group code ownership, and required inspections."
- H3. "In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems."
- H4. "Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects."

- H5. "Defect density in open source releases will generally be lower than commercial code that has only been feature-tested, that is, received a comparable level of testing."
- H6. "In successful open source developments, the developers will also be users of the software."
- H7. "OSS developments exhibit very rapid responses to customer problems."

Study Method: Following steps are being followed in this study.

- Research question from FreeBSD development team
- Extracting FreeBSD Source code with CVS
- Studding FreeBSD-bugs from e-mail list and Bug Report Database

Study Focus: Supporting the hypothesis, that has been established by the Mockus et al.

Name, Number and Type of the System

Table3.31: System used in study 25

S.no	Name	Number	Type
1	*FreeBsd*	1	Open Source

Study Span: No information available time period for repository being used

Result/Contribution: This study examined whether the FreeBSD project supported Mockus et al hypotheses. Collected data from FreeBSd was big enough for supporting all Mockus et al hypothesis.

- This data supports hypotheses(H3) about developers, core developer and contributors
- Also support the (H5) and (H6), which is about the defect density and users
- Results from this study show, the hypothesis about core developers (H1) needs revision.
- It also not support the hypothesis about the need for a proper procedure to organize the work ,which is Hypothesis 2

26. "Evolution of the core team of developers in libre software projects [61]."

Research Question: This study analyzed the role of core team in evolution of Libra Software, this study provide and propose the quantitative approach to analyze the different role of Libra software Team.

Study Method: Employer turnover is very common in traditional development, however in liber software projects the study of developer turnover has not been an active research topic. r.

- Selecting Liber Package
- Extracting every comments via CVSAnalY Tool
- Splitting life time of project in equal duration
- Identifying the most active developer through comments Producing a graph

Study Unit: Core teams deploying on system and sub system level of any project

Study Focus: Core team evolution with the passage of time

Name, Number and Type of the System

Table3.32: System used in study 26

S.no	Name	Number	Type
1	Ten releases of OpenBSD	1	Open source

Tools: Only CVS tool is used for extracting the empirical information.

Result/Contribution: The major focus of this research study is to validate the **methodology** that has been proposed for evaluating the core team working on libra projects.

27. "An automatic approach to identify class evolution discontinuities[4]"

Study Hypothesis: This study proposed an approach for identifying discontinuities of the classes during the evolution which based on "vector space information retrieval". In this research they contribute for locating the possible refactoring in term of classes merge, split and rename. They do not define the any hypothesis explicitly.

Study Method: This approach locates the refactoring events that inspired from vector space with Linear algebra and vector composition.it is done through mapping of classes into vector space, that done via idfj where " idfj = total Number of Documents/Number of Documents containing

the jth term", another part of this study is case to be consider as refactored classes, it done through following way

- Class replacement identification
- Extrication of classes
- Knowing about the split class
- Classes that merge
- Two classes merge into one new class
- Recombining of classes

Study Unit: Classes are used as a basic unit for study.

Study Focus: The authors present the approach to know about the identifying refactored class during the evolution of open source system

Name, Number and Type of the System

Table3.33: System used in study 27

S.no	Name	Number	Type
1	40 releases of dnsjava	1	Java

Tools

Table3.34: Tools used in study 27

S. No.	Name
1	JavaCC, Java lexer, Perl Scripts

Result/Contribution: This research present approach that identified the refactoring difference between Class Level by using two subsequent releases, basically the aim of this research is to know about the split, merge and modified feature of the class.

28. "Empirical Study on Class Sizes for Large Java Systems [31]"

Research Question: This study find the small class size phenomena in term of LOC by doing study on different java base open source system, they found only few class, that have large

size where as many classes have small size, this study also favor the object oriented approach adoption and usage of small size of the classes.

Study Method: This study performed on eighteen java base system of different domain, They count physical source line of code in each source file (.java file) for finding the size of the system, and do not count private and inner classes separately as the sizes of private and inner classes are counted into the associated public classes. And they also consider Java Interfaces as special cases of abstract classes and count interface files as class files.

Study Unit: The study unit use for this research is Line of code.

Study Focus: Measuring the size of distributed classes.

Name , Number and type of the system

Table3.35: System used in study 28

S.no	Name	Number	Type
1	Ant, Zeus, Cocoon, Jena, jetSpeed, Tomcat 5, Protégé, jBoss, JDK 1.4.0, JDK 5.0, Netbeans, Eclipse, Tomcat 4, ArgoUML, myFaces, jEdit, jHotdraw, Jetty, DrJava, JavaCC, jCV, Struts, Compiere, James, Bluej, JabRef, JMeter, InsECT, KBVT, JBooks	30	Java base open source system

Study Span: There is no information available for the life span of the project repository

Tools: No information available for data extractor tools in this study

Result/Contribution: This done the studied on multiple of Java based open source systems in which they found

- That many Java classes have only small sizes whereas a few classes have large size.
- Furthermore they discovered that the sizes of classes are 'lognormal *distributed'*. then they derived a novel size estimation model based on the lognormal nature of the class size distribution.

- The model Of Hongyu Zhang el shows that the size of a large Java system can be estimated based on the number of classes with ,
- They also believe the object oriented approach leads the small size orientation.

29. " ADvISE: Architectural Decay In Software Evolution[30]"

Research Question: The following research questions are formulated.

RQ1: "What are signs of architectural decay and how can they is tracked down."
RQ2: "Do stable and unstable micro-architectures have the same risk to be fault prone?"

Study Method: The study method is consisting of following five steps.

- Step1: Extracting class diagram.
- Step2: Identifying classes renaming using structured bases and text bases similarities.
- Step3: Computing the diagram matching between two version using bit vector algorithms.
- Step4: Making architectural diagram cluster for finding stable macro architecture.
- Step5: reporting the result of micro architecture between two programs version.

Study Unit: The major units of this study are number of triplets, classes and class diagram that are extracted from two version of selected system.

Study Focus: This study proposes a quantitative approach for studying the object oriented systems architecture. The architecture represent as set of triplets (S R T), where S and T are the classes and R show the relation between them.

Name, Number and type of the system

Table3.36: System used in study 29

S.no	Name	Number	Type
1	JFreeChart, Rhino and Xerces-J,	3	Open source

Tools: The PADL reverse engineering tool is used for recovering the class diagram from source code.

Results /Contribution: The following contribution and results are compiled from this study.

This study presents the novel approach called ADvISE for studying the architectural decay.

Presenting bit-vector and incremental clustering algorithm that can match the two versions of any program architecture. This approach applied on three open source software and following results are obtained.

- Graph of the architecture provides the useful insight of the programs.
- Stable microarchitectures and this is less bug prone than the decayed architectures

30. An Approach for Evaluating the Effectiveness of Design Patterns in Software Evolution[54]

Research Question: This study consists of two parts. First, proposes the evaluation approach for design patterns effectiveness during the evolution of software systems. Second part presents the study result of the experiment for validating the approach. There is no research question defined explicitly.

Study Method: The approach is defined by using following steps.

- Defining pattern application context
- Defining pattern application model
- Examining the pattern effectiveness in evolution

Study Method: The demonstrate experiment conducted by using following criteria.

- Step1.Source code available openly.
- Step2.Ten version of the selected system should be available.
- Step3.Selected system must be application system instead of any framework.
- Step4 software architect should not be changed.

Study Focus: This study proposes the evaluation approach for design patterns effectiveness during the evolution of software systems.

Study Unit: There are three design patterns are used for this study i.e. strategy, command and template.

Name, Number and type of the system

Table3.36: System used in study 30

S.no	Name	Number	Type
1	JAXE	1	Free java base XML editor

Tools

There is no information available about the data extracting tools that are used in this study.

Results\Contribution: The following research contribution has been made by this study.

- Proposing the analytical effectiveness assessment method for design patterns
- Developing the tool for proposed method
- Conducting experiment on well-known open source systems.
- The result show the most of the design patterns are found at the latter stages of the software and there are some short life design patterns instances that exist in almost every design pattern.

31. "The Evolution of ANT Build Systems[49]"

Research Question: The following research questions are formulated for this study.

- RQ1. " *Do the static size and complexity of source code and build system evolve similarly*?"
- RQ2. " *Does the perceived build-time complexity evolve?* "

Study Method: The following steps are followed by conducting this research study.

- Data extraction from official software releases of open source products.
- Extracting the static and dynamic evolution metrics from extracted data.
- Analyzing the each release snapshot build system files with program source files and getting the result log after executing the build system.
- Reporting the results

Study Unit: To validate the research question spate metrics are used for both perspectives. For static perspective, target count, target task, file count, Halstead Complexity and static line of code is used. The understanding of dynamic perspective is achieved through Build Graph Length, Build Graph Depth, Target Coverage and Dynamic Build Lines of Code (DBLOC) metrics.

Study Focus: This paper presents the build system for ANT from static and dynamic perspective. Static system discusses the specification from source code domain perspective by using the software metrics while the dynamic perspective is the representative of the simple build runs that are conducted for analyzing the output logs.

Name, Number and type of the system: The following four project are used

Table3.37: System used in study 31

S.no	Name	Number	Type
1	ArgoUML, Tomcat, JBoss, Eclipse	4	Open source software

Results: The observations on static and dynamic perspective show the growth and complexity of the build system having different patterns. The growth of Eclipse correlated with the project plugging count. Build system do not convert in recursive to flat design when the established. There is highly correlation observe between Halstead complexity and system size in SBLOC.

32. "The Evolving Structures of Software System [72]."

Research Question: This study deals with the nature of evolving software by using little house concept that's provides the microscopic view of the software system. This concept illustrate the software system can be model in a graph with five components. There is no research question defined explicitly.

Study Method: The study conducting method is consisting of following steps.

- Selection of the 13 open source system
- System should be developed in java language
- System at least having 4 year age with five public releases.
- Extracting the graph of each system and exporting into pakek file format.
- Identifying the classes for each components of little house.

Study Unit: The graph and classes are used as the basic study unit, the other software metrics are afferent coupling, lack of cohesion methods, cohesion of responsibility, depth of inheritance tree, public fields and public methods.

Study Focus: This study observe the nature of evolving software by using little house concept that's provides the microscopic view of the software system. The components of little house are largest strongly connected components (LSCC), classes from *In*(In),classes from *out* (out), classes from *Tendrils*(Tendrils),classes from *Tube*(Tube) and a class that has no connection*(Disconnected)*.

Tools: The Connecta and Pajek are used for extracting the graph and other related metrics of this study.

Name, Number and type of the system

The following four project are used

Table3.38: System used in study 32

S.no	Name	Number	Type
1	DBunit, freeCol, HIbermate, jasper, jgroup, jGNash, Jml, Jsch, Junit, Logisim, Med's, Phex, Squirrel.	13	Open source software

Results: The study address the understanding the nature of software system by using the little house concept, basically that consist of the five components. The result show there are two components that suffer in substantial degradation. The following results are identified about the each component.

- LSCC: The empirical evidence show the ratio of good classes DBunit decreased as the program increases.
- Disconnected: There is little variation in quality of the classes and the rate of good classes decreased by 1.8.
- In: the classes evaluated by good with the growth of the system
- Tubes: the rate of classes with good of COR decreases by 7.2.
- Tendrils: the rate of classes with good of AC decreases by3.

33. "An Exploratory Study of the Evolution of Communicated Information about the Execution of Large Software Systems [63]".

Research Question: This study understands the dynamic nature of the software by using communication information (CI) about the execution of the system. The following research question has been formulated.

- **RQ1:** How does the *CI* evolve over time?
- **RQ2:** What types of modifications happen to *CI*?
- **RQ3:** What information is conveyed by the short-lived *CI*?

Study Method: This study based on the exercising CI features across the each release of the selected system. Each version of the selected system runs in experimental environment with the same workload. The execution logs are collected from subject systems. For Hadoop two logs are generated first is wordscount and second is grep. The enterprise system workload is collected through choosing the subset of the feature across the all version. The communication interfaces are recovered by generating the execution logs.

Study Unit: This study deals with the communication interfaces that are deleted, modified and added over the time within two systems.

Study Focus: The focus of this study is providing the tractability technique between communication interfaces and APPS.

Name, Number and type of the system: The following four project are used

Table3.39: System used in study 33

S.no	Name	Number	Type
1	Ten release4s of Hadoop, nine releases of enterprise system	2	One open source and one commercial enterprise system.

Results: This study shows the communication interfaces have the higher changes rate across the all versions. The Log Processing Apps that track implementation-level CIare more insubstantial than domain-level log processing Apps's CI because of short live implementation-level CI. Moreover the result show the additional maintenance effort require for maintaining the log processing apps's in term of resource allocation.

34. "An Evolutionary Study of Reusability in Open Source Software"

Research Questions: This paper presents the conceptual model for reusability and how the software reuses with the passage of time. There is no research question explicitly defined.

Study Method: In this study two open source are used according to following criteria. After that different require metrics are extracted with the help of tools.

- The source code of the system must be available.
- Selected system should be developed in java platform
- Enough version of the software should be available

Study Unit: The study units are divided into attribute, sub attribute and metrics categories that are flexibility instability, understandably comments, size, number of classes, %comments, LOC, NOM, portability, independence, fan-out, maintainability, complexity, variability and abstractness.

Study Focus: This study illustrates the conceptual model for reusability assessment and studying the reusability of the software in evolution.

Name, Number and type of the system: The following four project are used

Table3.40: System used in study 34

S.no	Name	Number	Type
1	Jasmine, pBeans	2	Java base open source system

Results: The results show the java base assembler "Jasmine" packages are remaining same in 6 version of the evolution from version 1.0 to 2.4.while the number of classes evolved from 99 to 118. Moreover methods 174 methods are increased from 618 to 792.the five pBeans packages are evolved in ten selected version from three to eight. The 21 classes are increased from 28 to 49 and 180 methods are increased from 161 to 341 methods.

35. "An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution [51]."

Research Hypothesis: The following research hypotheses are formulated in this study.

- H1: "Are there more bug fixes after API-level refactoring?"
- H2: "Do API-level refactoring improve developer productivity?"
- H3: "Do API-level refactoring facilitate bug fixes?"
- H4: "Are there relatively fewer API-level refactoring before major releases?"

Study Method: The process of validating research hypothesis starting from identification of refactoring revisions from selected systems, the second step is identification of bug fix revisions, third step is identification of bug-introducing changes and fourth step is change distilling and last step is manual inspection of automatically collected data.

Study Unit: This study takes API and bugs as a study metrics.

Study Focus: This study deals with quantitatively and qualitatively with the evolution of API of the three open source systems.

Name, Number and type of the system: The following four project are used

Table3.41: System used in study 35

S.no	Name	Number	Type
1	Jdit, JEdit,, Columba	3	Java base open source system

Results: The results show the number of bugs increase after the API level refactoring's. The time taken for fixing the bugs' decrease a large number of refactoring revisions include bug fixes at the same time or are related to later bug fix revisions. The last, API level refactoring occurs frequently before releasing the major releases of application.

36. Analyzing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study [41].

Research Questions: This study focus on the two goals: First, forecasting about clone density in subsequent releases of the system. Second, applying statistical methods for understanding the clone proprieties. The following research questions are formulated.

- Using a simple statistical model how accurately can we predict the amount of code clones in future releases?
- Is there any common pattern in the evolution of clone density over releases of evolving software systems?
- Is there any significant difference between the existence and evolution of exact clones and near-miss clones?
- Do programming languages/paradigms have any effect on the amount and evolution of code clones in the evolving systems?
- How do the sizes of systems and functions affect density of exact and near-miss clones?

Study Method: This experiment conducted with the help of NiCad clone detection tool .the following steps describe the study methods

- Adjusting the clone density and its correlation coefficient.
- Forecasting the clone density.
- Identifying the subject systems.
- Clone detection in subject systems.

Study unit: Line of code and functions are used as major study unit of this experiment.

Study Focus: This study investigates the evolution of "near-miss clones" concept in medium to large open source software of different type. This study covers the lavage domain of software that is written in different language namely C#, C and Java. Its cover the three types of clone i.e. Type 1, type 2 and type 3 clones.

Name, Number and type of the system

The following four project are used

Table3.42: System used in study 36

S.no	Name	Number	Type
1	Apachi ant, ArgoUML Commander4j, DavMal Jasper Report, JEdit	6	Java base
2	Conky, GCC, GIT, Linux Kernal, PostGrecSQL, Sammba	6	C
3	NANT, CruiseControl, iTextSharp, ProcessHacker WixEdit, ZedGraph	6	C#

Tools: This experiment conducted with the help of NiCad clone detection tool.

Results: The following results are identified after conducting this experiment on relation between functions in system and the amount of clone changed.

- Increase in the number of function, the number of clone fragments are also find as increase.
- There is a week relation between clone density and number of functions.
- Average clone density for larger system is less than from smaller system.
- The average error are estimated in all system is 2.35.

37. Using source code metrics to predict change-prone java interfaces[20]

Research Hypothesis The following research hypotheses are formulated.

- H1: IUC has a stronger correlation with the #SCC of interfaces than the C&K metrics
- H2: IUC can improve the performance of prediction models to classify Java interfaces into change- and not change-prone

Study Method: The first step of conducting this study is Extracting the SVN,CVS,and GIT repository of the selected projects with the help of repository extraction tools. The second step

consists of source code metrics computation in which model importer and metrics computation steps are performed. Third is SSC extraction that also consist of retrieving the versioning data from repositories and source code change analysis .After collecting the metric from source code metrics computation and SSC extraction the correlation and prediction analysis is performed.

Study Unit: This study basically deals with the java interfaces. Following methods are used for understanding the change prone java interfaces. First is Object-Oriented Metrics & Interfaces

The following metrics are used to test the OO interfaces

- Coupling Between Objects (CBO)
- Lack of Cohesion Of Methods (LCOM)
- Number Of Children (NOC)
- Depth of Inheritance Tree (DIT)
- Response For Classes (RFC)
- Weighted Methods per Class (WMC)

Study Focus: The goal of this empirical study is to evaluate the possibility of using the IUC metric for predicting the change-prone interfaces and to highlight the limited predictive power of the C&K metrics.

Name, Number and type of the system: The following project are used

Table3.43: System used in study 37

S.no	Name	Number	Type
1	Hibernate3, Hibernate2, eclipse.debug.core eclipse.debug. ui eclipse.jface eclipse.jdt.debug eclipse.team.core eclipse.team.cvs.core, eclipse.team.ui eclipse.update.core	10	Java base

Tools: Then following tools are used for extracting the study metrics

- Evolizer Famix importer
- Evolizer version control
- ChangeDistiller
- PASWStatistics

Results: This study provides the starting point for interfaces related quality and impact of design violation. The result show IUC has the stronger correlation with the #SSC then the C&K. The IUC model can improve the performance of the predication models in java interfaces.

38. Monitoring Software Quality Evolution by Analyzing Deviation Trends of Modularity Views[66]

Research Question: This study based on two parts. First, proposed the technique for analyzing the degradation trends of software quality. Second part presents the empirical study that consists of the following research questions.

- Q1. What deviation trends of different modularity views does a software system show during its evolution?
- Q2. How do we understand software quality evolution by analyzing the deviation trends?
- Q3. Can we get useful feedback for evolution decisions by monitoring the deviation trends of modularity views?

Study Method

The following steps are involved for constructing this approach and conducting the empirical study.

Step 1: The following steps are involved in first step.

- Extracting the packages views by constructing the different modularity views of each version.
- Extracting structural interrelation.
- Extracting the semantic similarity.
- Making the cluster of structural interrelation and semantic similarities.
- Extracting the structural and semantic cluster.

Step2: SiMo Metrics Computation

Step 3: Analyzing deviation trends

The empirical study conducted by finding the difference between the sources file of the each version. The main focus will be on the release notes and changes logs. In last the result will be evaluated through the proposed approach.

Study unit: To answers these question, packages, line of code and source file are used as study unit.

Study Focus: This study proposed the assumption base technique for analyzing the degradation trends in software evolution.

Name, Number and type of the system: The following project are used

Table3.44: System used in study 38

S.no	Name	Number	Type
1	JFreeChart, JHotDraw, JEdit	3	Java base open source

Tools: The source code of the selected open source system is analyzed through the UMLDiff. UMLDiff finds the changes by comparing the neighbor version of any selected systems.

Result: The following observation is compiled from this study.

- "Different subject systems present different characteristics in their deviation trends of SiMo metrics."
- "Reasonable balance was achieved between the structural perspective and semantic perspective of the packages."

39. "Studying Software Evolution for Taming Software Complexity [65]"

Research Question This paper presents the software complexity in software evolution perspective. The following questions are address in this study.

- How does software complexity change over time: does it always grow, or does it decrease as well?
- How do evolution trends differ across programs? Are traditional software complexity metrics good indicators of how easy it is to change the program?
- What are some typical steps programmers take to reduce complexity, and how do these steps affect complexity metrics values?

Research Method: The following steps are include

- Downloading the public available releases of all software.
- Configuring (using configure) and preprocessed (using gcc -E) the main server in each release, excluding test programs.

- Merging all the source code into a single .c file.

Study Unit: This study used the full range of complexity metric that provided by the ASTdiff and RSM; the following complexity metric are extracted with the help of tools and used as study units.

- Cyclomatic complexity
- Interface complexity
- Mean module size
- Coupling
- Calls per function,
- Application size.

Study Focus: This main purpose of this study is to understanding the complexity pattern evolution of seven sizable open source programs.

Name, Number and type of the system

The following seven project are used

Table3.2.47: System used in study 39

S.no	Name	Number	Type
1	Samba, Sendmail, Bind, OpenSSH, SQLite, Vsftpd, Quagga	7	C based Open source

Tools: The CIL, ASTdiff and RSM are used as data extractor tools

Results: The following results are formulated for selected metrics.

- **About Module size:** The result show the module size is different for different systems
- **Calls per function:** The empirical result show the average calls per functions are decreased with the passage of time
- **Coupling:** the absolute value of coupling increase for all programs
- **Cyclomatic complexity:** Cyclomatic complexity is increased all the time in all programmers.
- **Interface complexity:** Interface complexity follows the same patterns as cyclomatic complexity had.

40. "Understanding Structural Complexity Evolution: a Quantitative Analysis [1]".

Research Question This study analyzed the source code of 5 open source software projects

- *"RQ1: How does the developer's experience within a project influence the increase or decrease in structural Complexity caused by his commits?"*
- *"RQ2: How does the size variation introduced by commits influence the increase or decrease in structural complexity?"*
- *"RQ3: How does the change diffusion introduced by commits influence the increase or decrease in structural complexity?"*

Study method: The data of selected open source system is organized online in the form of repository that is extracted by using following steps.

- Extracting the source code from repository with the help of versioning control system analyzed.
- Listing the all comments from authors
- Determination of programming language inclusion rules.
- Calculating the cyclomatic complexity of java and C

Study Unit: The following measures are used

- The variation in size of line of code
- Structural complexity
- Comments
- Changes in file

Study Focus: The basic aim of this paper is investigating those factors that can influence the evolution of structural complexity.

Name, Number and type of the system: The following five projects are used

Table3.2.47: System used in study 39

S.no	Name	Number	Type
1	Clojure, Node, Redis, Voldemort, Zeromq2	5	C,C++ and based Open source

Tools: VCS and CVS tools are used for extracting the source code repository and analizo's Tree-evolution tool is used for listing all source code trees.

Results

The following results become the part of this study.

- All of the studied factors have influence over structural complexity evolution in at least 2 regression models
- Change diffusion is most influential factor among the studies
- System growth is not necessarily associated with structural complexity increase
- Different projects are influenced by different factors.
- Factors that influence the increase of structural complexity are different from the ones that influence the decrease.

3.3. Summary

This chapter reviews the 40 primary studies of open source software evolution that done from 2000 to 2012. Based on the research focus of those studies, we categorized them into four major part : 'software evolution trends and patterns', 'evolution process support', 'evolvability and characteristics' The first category is further refined into three sub-categories: software growth, software maintenance and evolution economics, and prediction of software evolution. Most of studies done for analyzing software evolution trends over a time. Some studies give a look to maintenances effort. Some explore the level of granularities e.g., class level, file level, and module level to measure. However, this review has also shown that there are diverse interpretations of the same terms, e.g., module, lines of code, rate of growth. This review also report the mostly studies discuss the OSS evolution at source code level. However, software architectures are inevitably subject to evolution. Therefore, it is of major importance to put more focus on managing OSS evolution and assessing OSS evolvability at the software architecture level besides the code-level evolution.

Chapter 4: Results

CHAPTER 4: RESULTS

Introduction

SOFTWARE systems are continuously evolved to meet the requirements of their users. A good understanding of the evolution process followed by a software system is essential. In this study, we compute a set of different metrics by counting various architectural elements from selected enterprise resource planning systems. The basic focus of this study is on following architectural elements

- an removal ,addition and modification of packages
- an removal, addition and modification of classes

In order to understand the architectural evolution of open source software system, we extract and observe the set of metrics from each release and understand how these metrics change with the passage of time.

This rest of chapter discusses the result from empirical data as below

i. **Macro level analysis**: This section discussed the architectural change patterns on system level.
   - Packages and classes added on system level.
   - Packages and classes removed on system level.
   - Packages and classes modified on system level

ii. **Micro level analysis**, this section discussed the difference between the releases and find the architectural evolution on subsystems/modules level as follow.
   - Packages and classes added in subsystem.
   - Packages and classes removed in subsystems.
   - Packages and classes modified in subsystems.

Moreover this section also discusses the major subsystem's evolutionary nature with their dominate change factors for both packages and classes.

4.1. Adempiere: Macro Level Analysis

This section presents the result from macro level observations that are taking from the difference of two consecutive releases on system level. These consecutive releases represent through R_i and $R_{(i+1)}$, where the R_i is first release and $R_{(i+1)}$ is the second release from two subsequent releases. Δ represents the difference of changes; these changes may be removal, additions or modification of packages/classes. For example if we want to calculate the difference between release 1 and release 2 the Δ should be Δ1 for first two releases, if we go for 2 and 3 that time Δ will be Δ2 and so on. We applied this formula on all downloaded releases for counting the total number of removed, added and modified packages/classes

Initially sixteen releases of Adempiere are chosen but the difference of last two releases shoot – up exponentially. This affect the uniformity of vertical scales; so did not depicted on the presented figures of all respective figures for packages and classes. In next section will present the evolution of architectural patterns of Adempiere. These change patterns observed on system level with respect to packages and classes/interfaces.

The result from a sixteen releases of adempiere shows small changes has been occurred on package level. Few packages are removed, few numbers of packages are added and few little changes have been made within the packages. These changes put great effect on classes or interfaces. It also observes change trend is not uniformly followed during the evolutionary period of the product.

4.1.1. Adempiere: Packages Removed Trends

Figure 4.1show the trends of removed packages that have been chosen from fourteen releases. Initially sixteen releases of Adempiere are selected but the difference of last two releases shoot – up exponentially. Every Δ show on horizontal axis the difference obtained from the two subsequent releases while on vertical axis the total numbers of removed packages are enclosed. These result shows that very few packages are removed from Adempiere. The trend of removing packages is not uniform. Technically speaking, liner trends are observed in fourteen depicted releases among from all sixteen releases.

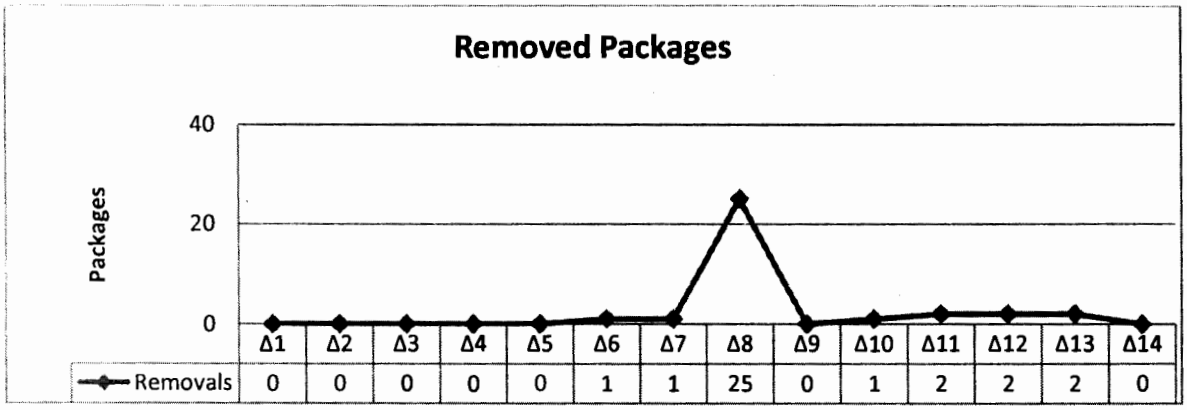


Figure 4.1: The Adempiere Packages Removed

4.1.2. Adempiere: Packages Added Trends

Maintaining the pace of large software demands the regular development. It is the difficult and complex task in which developer direct and controls the natural shape of the project. The study of Adempiere fourteen releases show that the ratio of total added packages is more than the removed one. Total number of packages that added in Adempiere are depicted in Figure 4.2 that are being obtained from the same formula that used for tracking down the pattern of removed packages

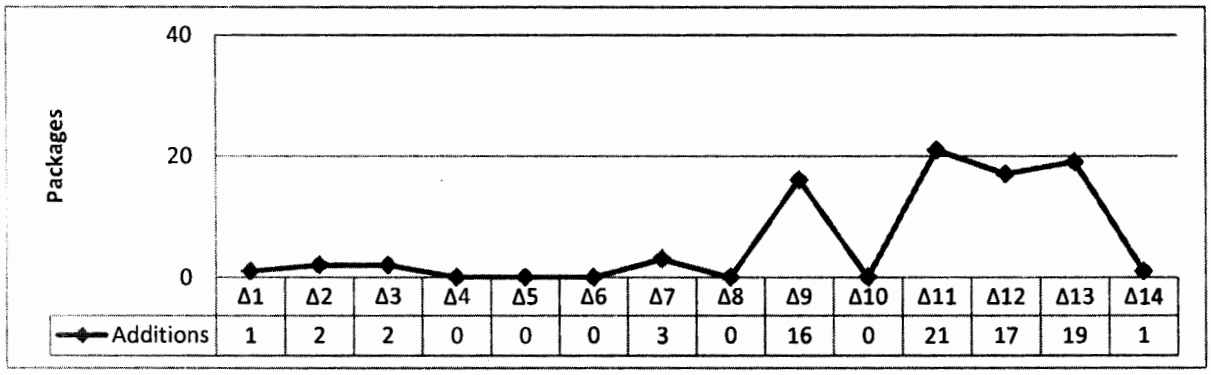


Figure 4.2: The Adempiere Packages Added

In an above depicted figure for package added during the life of software, numbers of packages that are added till release 15 is growing in a linear way, after that they suddenly change their original nature as exponential.

Moreover, a system having age more than three year and the total number of packages are being added is just between the 80 and 100, which show overall architecture of the ERP is stable.

4.1.3. Adempiere: Packages Modified

Software demands for change with the passage of time for surviving in a real world environment. As the software gets modified with new requirement it becomes complex and draft away from its original Architecture, for the cop of this problem design need to change according to new requirement. Basically architecture of the any software consists on the components and relationship between them. Packages are one of the major components of the architecture. So in This section we focus on analysis of the changed within the packages in Adempiere. Figure 4.3 shows.

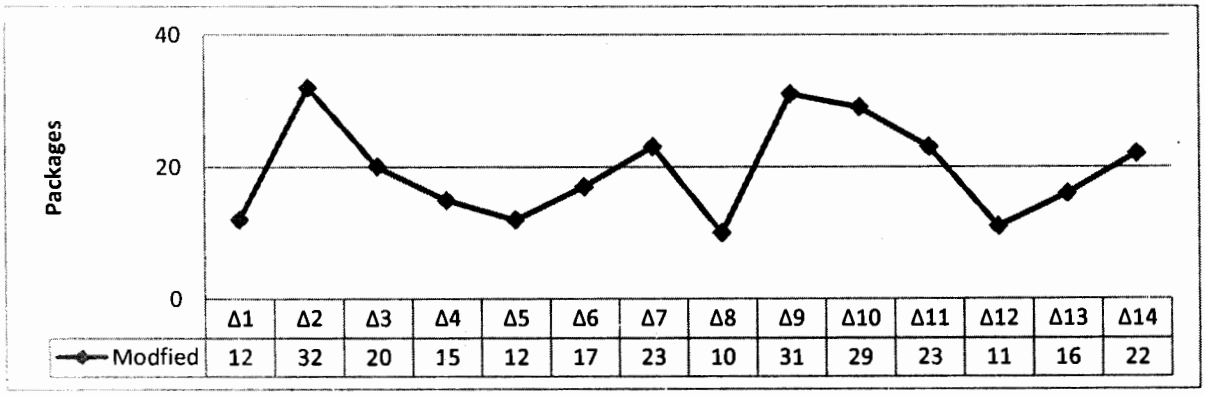


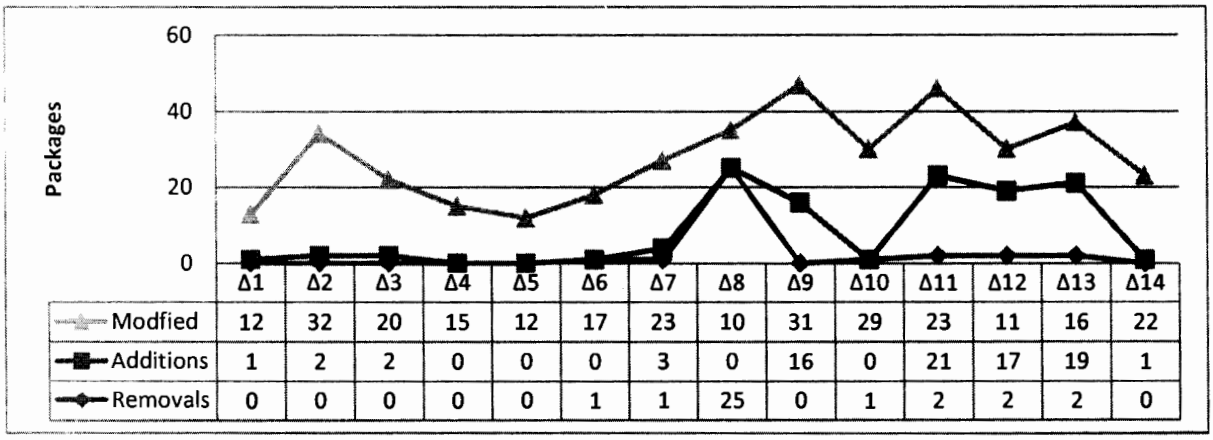
Figure 4.3: The Adempiere Modified Packages

Changed packages curve. The maximum numbers of packages change 300 to 350 in last two releases. This figure examine changes within the packages relatively more than added and deleted one packages.

4.1.4. Adempiere Packages: overall analysis

Figure 3.4 show the net number of packages added removed or changed from release 1 to 16 on an incremental basis, green line indicate the packages that are changed, red line show the total number of packages that come in a System while blue line represent the total number of

packaged that are being deleted.



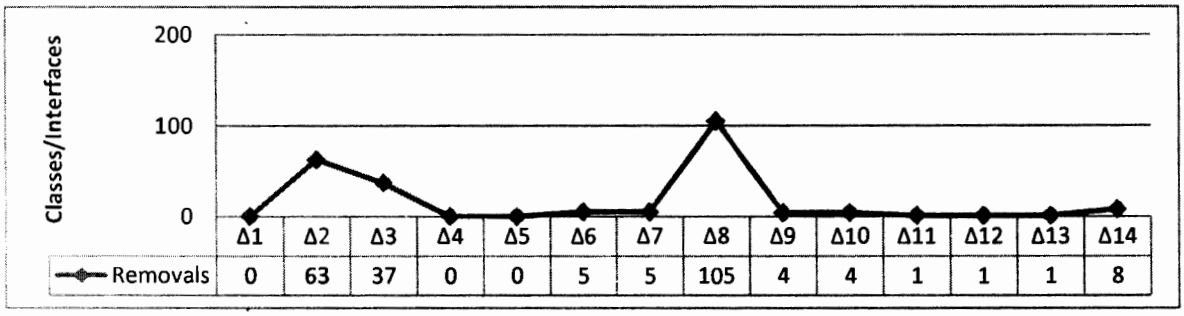
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14
Modfied	12	32	20	15	12	17	23	10	31	29	23	11	16	22
Additions	1	2	2	0	0	0	3	0	16	0	21	17	19	1
Removals	0	0	0	0	0	1	1	25	0	1	2	2	2	0

Figure 4.4: The overall Packages Trend/Pattern

With horizontal axis depicted the number of releases that are under observation and the total number of packages that are being deleted, added and changed show with vertical axis .It is noticeable from above depicted figure, almost consistent evolutionary change in Adempiere has been done except in last two or three releases . The maximum numbers that are being observed in evolutionary phenomena, about changed packages, near about 350. Here we can see less than 100 packages are added while less than 50 packages are removed from entire releases. That's indicate that the basic architecture of Adempiere business suit is stable

4.1.5. Adempiere: Classes/Interfaces Removed

This section presents the removed classes' pattern phenomena in 16 releases of Adempiere. Figure 4.5 shows the trends of classes removed in different releases of Adempiere.



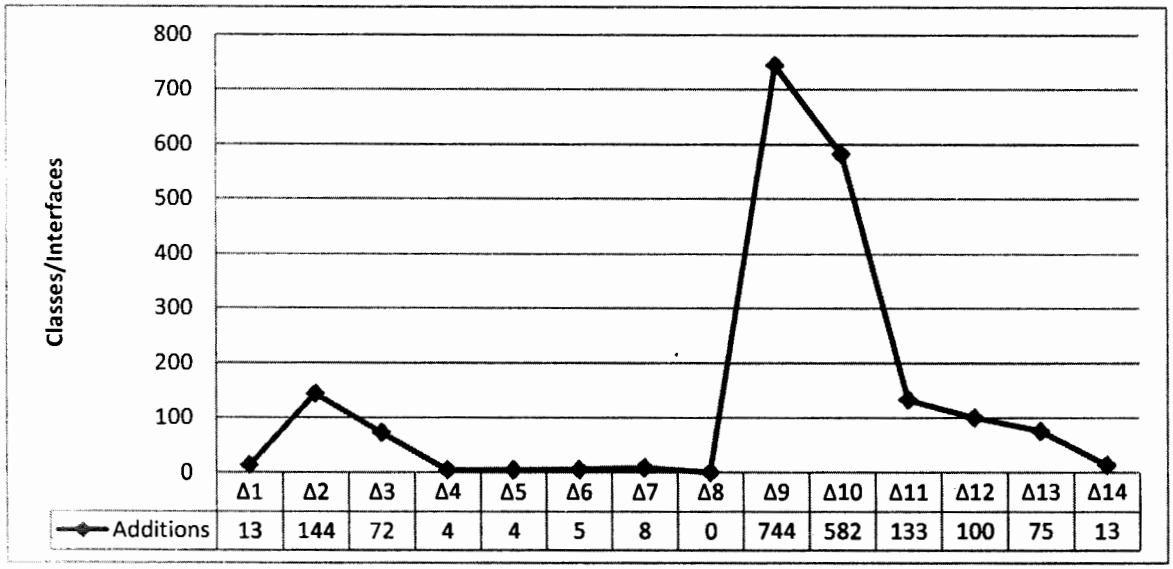
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14
Removals	0	63	37	0	0	5	5	105	4	4	1	1	1	8

Figure 4.5: Adempiere classes/interfaces Removed

Small numbers of classes are removed from every release; in this graph we can see (from release 4 to 14) removing number of classes not increasing from more than 10 classes per release. That show the consistent behavior of classes removed. But on starting and ending point of this graph it also being observed relatively more classes was deleted.

4.1.6. Adempiere: Classes/interfaces added

Addition of classes can be the another cause of changing architecture , in this section we are observing evolutionary patterns with respect to addition of classes in Adempiere release, figure 4.6 show the overall addition of classes from release 1 to 16. In which we see more classes are added as compare to removed one.



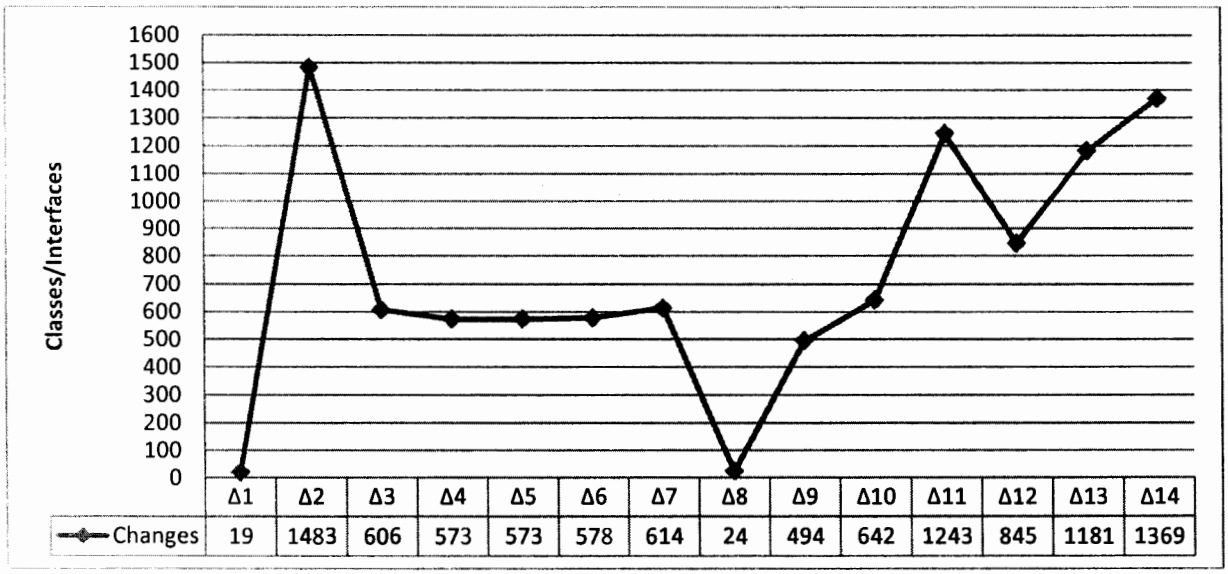
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14
Additions	13	144	72	4	4	5	8	0	744	582	133	100	75	13

Figure 4.6: Adempiere classes/interfaces added

Most of the added classes exist in release 15 and 16, it is the same point where the maximum number of the packages added, deleted and changed. Nearly 2000 classes are added in last two releases. Another major change we observe in between release number 8 and 11 where the peak of graph line touches about 700 classes.

4.1.7. Adempiere: Classes/ Interfaces Modified

Classes demands for changes with the passage of time for maintaining the pace of application. Usually within the classes or interface different number of Methods added, deleted or being changed.



	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14
Changes	19	1483	606	573	573	578	614	24	494	642	1243	845	1181	1369

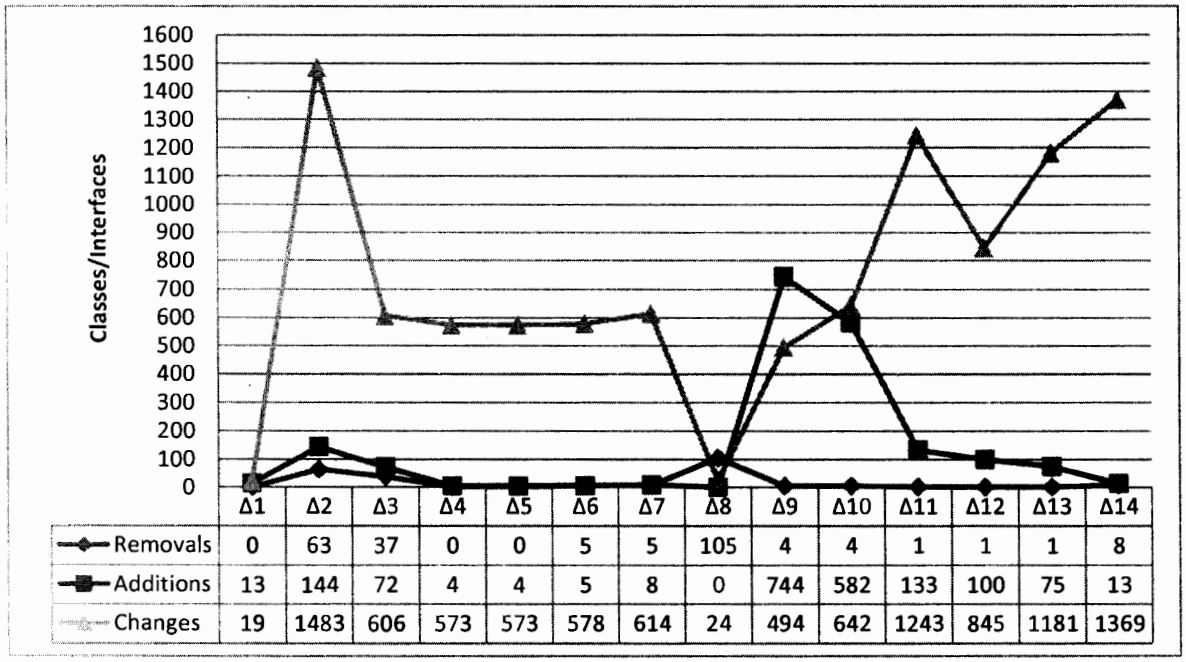
Figure 4.7: Adempiere classes/interfaces Modified

In this section we are discussing about how much classes are changed during selected releases, what are basic change patterns in classes/ interface? The answer of these questions we can obtain from figure 4.7 where thousands of classes were changed in 16 releases of Adempiere. Consistent and Linear changed behavior has been observed from release a 1 to 15 in which only edges of line maximum touching nearly 2000 classes are changed. But difference of last two releases presents the exponential curve and touching the 12000 changed classes. This indicates that the major changes occurred in last two releases

4.1.8. Adempiere Classes/interfaces: overall analysis

The stability of the any software based on the different components of architecture. Packages and classes are the basic components of the software. In literature we have found the few studies that discussed the classes and packages simultaneously or separately from evolution perspective, most of the available studies are taking size, classes' continuity or discontinuities and about

refactoring etc. One of the primary contributions of this thesis to know; how much classes are deleted in two consecutive releases? In which release these classes and interface are deleted? How much classes & interface add and when this happened? What is the total number of changed classes & interface? And in last what the change pattern is with respect to removed, added and modified classes & interfaces with respect to each other. Figure 4.8 shows the graphically represent the classes& interfaces deleted, added or changed during the 16 releases of Adempiere Business suit.



	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14
Removals	0	63	37	0	0	5	5	105	4	4	1	1	1	8
Additions	13	144	72	4	4	5	8	0	744	582	133	100	75	13
Changes	19	1483	606	573	573	578	614	24	494	642	1243	845	1181	1369

Figure 4.8: The Adempiere Classes/interface Trend pattern

During the observation of Adempiere on class level we see the different ups and down in above depicted figure, in this figure total we can see the total number of pattern of added, changed and deleted classes. major changes has been made in release 15 to 16 , where line touches the highest number 12,000 for classes changed , nearly 2000 classes were added and 200 classes was removed from the same releases . These are the topmost points for every type of Architectural change that can effect on the evolution trends.

4.2. Adempiere Evolution on Micro Level

Describing the architectural change trend of Adempiere on subsystem level is the next topic where system consist on a number of different subsystems, For example, Linux operating

system consists on system process scheduler, virtual file system, memory manager and interface for networks etc. Just like Linux, Adempiere also contain the different subsystems in which each subsystem has particular functionality, in a coming two sections my aim to analyze the packages and classes/interfaces on subsystem levels. Subsequent subsections entitle with the name of *Adempiere: packages on subsystem level* and *Adempiere: classes/interface on subsystem level.* During the data extrication phase we have collected the number packages and classes/interface that ware removed, added or changed from subsequent releases. But here we only present the total number of changes in which included removed, added and changed\updated packages and classes.

4.2.1. Adempiere Packages: Micro Level Analysis

Adempiere is one of the best and successful java base ERP that have approximately 5 year age. With the passage of time few subsystems added and few are being removed. The ratio of added subsystem is more than the ratio of deleted once. Highlighted spots in Table 3.1 show the system being removed and starting point for added system. Basically in a micro level or subsystem analysis we are interested to identify architectural growth pattern for each subsystem. Table 4.1 also shows the Total packages level changes for every sub systems of Adempiere. We find the total number of changed packages by deriving following formula.

Total Changed Packages= Removed Packages+ Added Packages+ modified packages

In micro level or on subsystem level analysis, we observe the different type of changes on subsystem, we select the each system and compare with next and current release, after that we generate the report for packages removed, added or changed for all available releases. During the micro level analysis we find few subsystems completely removed from Adempiere after some release, for example dbport is reported removed after seven releases, we have compared these subsystems for to know about package and classes changes trends or architectural growth patterns. System that removed or being added in Adempiere is summarized in Table 4.2. Where Second Colum of this table contain name of the systems while third and four column of the this table show the release number from particular system added or removed

Table 4.1: Adempiere Packages on Subsystem

S.no	Subsystem_Name	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Total
1	Base	4	1	1	4	4	2	4	1	25	17	26	19	14	25	25	182
2	Client	1	0	7	2	2	4	7	5	13	7	16	14	8	25	13	124
3	dbPort	5	7	6	4	4	6	6									38
4	Extend	0	1	0	4	1	0	0	0	1	1	3	3	0	1	1	16
5	Install		1	1	0	0	1	1	0	1	0	0	0	0	1	1	7
6	interfaces	0	1	0	0	0		2	0								3
7	JasperReports									2	1	1	0	0	2	2	8
8	JasperReportsweb APP									0	0	0	0	0	3	3	6
9	Looks	1	4	4	1	1	3	6	2	3	3	0	0	0			28
10	migration				0	0	0	1	0	0	1	0	0	1	1	1	5
11	Posteria								0	1	1	2	0	0	1	1	6
12	Print	0	2	0	0	0	0	2	2								6
13	serverApps			1	0	0	1	0	0	1	0	1	0	0	0	1	5
14	serverRoots	1	0	0	0	1	1	0	0	0	2	1	1	1	1	1	10
15	Sqlj	1	1	0	0	0	1	1	0	1	1	1	1	1	0	0	9
16	Tools		4	0	0	0	0	0	0	0	0	0	0	0	0	0	4
17	WebCM			2	0	0	0	0	0	0	0	0	0	0	0	0	2

Micro Level Analysis shows only few subsystems are get modified their architecture, "Base,client ,db port and looks " are found being most evolvable subsystem of Adempiere in which 128,112,38 and 28 packages were changed respectively. If we observe this trend release by release, we find the **unusual trend change pattern**. suppose we see the almost Linear changes pattern in a 'Base' system till release number eight where only one package was get modified, removed or added, but in a release nine about twenty five packages are changed. These figures is enough to indicate the major changes being made in the core architecture of base system.in the case of "client" most of the architecture changed linearly and it show the almost consistent change patterns in all releases .while the "DbProt" removed in seven release of Adempiere .

Table 4.2 Subsystem Removed and Added

S.NO	System Name	Added From	Removed From
1	dbPort		**Release 8**
2	Install	Release 2	
3	JasperReports	Release 2	
4	JasperReportsweb APP	Release 8	
5	Looks		Release 14
6	Interfaces		Release 9
7	Migration	Release 4	
8	Posteria	Release 7	
9	serverApps	Release 3	
10	Tools	Release 2	
11	WebCM	Release 3	
12	Print		Release 9

4.2.2. Adempiere Micro Level: Classes Analysis

We studied the many sub system of Adempiere to analyze the classes/interfaces evolutions, for this purpose we collected the data from about seventeen subsystems as shown in Table 4.3. Each subsystem consists on hundreds of classes and in a few systems this number reached in a thousand. In evolutionary life of subsystem classes are also being deleted, added or changed.

These changes can put effect on architecture of the any subsystem, which might be lead toward the major changes in whole system. In this section we will observe the architectural evolutionary

patterns of the adempiere subsystems with respect to classes/interfaces are changed. We also identified which subsystems are more evolvable and in which release they get modified with respect to classes

Table 4.2 shows the changes in every release for each available subsystem. Here we only show the total number of changed classes. Basically this word "*Changed*" refer toward the number of deleted classes, number of added classes and number of change/updated classes. Or more formally

Total Changed Classes/interfaces= Removed Classes+ Added Classes+ updated/changed classes

The highest number of classes that are being changed in all releases of one subsystem is 10168 that was deleted from base system. This shows again the base is most evolvable subsystem in Adempiere with respect to classes where the most of changes has been done for 'base' after eight releases. Another's most evolvable system are "dbport" with 2508, extend with 1103, client with 276 and looks with 274 changed packages are being identified. Change trend of these all subsystem is different from each other, for example Major changes in a Extend and looks are observe in a only two releases while " base", "dbport" and "client" continually changed .

Table 4.3: Adempiere Classes/interfaces on Subsystem Level

S.no	Subsystem Name	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Total
1	base	17	496	21	10	10	5	15	1	1181	1191	1331	1241	1371	1639	1639	10168
2	client	1	0	18	0	2	6	33	18	33	24	45	12	18	1639	33	1896
3	dbPort	11	762	574	15	15	564	567									2508
4	extend	0	1	0	549	549	0	0	0	0	0	0	0	0	2	2	1103
5	install		33	4	0	0	5	1	0	3	0	0	0	0	7	7	60
6	interfaces	0	8	0	0	0		2	0								10
7	JasperReports									4	0	1	0	0	3	3	11
8	JasperReportsweb APP									0	0	0	0	0	4	4	8
9	looks	0	160	88	1	1	6	6	2	9	0	1	0	0			274
10	migration				0	0	0	2	0	0	1	0	1	0	1	1	6
11	Posteria									0	1	1	4	0	2	2	10
12	print	0	38	0	0	0	0	2	3	0							43
13	serverApps	2	64	0	0	0	0	0	0	10	0	1	0	0	0	2	79
14	serverRoots	1	0	0	0	2	1	0	0	0	3	1	1	1	2	1	13
15	sqlj	1	9	0	0	0	2	1	0	2	1	1	4	1	0	0	22
16	Tools		119	0	0	0	0	0	0	0	0	0	0	0	0	0	119
17	WebCM			10	0	0	0	0	0	0	0	0	0	0	0	0	10

4.3. Adempiere: Evolution of Major Subsystem

After investigating ERPs architectural evolution on system and subsystem level, we then decided to investigate architectural growth on major subsystem level for each ERP, there are seven major subsystems for Adempiere and Apache OFBiz; while in the case of open bravo we have selected five major modules (directory hierarchy of source code hierarchy of open bravo is slightly different from others ERPs). Following are nineteen major source subsystem/module that become under the major subsystem category (1, 2, 3).

1. **Adempiere:**
    - **Base**: responsible of basic Functionality for Server and Client. The project creates a base.jar file which is included in server and client jars.
    - **Client**: Contains basic functionality for client and there usability. For example packages for custom form creation, window creation, printing etc. is major responsibility of the client subsystem.
    - **Extend**: Any extensions to the original project are made by using this subsystem. You cannot modify original classes, you have to copy the package hierarchy here then write the modified version in extend or created custom classes extend from base classes.
    - **Install**: As the name indicates it contains installation setup with three different types
        - Setup in which you give configuration details.
        - Silent setup in which the already saved configuration details are used and
        - Update to implement the changes you made.
    - **Interfaces**: It contains interface information. Such as for images files, ouicon files etc.
    - **Looks**: Concerned with the look and feel. Adempiere Look and Feel adds color capabilities as well as texture backgrounds. Adempiere Looks is a 3D extension of the Java Metal Look and Feel.
    - **Print**: there is no separate source package for print it comes inside client.

- **ServerRoots**: Contains the base functionality for the working of EJB's, web and server tasks.
- **ServerApps**: Concerned with the web module of the ERP i.e. it provides base functionality for the web based client
- **Sqlj**: As the name indicates it contains SQL queries for communicating with the database.
- **Tools**: Contains the basic library of jars used to run the project.
- **WebCM**: Contains Servlets for running the web based clients.
- **Posterita**: Poserita is an independent company that has made its own additions to Adempiere. All the additional functionality they have added is in their own package poserita.

4.3.1. Adempiere Major Subsystem: Packages Evolution

This section consist on the study of the few selected subsystem that found most evolvable, our criteria is the same as did on system and subsystem level.

Figure 4.9 shows the overall change packages trends for major adempiere subsystem on package; overall changes in packages influence from different change factor that are removed, added and modified packages ,we can see the" base" and "client" are the most evolving subsystem on package level evolution.

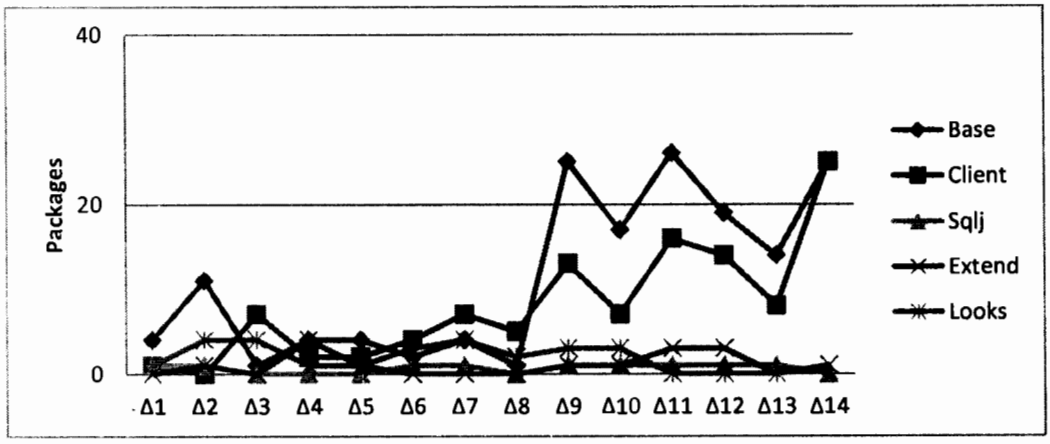


Figure 4.9: Adempiere Major Subsystem: Packages Evolution

Figure 4.9 also shows the growth rate of the Adempiere ERP others subsystems are more slowly than the base and client. Moreover this figure also presents selected subsystem are not growing steadily with respect for packages, some system(base, client) are growing speedily and some are growing slowly(sqlj, extend, looks).

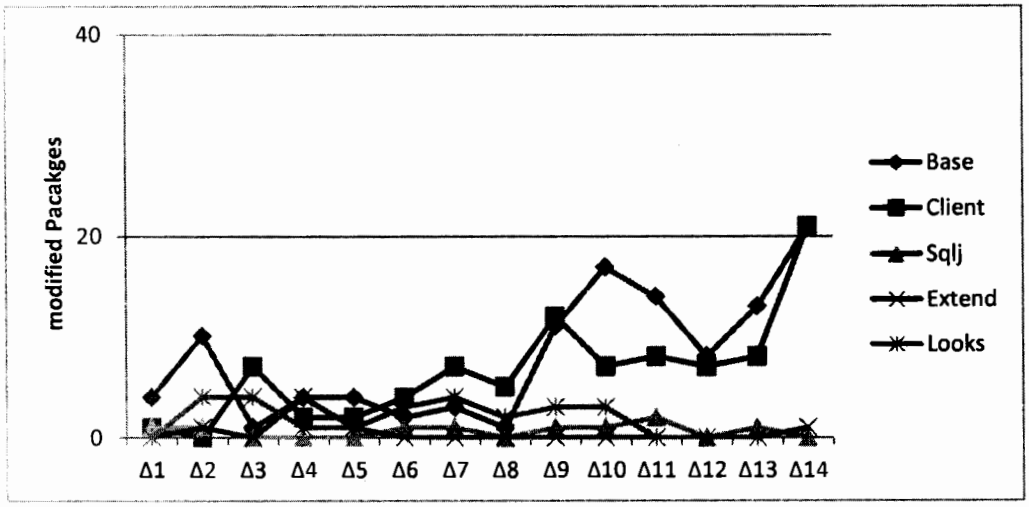


Figure 4.10 Adempiere Package: Dominant change factor

Another interesting thing that we have analyzed on subsystem level is modified/updated packages are playing as "dominant change factor" role in packages evolution, see figure 4.10 This figure present the dominant change factor patterns in Adempiere, it is about same trend that has been observed in overall change packages. We also observed major reason behind the overall change in packages is modification.

4.3.2. Adempiere Major Subsystems: Classes Evolution

Table 4.3 presents summary statistics for overall classes' changes in all release of adempiere on subsystem level. In Figure 3.11 we have examined the classes/interface change pattern of adempiere, selected system are same that are being selected for to knowing packages changes. Thousands of classes ware changed in specified period; below depicted figure present with the passage of time in Base and client system overall changes become increase,

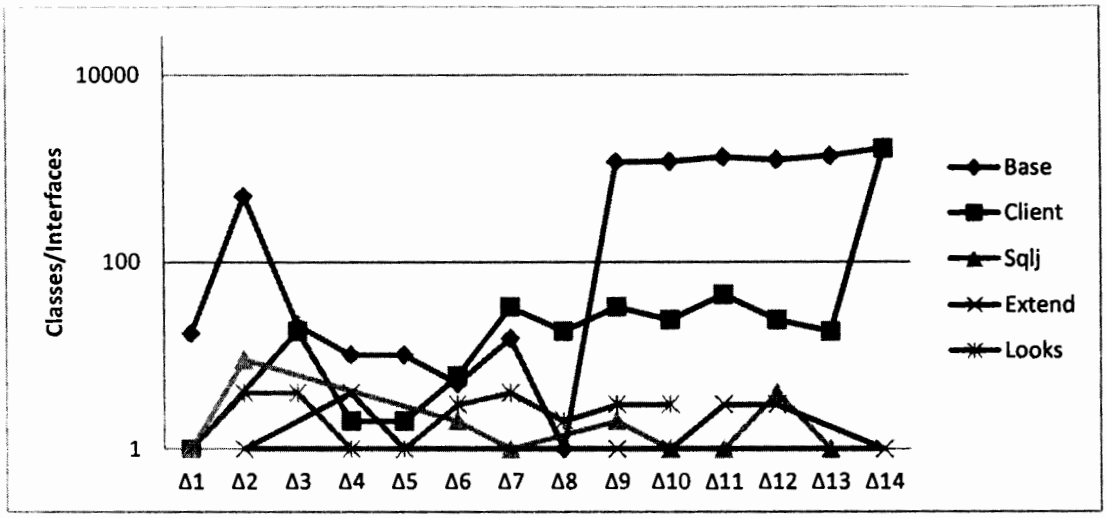


Figure 4.11: Adempiere Classes: subsystem evolution

While in another subsystem these changes decrees, moreover this figure also present changes on class level also not growing steadily as in the case of every above selected system we can see. In some system these changes occur in slowly and after specific period of time these become zero. In pervious study we already discussed the ratio of classes modified is more then removed and added classes, here we find the same pattern for adempiere major subsystem classes evolution, see the figure 4.12 that present the dominate change factor in classes, that is modification of classes, almost same change pattern is being observed that was in figure 4.11.

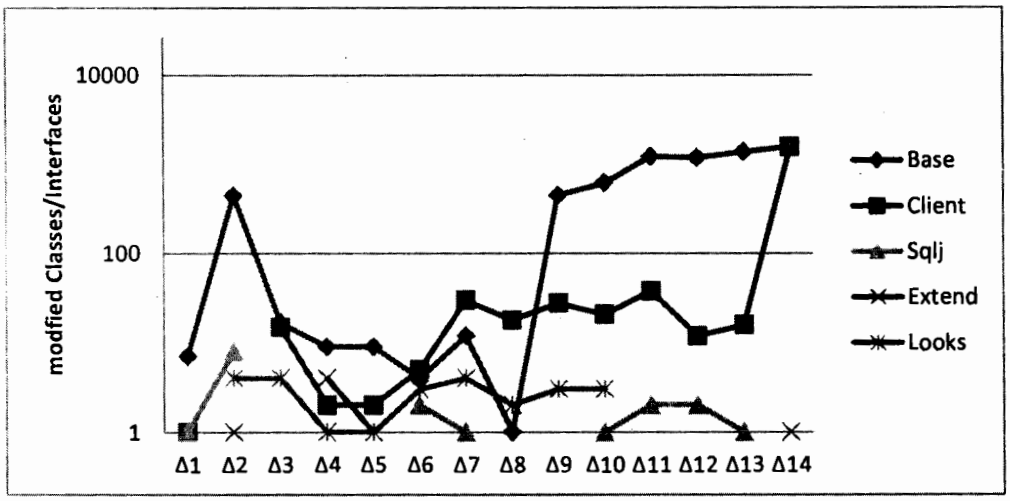


Figure 4.12: Adempiere Classes: Dominate change factor

4.4. Apache OFBiz: Macro Level Analysis

Apache OFBiz Stand for the "Apache Open for Business Project "is an open source enterprise automation software project that licensed under the Apache License Version 2.0. (1)

We select the nine releases of Apache OFBiz with 53 subsystems for finding out the Architectural patterns. Our aim to study is same as in a case of Adempiere. We compare the each release with their subsequent release. Results from this comparison provide the difference of packages, class and method for two releases which shows how many packages, classes and methods were added, removed and modified. This process, we applied on all available release .after that all acquired results classified into macro and micro analysis.

4.4.1. Apache OFBiz: Removed Packages Trends

When system evolve, it is possible old packages are get removed, in a figure 4.13 we can see where and how much packages are removed from available releases of Apache OFBiz .

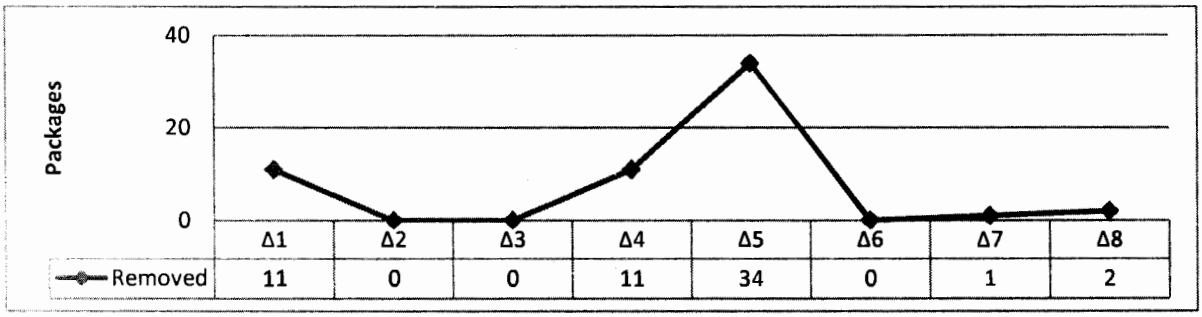


Figure 4.13: The Apache OFBiz Removed Packages

In above figure different ups and downs are visualize, that show infrequent removed packages pattern trends. Most of packages removed from release 5, where the highest total number of removed packages reached between 30 and 35. While in other releases packages not removed more than 10 packages per released.

4.4.2. Apache OFBiz: Added Packages

Another reason that makes the cause of architecture changes is the packages addition; here we are presenting the observation from our results about the packages addition in nine releases of the apache OFBiz.

In a figure 4.14 we can illustrate the trend of the added packages within the nine releases of apache OFBiz. Packages that are added in one release not more then 50, but the trend of these all packages is infrequent. Different curves are being made; ratio of added packages is almost different from the ratio of packages added in another release. Packages that added in start it is almost similar in next 2 releases that reaching near about 10 packages added per release

After release 3 the peak of curve touched the most added packages (about 49) in a single release but suddenly it goes down in a very next release(just about 2 or 3 packages). In release 6 and 7, packages start gradually increases in there total number and rise to 29 packages added in a single release. The difference of last two releases shows only 9 packages made become the part of system.

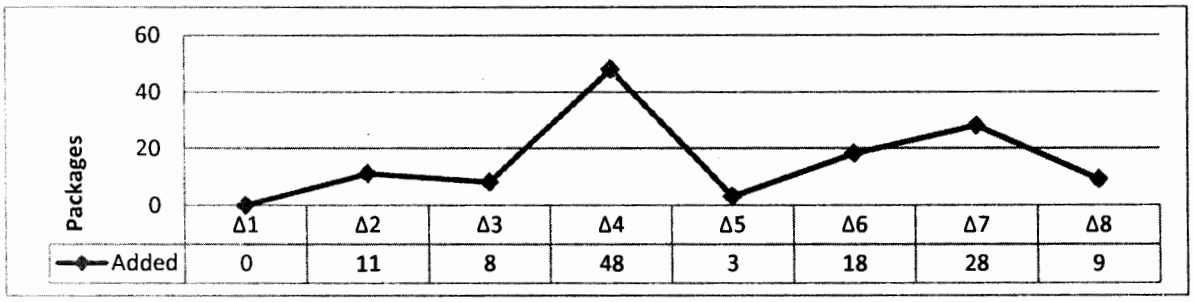


Figure 4.14: The Apache OFBiz Added Package

4.4.3. Apache OFBiz: Modified Packages

Observing the changes within the packages is the topic that we are discussing under this section. These changes refer toward the modification within the single or group of packages. Changes/modification within the packages may also effect on the core architecture of the software application.

A similar approach was followed to extract the changed packages from the apache OFBiz nine releases, which have used for removed or added packages. Figure 4.15 reports the evolution of the overall changed packages. The ratio of changed packages consistent and more than as compares to delete or added once. Growth of changed packages is increased continually and falls down on certain points as well. The figure shows that less than 80 changed packages was the start of evolutionary phenomena, near about 160 changed packages are the utmost point for

single release in apache OFBiz whereas between 100 and 120 packages is the end point of the changed packages from available releases.

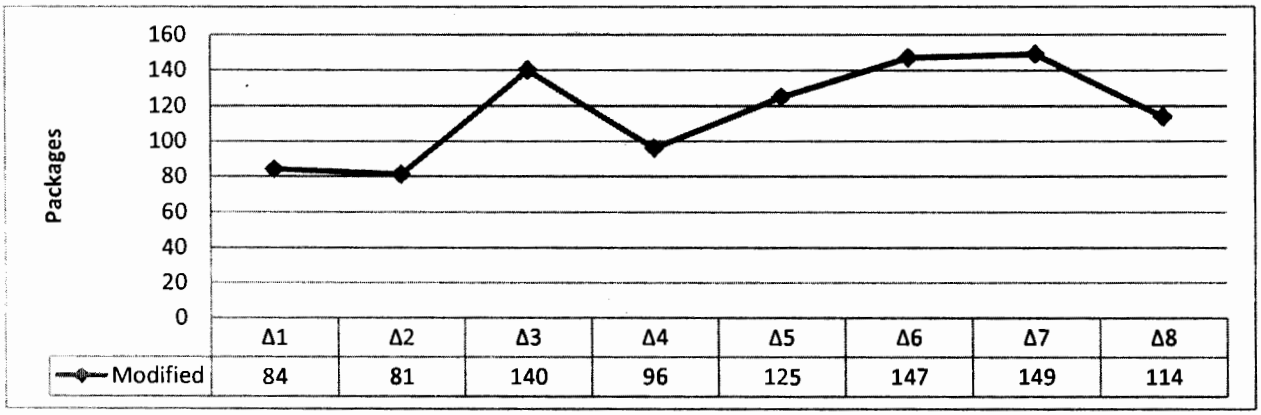


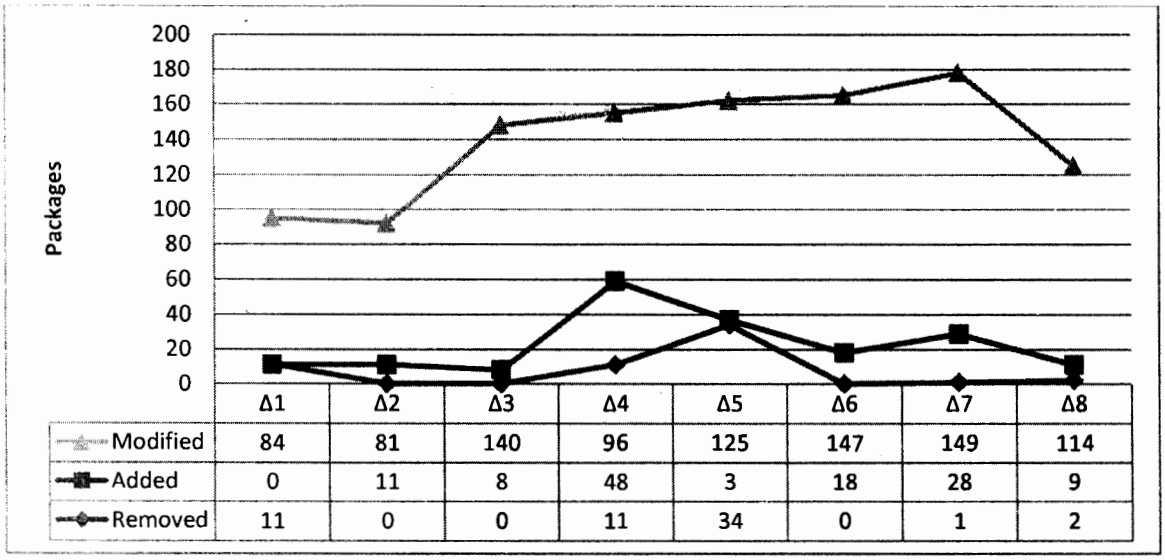
Figure 4.15: The Apache OFBiz Modified Packages

4.4.4. Apache OFBIZ Packages: Overall Analysis

In this section, the architectural evolution behavior of an open Source Enterprise Resource Planning system Apache OFBiz has been observed through detailed empirical Analysis on system level. We found that the number of packages increases gradually over most of releases in term of addition and modification, in our analysis we have mainly focus on to tracking the patterns of architecture changed packages over the evolutionary life of Apache OFBiz.

We also find that the added classes has higher ratio then removed, and modified packages is more than from both. Figure 4.16 visualize the number of packages added removed and modified across the all available releases.

We detected that the packages are continuously removed, added and modified between the releases and in most cases much more packages are modified than removed and added packages



	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8
Modified	84	81	140	96	125	147	149	114
Added	0	11	8	48	3	18	28	9
Removed	11	0	0	11	34	0	1	2

Figure 4.16: The Apache OFBiz overall Packages Trends

We can easily recognize that the number of packages increases gradually from all aspects; it may be removed, added and modified packages. We also noted most of packages removed between release 4 and 6, packages that added mostly become the part of apache OFBiz between release 3 and 5. The heuristic about modified packages of apache OFBiz can also be viewed in above figure, it start from 80 packages modified in starting releases and reached up to 140 to 150 packages change in certain releases.

4.5. Apache OFBiz: Classes/Interface

Software systems need to evolve during the software life cycle by adding new features and to improve its quality. Addition, deletion and modification of classes are also possible. Apart from many studies on software evolution that emphasize the statistical changes of the software system by analyzing its evolution metrics (3, 4, 5, and 6), little work has been done to understand the nature of software structure evolution (7) in this section we are observing the statistical changes in Classes.

4.5.1. Apache OFBiz: Removed Classes/Interface Trend

In this section we discussed the findings about classes/interface removed during the evolution of apache OFBiz life. Figure 4.13 visualize the classes removed in 9 releases of apache OFBiz.

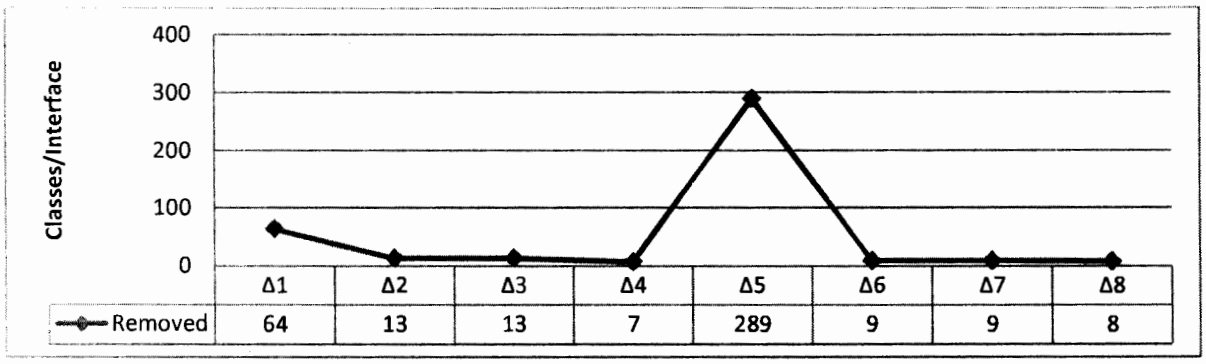


Figure 4.17: The Apache OFBiz Removed Classes/Interfaces Trend

Observing Figure 4.17, there is clear distinct between less removed and more removed classis's release. For example in start of difference of release 1 and release two about 50 classes are removed, after that figure visualize the removed classes phenomena is stable till difference of release 4 and 5. The number of removed classes touched trend presents only one major peak around difference of release 4 and 5 of the apache OFBiz's life-cycle where the classes heavily changed (300 removed classes).For last two or three releases classes removed in smooth way.

4.5.2. Apache OFBiz: Added Classes/Interface Trend

Figure 4.18 visualize the data about added classes, which extracted from the 54 subsystem and 9 different releases of the apache OFBiz with maximum and minimum number of classes/interfaces removed.

This figure also shows where the classes are added in normal or smooth way and where the trend of added classes is abnormal. Empirical data from apache OFBiz show that the growth of added classes is not consistent, for example in initial releases of apache OFBiz the ratio of added classes about 50 or 55 but after that it fall down near about 13 classes in difference five. In a very next release maximum number of classes added in the life cycle of Apache OFFBIz that reached to 200 added classes .

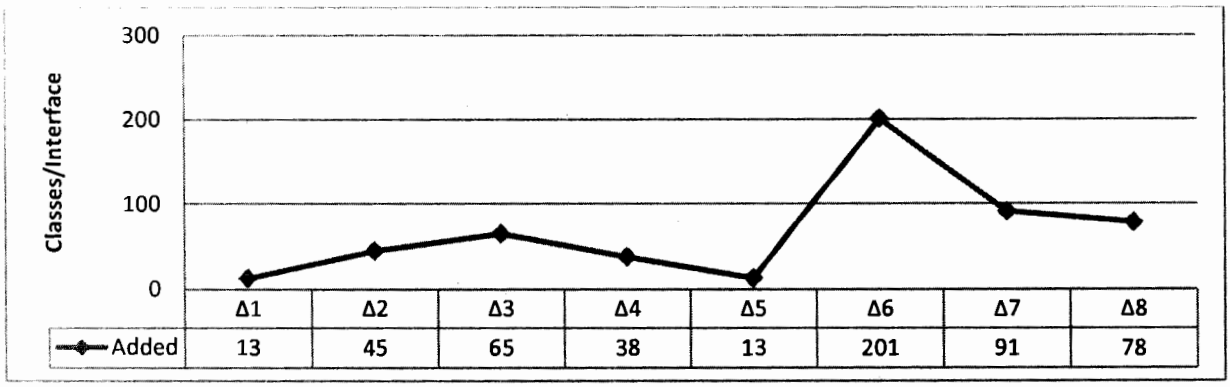


Figure 4.18: The Apache OFBiz Added Classes Trend

The difference of 7, 8 and 9 release also show the significant number of classes added , over all the growth of apache OFBiz for added classes/ interfaces is not consistent .

4.5.3. Apache OFBiz: Modified Classes/Interface Trend

Changes within the classes/interfaces is next evolutionary factor for our selected ERP, where we analyze the what is overall trend of change classes, in which release maximum number of classes changed and from where to where changed ratio is stable and where the change ratio is not stable Figure 4.19 visualize the overall change class's trend for Open source apache OFBiz in which we can observe what is the answer of our few questions that highlight in the start of current section.

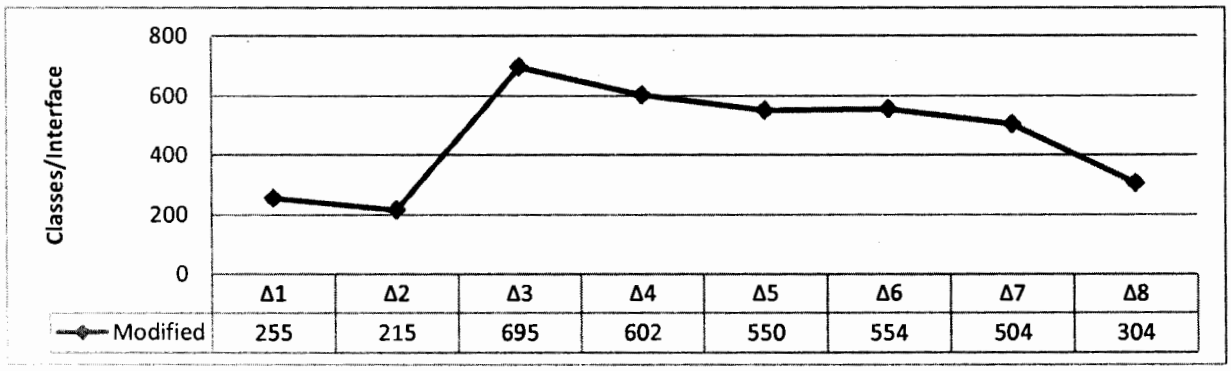
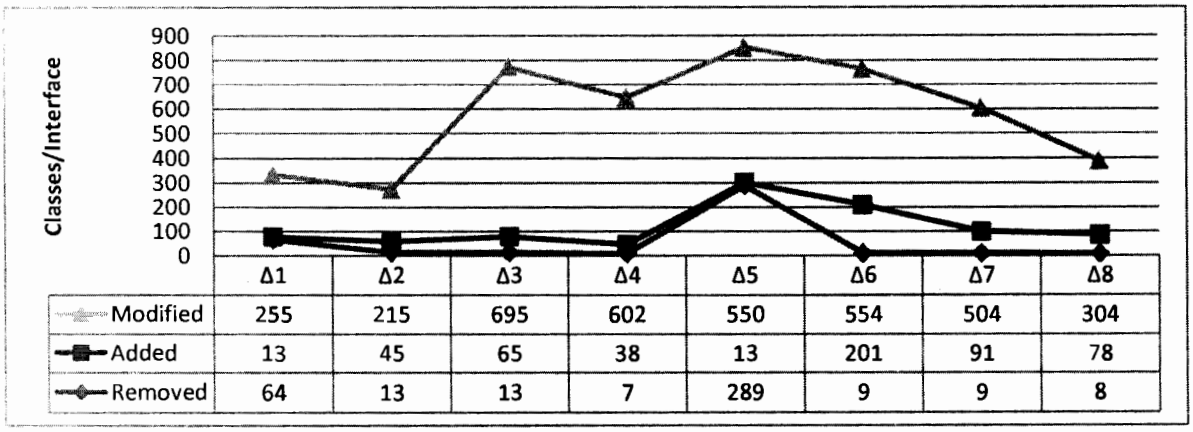


Figure 4.19: The Apache OFBiz Modified Classes/interfaces

The ratio of change/modified classes is bigger than the ration of added and removed classes. The trend of change classes also show the consistently changes in every release. The maximum number for change classes is 700(in difference of release 4 and 5) and minimum number is 200 classes changed in release 2.

4.5.4. Apache OFBiz Classes: Overall Trend

So for, i have investigated and observe the trend of deleted, added and modified classes/interfaces, within the apache OFBiz available releases.



	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8
Modified	255	215	695	602	550	554	504	304
Added	13	45	65	38	13	201	91	78
Removed	64	13	13	7	289	9	9	8

Figure 4.20: The Apache OFBiz classes: Overall Trend

Figure 4.20 visualize the all trend that has been extracted from empirical data of apache OFBiz for removed, added and modified classes. The evolutionary trend for change classes is different from the deleted and removed classes. We have investigated, it is not necessary all type of changes (removed, added and changed classes/interfaces) is occurred in same release. For example maximum classes removed between release 4 and 6, added during the release 5 and 7 and changed during the release 2.

4.6. Apache OFBiz: SubSystem Level Evolution

In this section we briefly summarize our findings with regard to evolution on subsystem level of apache OFBiz. To quantify the changes on packages or class level we compared the source code of apache OFBiz 53 subsystems on paired of consecutive releases. We divide, this section in a two subsection, first section present the data about Packages evolution on subsystem level while another section describe the evolution for classes on subsystem level of apache OFBiz.

4.6.1. Apache OFBIZ SubSystem: Package Evolution

Table 4.4 present summary statistic of 53 subsystem of OFBiz on incremental basses of total changed packages in nine major releases. These total numbers of packages derived from the following formula same in the case of Adempiere packages

Total Changed Packages= Removed Packages+ Added Packages+ updated/changed packages

Most of subsystems were the part of apache OFBiz from start, only few systems become the part of system during life cycle, the thing that discussable here in our hypothesis perspective is , to identified the most evolvable subsystem or subsystems, most stable subsystem, the most evolvable release and stable releases is.

Green highlighted spot show the most and continuously evolvable subsystem in apache OFBiz. The most evolvable subsystem is "Product" that have the 108 packages changes during the evolutionary life.

The other important evolvable systems are "Contain" with 105 , "entity" with 91, "service" with 73 , "accounting" with 68, "order "with 61 and "minilang" with 59 packages changed are identified. The interesting thing is that these all system continually changed during the every release of apache OFBiz,

Another thing that we have observe interesting is about the subsystem that have not change on packages level , these subsystem have 0 modified ,removed or deleted packages. For example "jtom"," googlechekout"," humanres" and "idap" etc , few system also being observed that have small changes with 1 or 2 packages changed during the whole life of the subsystem. Overall we have identified apache OFBiz have continually evolvable subsystem on packages level. In next section we will present the changes in apache subsystem on classes/interface level .

S.NO	Subsystem	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Total
1	accounting	7	7	9	0	13	0	15	17	68
2	appserver	0	0	1	1	1	0	0	0	3
3	assetmaint	0	0	0	1	0	0	1	1	3
4	base	1	1	1	0	0	2	21	16	42
5	bi	0	0	0	0	0	0	0	0	0
6	birt	0	0	0	0	0	0	0	1	1
7	catalina	0	0	1	1	1	0	1	0	4
8	common	1	1	3	13	5	0	8	6	37
9	content	22	22	24	4	13	1	8	11	105
11	crowd	0	0	0	0	0	0	0	1	1
12	datafile	1	1	1	1	1	0	1	0	6
13	ebay	0	0	0	0	0	0	1	1	2
14	ebaystore	0	0	0	0	0	0	0	1	1
15	ecommerce	0	0	1	0	0	0	0	0	1
16	entity	11	11	12	16	15	1	16	9	91
17	entityext	3	3	5	6	5	0	0	3	25
18	example	0	0	0	0	0	0	0	0	0
19	geronimo	0	0	0	1	1	0	1	0	3
20	googlebase	0	0	0	0	0	0	1	0	1
21	googlecheckout	0	0	0	0	0	0	0	0	0
22	guiapp	0	0	0	1	1	0	1	0	3
23	hhfacility	0	0	0	1	0	0	1	0	2
24	humanres	0	0	0	0	0	0	0	0	0
25	idap	0	0	0	0	0	0	0	0	0
26	jetty	0	0	1	1	1	0	0	0	3
27	Jtom	0	0	1	0	0	0	0	0	1
28	manufacturing	5	5	5	3	3	0	3	5	29
29	Marketing	0	0	1	4	4	0	1	2	12

30	Minerva	1	1	1	0	0	0	0	0	3
31	Minilang	10	10	11	11	11	0	5	1	59
32	Oagis	0	0	0	0	0	0	1	1	2
33	Order	7	7	9	11	10	1	7	9	61
34	Party	2	2	2	4	4	0	4	2	20
35	Pos	0	0	11	9	6	0	9	0	35
36	Product	9	9	12	23	22	0	17	16	108
37	projectmgr	0	0	0	0	0	0	1	0	1
38	Rules	0	0	0	0	0	0	0	0	0
39	Security	1	1	1	3	1	0	3	0	10
40	securityext	1	1	1	4	2	0	2	0	11
41	Service	9	9	10	14	13	0	11	7	73
42	Shark	0	0	15	0	0	0	7	0	22
43	Sql	0	0	0	0	0	0	0	2	2
44	Start	0	1	1	0	0	0	1	1	4
45	Tests	0	0	1	0	0	0	0	0	1
46	Testtools	0	0	0	3	1	0	3	1	8
47	Webapp	0	0	0	9	9	0	5	2	25
48	webslinger	0	0	0	0	0	0	2	0	2
49	Webtools	1	1	1	4	4	0	2	1	14
50	Widget	0	0	0	10	10	0	10	8	38
51	workeffort	0	0	2	3	3	0	3	0	11
52	Workflow	4	0	4	0	0	0	4	0	12
53	webPos	0	0	0	0	0	0	1	0	1

Table 4.4: Apache OFBiz Packages Subsystem Analysis

4.6.2. Apache OFBiz Classes: Subsystem Level Analysis

Classes/interface are used to promote the extensibility and reuse in object oriented system, understanding the evolvement pattern classes on subsystem level is the main focus of this section

Table 4.5 presents summary statistics for incremental changes of classes\interfaces in the 53 sub-systems of apache OFBiz. By change we mean, classes removed, classes added or classes updated in evolutionary life of the system. In this section we explore, how and where the classes are added, removed and updated. Which subsystems are more evolvable with respect to classes, which one is stable?

The most evolvable system is "minilang" with 557 classes' changes. Other important evolvable subsystem are "widget" with 522, "entity " is also with 522 ," content" with 423 ,"service" with 361 and "product" with 207.These subsystems are continually change in every release of apache OFBiz, stable subsystem are "assetmaint","bi", "birt", "crowed " ,"ebay" etc , from above table we can see the ratio of stable system are more than the ratio of evolvable system . over all architecture of the system is stable ,only in few system it is evolvable.

S.NO	Subsystem	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Total
1	accounting	9	9	14	0	16	0	21	28	97
2	appserver	0	0	1	2	2	0	0	0	5
3	assetmaint	0	0	0	1	0	0	1	1	3
4	base	3	0	0	0	0	0	86	36	122
5	bi	0	0	0	0	0	0	0	1	1
6	birt	0	0	0	0	0	0	0	1	1
7	catalina	0	0	2	4	4	0	2	0	12
8	common	4	4	8	13	13	0	10	16	68
9	content	76	20	204	43	32	2	14	32	423
11	crowd	0	0	0	0	0	0	0	1	1
12	datafile	3	3	8	6	6	0	2	0	28
13	ebay	0	0	0	0	0	0	4	2	6
14	ebaystore	0	0	0	0	0	0	0	10	10
15	ecommerce	0	0	1	0	0	0	0	0	1
16	entity	98	98	107	61	2	1	101	54	522
17	entityext	4	4	17	25	23	0	15	4	92
18	example	0	0	0	0	0	0	0	0	0
19	geronimo	0	0	0	2	1	0	1	0	4
20	googlebase	0	0	0	0	0	0	1	0	1
21	googlecheckout	0	0	0	0	0	0	0	0	0
22	guiapp	0	0	0	2	2	0	1	0	5
23	hhfacility	0	0	0	1	0	0	1	0	2
24	humanres	0	0	0	0	0	0	0	0	0
25	idap	0	0	0	0	0	0	0	0	0
26	jetty	0	0	1	1	1	0	0	0	3
27	Jtom	0	0	1	0	0	0	0	0	1
28	manufacturing	12	12	15	11	9	0	8	11	78
29	marketing	0	0	1	3	3	0	1	2	10
30	minerva	0	1	1	0	0	0	0	0	2

31	minilang	37	37	121	179	175	0	8	0	557
32	oagis	0	0	0	0	0	0	3	2	5
33	order	17	17	25	28	25	2	19	24	157
34	party	4	4	8	11	8	0	9	4	48
35	Pos	0	0	22	36	23	0	23	0	104
36	product	25	25	24	12	48	0	51	22	207
37	projectmgr	0	0	0	0	0	0	1	0	1
38	rules	0	0	0	0	0	0	0	0	0
39	security	1	1	4	3	2	0	3	0	14
40	securityext	4	4	4	1	1	0	0	0	14
41	service	29	29	63	94	83	0	49	14	361
42	shark	0	0	42	0	0	0	22	0	64
43	sql	0	0	0	0	0	0	0	49	49
44	Start	4	4	4	0	0	0	4	4	20
45	tests	0	0	5	0	0	0	0	0	5
46	testtools	0	0	0	11	10	0	3	0	24
47	webapp	0	0	0	63	65	0	12	7	147
48	webslinger	0	0	0	0	0	0	6	0	6
49	webtools	1	1	2	3	2	0	5	1	15
50	widget	0	0	0	209	188	0	63	62	522
51	workeffort	0	0	5	27	15	0	28	2	77
52	workflow	4	0	63	0	0	0	25	0	92
53	webPos	0	0	0	0	0	0	1	0	1

Table 4.5: Apache OFBiz Classes\interface Evolutio

4.7. Apache OFBiz Major Subsystem

Apache OFBiz work under the license version 2.0, it provides the comprehensive set of feature for medium to large scale of business, basically apache OFBiz divides in there su three type of module in which first is Framework second is applications last is special purpose system. These module consist on subsystem for different type of operation, we have selected framework subsystem as major subsystem because it provides the base for different subsystem, in this section we focus on architecture evolution pattern for subsystem level. It been highlighted from packages, classes and dominate factor perspective. Following are the subsystem with description that consists on framework module.

- Base
- Entity

Entities in OFBiz, or rather in databases, are the basic units of a Model in the MVC framework. Simply speaking, an entity is a single database table. A table contains information about an entity in the real world, such as a person

- Minilang
- Security
- Data file
- Service
- Content

4.7.1. Apache OFBiz Major Subsystem: Packages Evolution

Figure 4.21 shows the content, service, data file, security entity minlang and base packages evolution patterns. In this figure we can see content and service are most evolving subsystem in which about more than 30 packages are removed, added or being modified. While another subsystems grow slowly with respect to content and service.

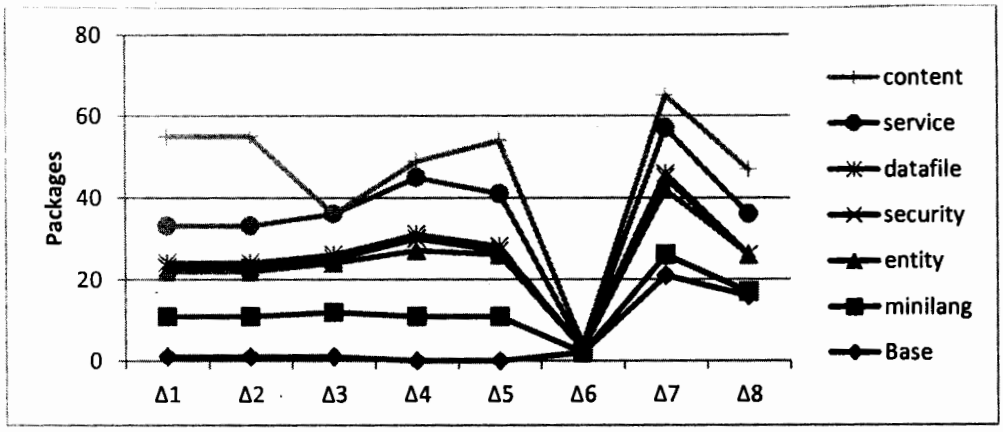


Figure 4.21: Apache OFBIZ Major Subsystem: Packages Evolution

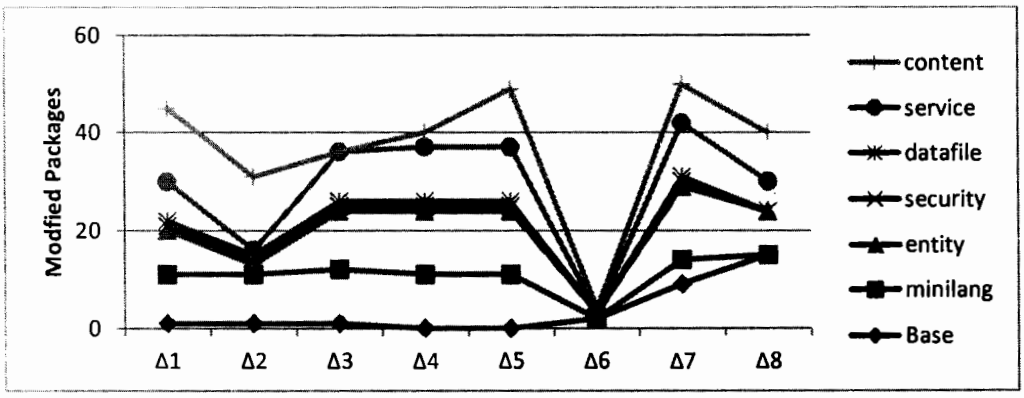


Figure 4.22: Apache OFBIZ Major Subsystem: Dominate factor

In apache OFBiz we can see some system are growing speedily and some are growing slowly. Moreover that we have find the interesting thing in the case of Adempiere about dominate change factor is modification of packages, here also the dominate change factor is modification of packages is a dominate factor, see figure 4.22

4.7.2. Apache OFBiz Major Evolving Subsystem: Classes/interface Evolution

The second major element of the large system architecture is classes, changes in classes can become cause for architectural change, figure 4.22 and 1.8 present the pattern for overall changes in classes and domain change factor in classes evolution for apache OFBiz major subs system evolution. I find a same pattern for classes as well that are in

packages, content and service overall change reached almost 400 classes in per release, content and service are found being most evolving subsystem, these changes find speedily and slowly selected subsystem of the apache OFBIZ. Figure 3.24 shows modification in classes as dominate change factor, in which we can observe clearly same pattern as for overall classes changed

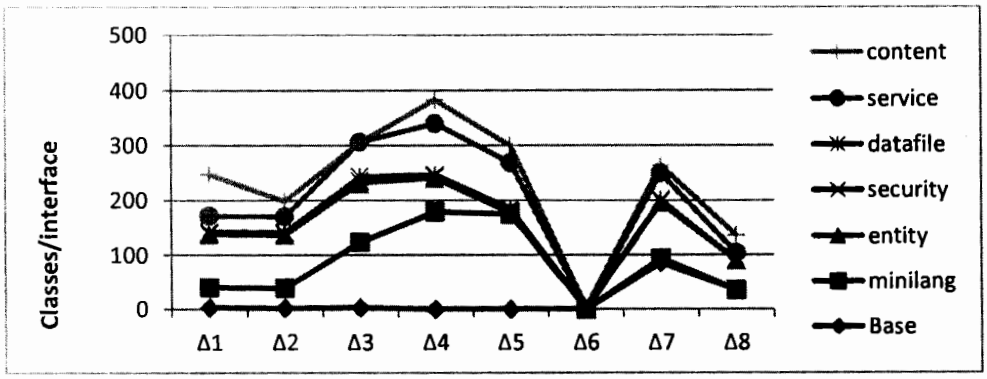


Figure 4.23: Apache OFBIZ Major Subsystem: Classes/interface

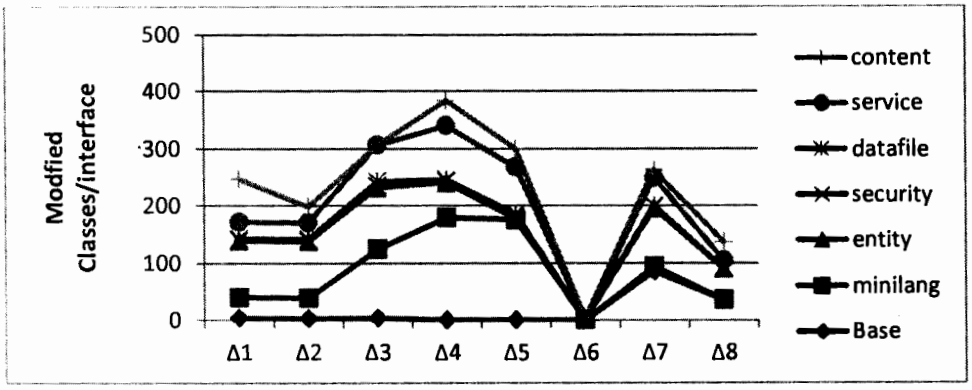


Figure 4.24: Apache OFBIZ Major Subsystem: modified classes

4.8. Open Bravo: Macro Level Analysis

Openbravo ERP Develop by open bravo company under the Openbravo Public License, this is web-based ERP Business Solution for Small and medium size company, Programming model of open bravo originally based on Compiere ERP. By using open bravo organization can be automate their common business process like Projects monitoring and control, sales and purchase process, procurement and manufacturing process etc.

4.8.1. Openbravo: Removed Packages

Sometime removal of packages effect on core architecture of the application, basically packages are removed for improvement purpose in application. Small numbers of packages are being removed during the evolutionary life of Openbravo; we have compared the source code of sixteen releases of Openbravo and observe the different trend with respect to removed packages.

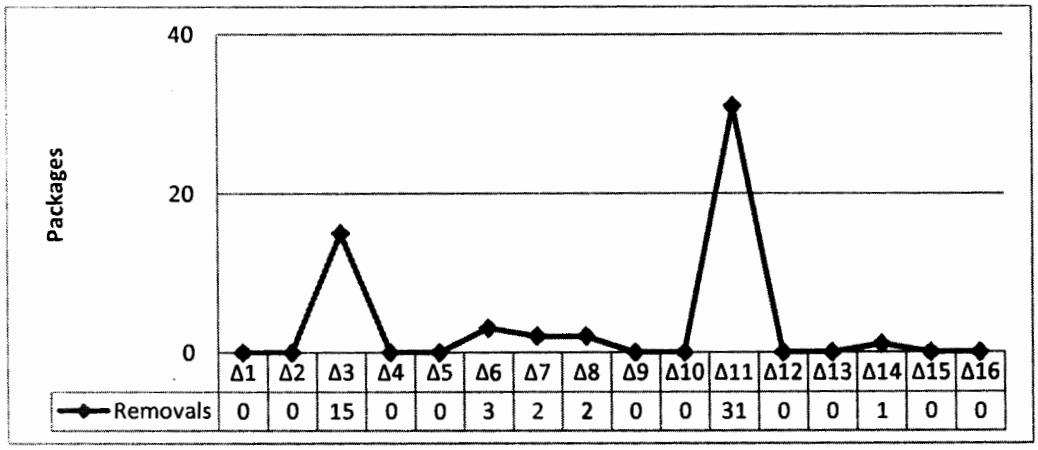


Figure 4.25: Openbravo Remove Packages Trend

If we see the total number of packages in a single release that are removed, it is not more than 35 on a system level. Total numbers of packages that being removed from 16releases that are not more than 54 packages. Packages are not continually deleted are removed from Openbravo system. We can't see the Linear curve for packages removed in all releases.

4.8.2. Openbravo: Added Packages

Every successful application need to develop and maintain on regular bases, with the passage of time these application enhance their functionality, so the new packages come in a system. In this section we present the different packages that added in long run of Openbravo. Figure 4.26 illustrate the different packages that are added in open source Openbravo in sixteen releases.

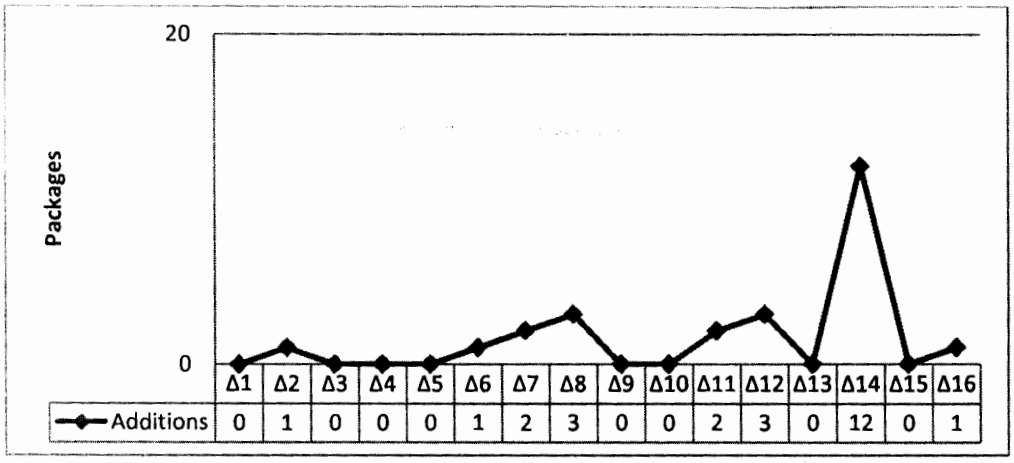


Figure 4.26: Openbravo Added Packages Trend

Interesting thing we have found about Openbravo ERP is the total numbers of added packages are less then removed packages. Total number of packages that added in a single release is not more than 12; only 25 packages are added in whole evolutionary life of Openbravo. Among of these 25 packages 12 packages are added in a single release. These packages are continually added 1, 2 or 3 packages in each release. We have also found zero packages added in a single release.

4.8.3. Openbravo: Modified Packages

Studying the changes in a packages with respect to number member of specific package add, delete or modified is our current topic of discussion. Figure 4.27 show the consistent change behavior of the Openbravo packages changes. Below depicted figure describe maximum total number of packages that changed in a single release is 29, total number of packages that changed in all releases are about 177, each release have few change packages that change. Sometime these changed packages are more than as compare to previous one

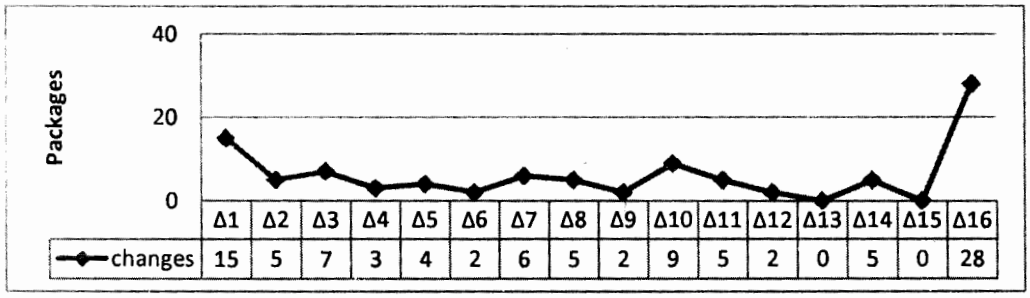
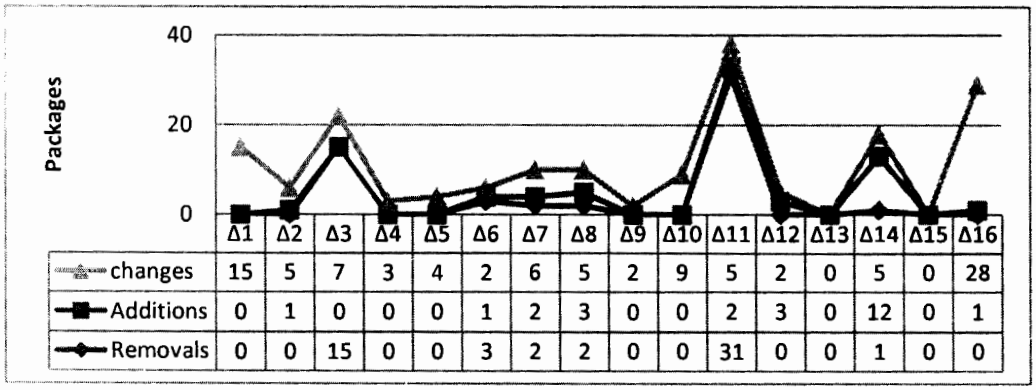


Figure4.27: Open bravo Modified Packages Trend

4.8.4. Openbravo Packages: Overall Analysis

We are conceding our study of architectural patterns identification by using Openbravo sixteen releases on package level, in this section our work is going toward the identification of architectural patterns with respect to changed packages.



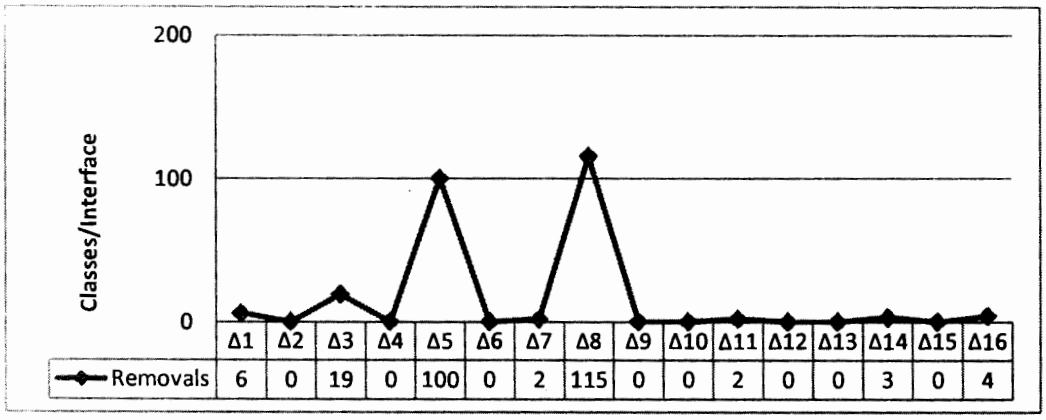
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Δ16
changes	15	5	7	3	4	2	6	5	2	9	5	2	0	5	0	28
Additions	0	1	0	0	0	1	2	3	0	0	2	3	0	12	0	1
Removals	0	0	15	0	0	3	2	2	0	0	31	0	0	1	0	0

Figure4.28: Openbravo Packages: Overall Analysis

Figure 4.28 shows the overall changes in packages trends, blue line show the packages remove, red line indicate the addition of packages in sixteen releases, while the green line point out the changes within the packages. In this ERP we only see the ratio of removed packages are more than the deleted once and added once. Trend is not Linear or consistent for all indicated line.

4.8.5. Openbravo: Removed Classes/Interface

This section present the changes in classes or interface by using the Openbravo releases, figure 4.29 depicted the overall removed classes in Openbravo different releases, few number of classes were removed from Openbravo.



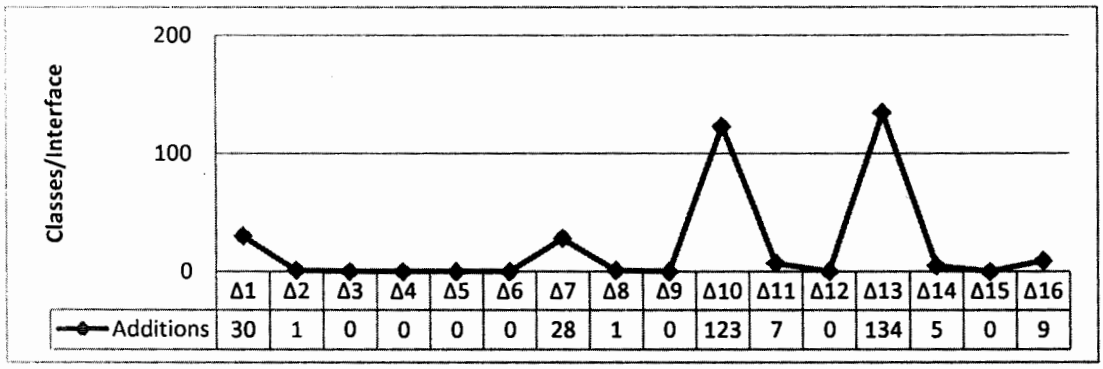
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Δ16
Removals	6	0	19	0	100	0	2	115	0	0	2	0	0	3	0	4

Figure4.29: Openbravo Classes Removed

The maximum number of classes that removed from a one release, it about 19, the trend of removed classes also not consistent in our chosen releases, decreasing trend line is inspected from removed classes of Openbravo.

4.8.6. Openbravo: Added Classes/Interface

The trend Openbravo's added classes are binge depicted in figure 4.30, which also extracted from the 16 releases.



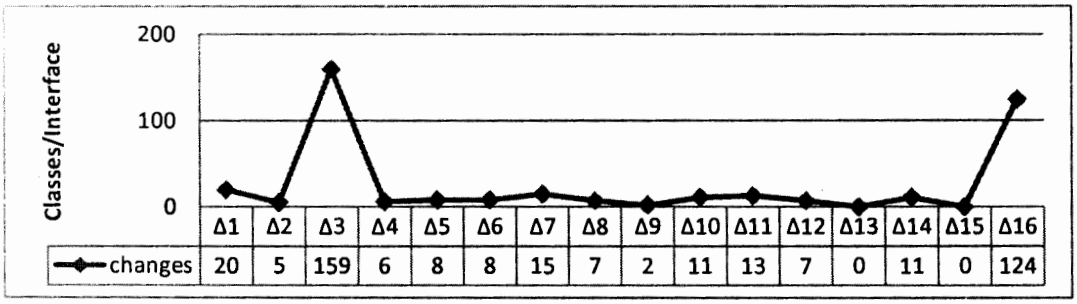
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Δ16
Additions	30	1	0	0	0	0	28	1	0	123	7	0	134	5	0	9

Figure4.30: Openbravo added classes' trend

The maximum number of classes that are added in single release of Openbravo is 30, these classes are binge added in the start of two classes, in a next four releases no class is being added, after that 28 classes are added. The overall trend is not Linear and consistent.

4.8.7. Openbravo: Changed Classes/Interface

Changes within the classes/interfaces is next evolutionary factor for identification of architectural growth pattern , where we analyze the what is overall trend of change classes, in which release maximum number of classes changed and from where to where changed ratio is stable and where the change ratio is not stable .



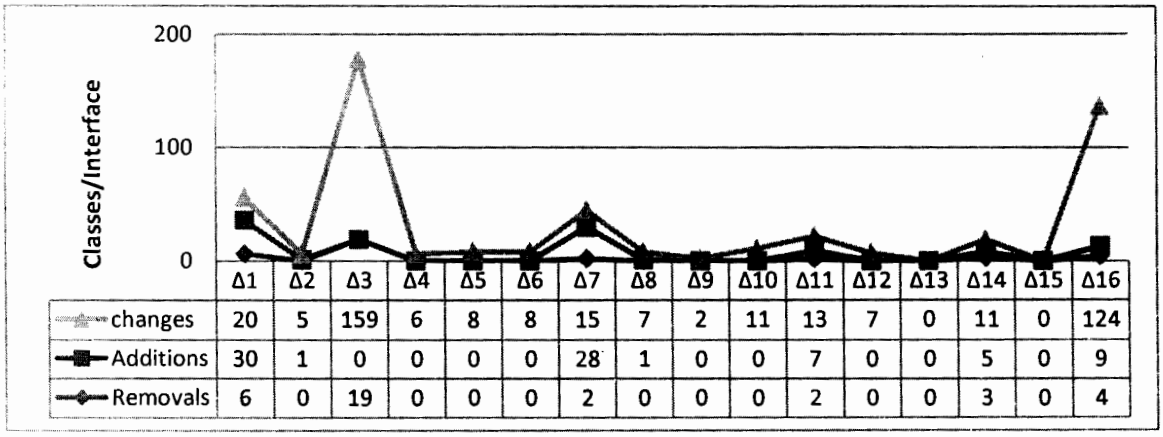
	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Δ16
changes	20	5	159	6	8	8	15	7	2	11	13	7	0	11	0	124

Figure4.31: Openbravo Classes Change

The trend of the changes classes is also not so much different from added are removed classes, only difference that has been observed the number of classes are more changed then deleted and removed one. The maximum number of classes that changed in single releases is 160. The overall change pattern is not linear or consistent.

4.8.8. Openbravo Classes/Interface: Overall Analysis

Classes removed, classes added or Changes within the classes are under discussion in this section. The numbers of removed added and deleted classes are significant low in Openbravo ERP than Adempiere and apache OFBiz. The trend pattern graph of removed, added and deleted classes is also slightly different from



	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10	Δ11	Δ12	Δ13	Δ14	Δ15	Δ16
changes	20	5	159	6	8	8	15	7	2	11	13	7	0	11	0	124
Additions	30	1	0	0	0	0	28	1	0	0	7	0	0	5	0	9
Removals	6	0	19	0	0	0	2	0	0	0	2	0	0	3	0	4

Figure 3.32: Openbravo Classes/interface overall Analysis

Adempiere and apache OFBiz. Figure 4.32 show the overall classes evolutionary phenomena for open bravo. We can see clearly the difference between the change classes and added or removed one. Total numbers for removed and added classes are under 22 for every subsequent release. Classes are mostly changed and added between release 2, 4 and 15(changed) while mostly removed in 6 and 8.we have also observe the change trend of open bravo packages is clearly different then Openbravo classes change trend.

4.9. Openbravo Evolution on Subsystem Level

We studied the three ERP source code for finding out the Architectural Change Pattern during the life of each ERP, for this purpose we have conducted study on system and sub system level. In this section we are presenting evolution on subsystem level. The organization of open bravo code is little bit different from other two ERP, others ERP's Subsystem having code separately

rather than under the single or multiple relevant module (see Table 4.3), while the code of open bravo organized in these following 5 module "SRC","SRC CORE","SRC DB","SRC TRL" and "SRC WAD". We compered these modules with relevant module in next release; the data were analyzed on packages and class level. The same process is used that was in case of Adempiere and Apache OFBiz. We have applied these processes on all downloaded releases for each ERP from packages and classes/interfaces perspective.

4.9.1. Openbravo: Packages Evolution on Subsystem Level

To analyze packages evolving pattern on subsystem level, we have extracted sixteen releases of Openbravo and inspected the source code for finding the number of removed packages, added packages and changed packages. Table 4.6 summarizes the data for packages changed.

Releases	Src	Src-core	Src-db	Src-trl	Src-wad
Δ1					
Δ2					
Δ3					
Δ4	3			0	0
Δ5	1			0	1
Δ6	4			0	2
Δ7	3	5		0	2
Δ8	3	5		1	0
Δ9	0	1		0	1
Δ10	0	0	9	0	0
Δ11	2	6	27	1	2
Δ12	1	4	0	0	0
Δ13	0	0	0	0	0
Δ14	12	3	0	1	2
Δ15	0	0	0	0	0
Δ16	22	5	0	1	1

Table 4.6: Open Bravo Packages Evolution

Green spot show the module was not existed in specific release and first Colum shows relevant release number, we observed only two modules are most evolving. That are SRC with 22 packages changes in release 16, and SRC DB with 27 changes in release 11, other there modules

are almost stable. The overall architecture of Openbravo is stable. We have observed unique in Openbravo than rest of our selected releases is this; starting three releases do not divided in a module. Table 4.7 shows the 'Src' packages modification.

Table 4.7: Open Bravo Packages Evolution

S.No	Packages Change
Δ1	15
Δ2	6
Δ3	22

4.9.2. Openbravo: Classes/Interface Evolution on Subsystem Level

Based on the same releases of Openbravo we have also collected the information about classes change on system level. Here we getting the "change "mean classes removed, classes delete and changes within the classes.

Table 4.8: Open Bravo Classes/Interface

S.NO	Src	Src-core	Src-db	Src-trl	Src-wad
Δ1					
Δ2					
Δ3					
Δ4	6			0	0
Δ5	6			0	2
Δ6	5			0	3
Δ7	5	9		0	31
Δ8	1	5		1	1
Δ9	0	1		0	1
Δ10	0	0	36	0	0
Δ11	5	7	0	3	7
Δ12	2	5	0	0	0
Δ13	0	0	0	0	0
Δ14	7	3	0	3	6
Δ15	0	0	0	0	0
Δ16	122	8	0	4	3

Green highlighted spot indicate the relevant subsystem is not existed in cross ponding releases, we have observed three modules are most evolving on subsystem for class level changes, that are SRC with 122 maximum number of classes changed in a release 16 , SRC DB with 36 classes changed in release 10 and SRC WAD with 31 classes changed in release 7. Table 4.8 also present the changes of classes as well on subsystem level ,if we compare thee both table and analyze them, then we find release 3 most evolving release of the Openbravo for classes with approximately 178 packages changed.

Table 4.8: Open Bravo Classes Evolution

S.NO	Classes Change
Δ1	56
Δ2	6
Δ3	178

4.10. Open Bravo Subsystem

Openbravo ERP is commercial open source web based ERP that has been developed by Openbravo Company. This is suitable for small and medium sized companies; the structure of this ERP different from others two module, it provide the subsystem within the module. Following are the major modules that become part of this section.

- **Src :**". This directory contains the all source code for the components that make up the core platform for this ERP in the forms of web application and services, **Src-core:** "This module is contains the source code for following core components "XmlEngine (View)", "SQLC (Model)", "HttpBaseServlet" (Controller) and "ConnectionPool."
- **Src-wad:** This directory contains the source code for Wizard of the Application Dictionary.
- **Src-db**: The src-db directory contains the source code required to create the dbmanager.jar file with database directory with all the model data.

- **Src-trl:** This module directory contains the source code of the language translator related functionality.

4.10.1. Open Bravo Major Subsystem: Packages Evolution

As already discussed in section (Ref micro and macro) very small change has been observed in the architecture of Openbravo.

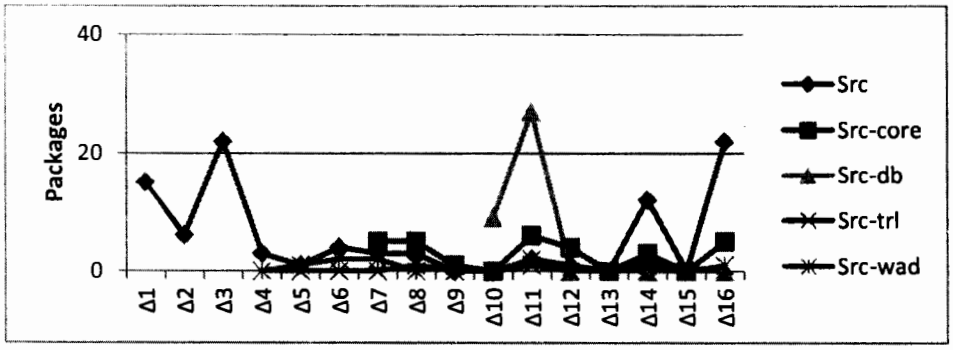


Figure 4.33: Open Bravo major Subsystem: Packages Evolution

Figure 4.33 shows the packages evolution trends in open bravo major subsystem, these changes are being found most infrequent then adempiere and apache OFFBIZ.

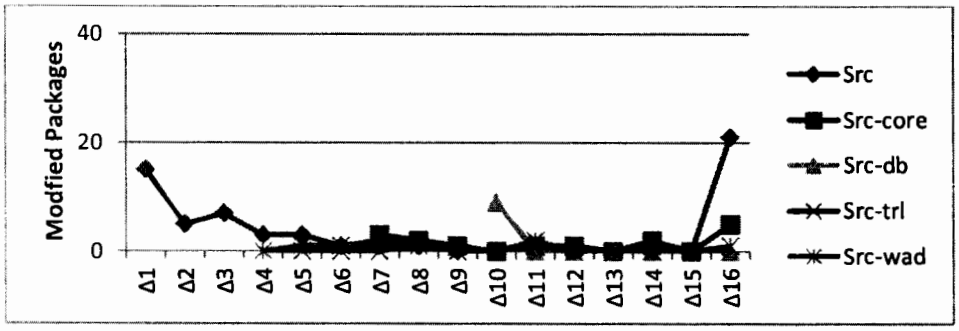


Figure 4.34: Open Bravo major Subsystem: Dominate Evolution

Most of module did not exist in initial release of apache OFBIZ; in a starting three releases there exist only one module that's called SRC, about 94 packages removed, added or modified in SRC sixteen releases in which 61 packages are changed. We found SRC as most evolving major subsystem with dominate change factor "modification in packages" another most evolving sub system is src-db with total number of 36 packages removed, added and modified, in this subsystem dominate changed factor is addition of packages has been observed. Src-core, src-trl

and src-wad are the subsystem in which very few changes being observed. Figure 4.34 present the dominate change factor for apache OFBiz major module.

4.10.2. Openbravo Major Subsystem: Classes/interface Evolution

Classes are also similar with packages evolution where few classes are modified, removed or added, these changes made infrequently in these subsystems figure 4.35 shows only the src and srcdb found the most evolving subsystem on class level.

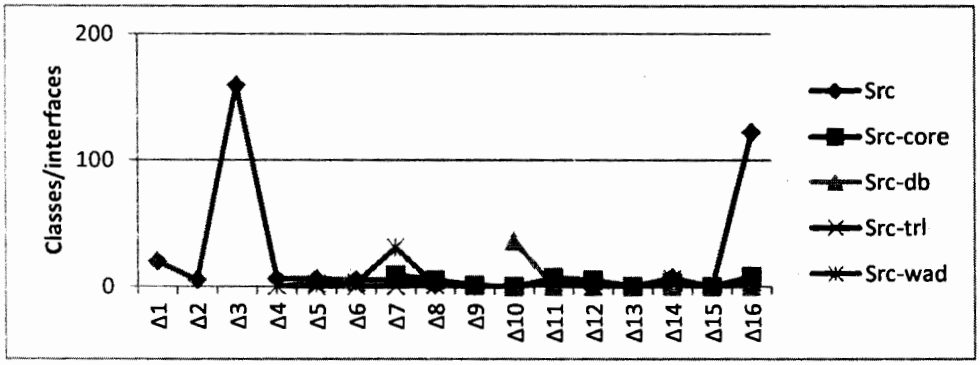


Figure 4.35: Open Bravo major Subsystem: Classes/interfaces Evolution

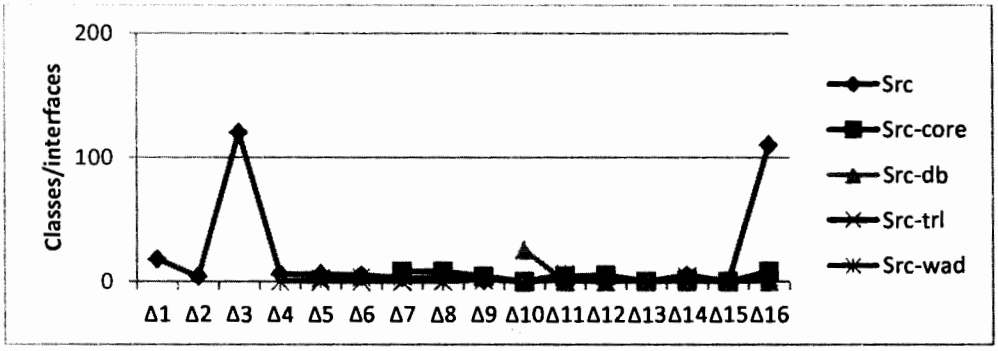


Figure 4.36: Open Bravo major Subsystem: Dominate factor

The growth trend for the architecture found Slow in some system (src-trl, src-wad,src-core), and speedily in remaining module. The dominate factor growth trends modification of the classes that detected in figure 4.36

4.11. Summary

There are several reasons that become the cause of change, usually demands from the user is one of the major origin of change in which he or she look for new emerging feature for the system. This study shows how these changes effect on the architecture of the system in term of packages and classes. Accommodating user need that are being changing in letter stage of the software system is crucial task, in our study clearly show how many packages and classes are different from one release to another release. Thousands of classes and hundreds of packages were modified, removed and added for accommodating user requirements in new version of the software system.

These findings show that maintenance effort, which is considered to be a considerable proportion of the development effort (for post initial releases), is spent on removing, adding and modifying packages/classes. The growth trend for these added removed and modified packages/classes is slightly different from system to system on both macro and micro level. Result from this study also shows the changes are localized(in some subsystem), not distributed on entire subsystem. Furthermore, the ratio of added, removed and modified packages/classes is completely different from each other. The summary of key observation from all ERP is presented below

1. Adempiere

Following are the observations that identified from adempiere

- Changes within the packages relatively more than added and deleted one
- Ratio of total added packages is more than the removed one.
- Exponential curve has been observed in last three releases
- Consistent evolutionary change observed in Adempiere on package level other than two or three releases
- Small numbers of classes are removed from initials release except in last three.
- More classes are added as compare to removed one. Most of the added classes exist in release 15 and 16,about 700 classes are added in 8 to 11,
- Nearly 2000 classes are added in last two releases
- Nearly 12000 classes were modified changed classes

Moreover, these changes are limited up to some subsystem, for example "base,client, dbpor and looks" are the systems that contain the major changes phenomena. We have investigated; the ratio of added subsystem is more than the removed subsystem

2. Apache OFBiz

Following are key observations that have been analyzed from apache ofbiz.

- Shows infrequent removed packages trends.
- Most of packages removed from release 5, where the highest total number of removed packages reached between 30 and 35.
- Packages that are added in one release not more than 50m, but the trend of these all packages is infrequent. ratio of added packages is almost different from the ratio of packages added in another release
- The ratio of changed packages consistent and more than as compares to delete or added once
- Clear distinct between less removed and more removed classis's release.
- Initially difference of removed classes is about 50 in release 1 and release 2, after that figure visualize the removed classes phenomena is stable till difference of release 4 and 5.
- Empirical data from apache OFBiz result shows that the growth of added classes is not consistent and Most of subsystems were the part of apache OFBiz from start, only few systems become the part of system during life cycle
- The most evolvable subsystem is "Product" that have the 108 packages changes during the evolutionary life
- other important evolvable systems are "Contain" with 105 , "entity" with 91, "service" with 73 , "accounting" with 68, "order "with 61 and "minilang" with 59 packages changed are identified. The interesting thing is that these all system continually changed during the every release of apache OFBiz.
- Some sub systems did not changed even single package like "jtom"," googlechekout"," humanres" and "idap" etc.

- These result also identified such subsystemsin which only one are two packages are being changed
- The most evolvable system is "minilang" with 557 classes' changes. Other important evolvable subsystem are "widget" with 522, "entity " is also with 522 ," content" with 423 ,"service" with 361 and "product" with 207.These subsystems are continually change in every release of apache OFBiz, stable subsystem are "assetmaint","bi", "birt", "crowed " ,"ebay" etc ,

3. Openbravo

Following are the findings that being observed after compering of the Openbravo releases.

- Total numbers of packages removed in all releases are not more than 54.
- Interesting thing we have found about Openbravo ERP is the total numbers of added packages are less then removed packages.
- One, two or three packages continually added in each release.
- These facts shows open bravo have stable architecture.
- The maximum number of classes that removed from a single release, it about 19, the trend of removed classes also not consistent in our chosen releases, decreasing trend line is inspected from removed classes of Openbravo.
- The maximum number of classes that are added in single release of Openbravo is 30; these classes are binge added in the start of two classes, in a next four releases no classes being added, after those 28 classes are added, The overall trend is not Linear and consistent.
- The trend of the changed classes is not much different from added are removed classes, only difference that has been observed the number of classes are more changed then deleted and removed one.
- The maximum number of classes that changed in single releases is 160

Chapter 5:

Software Evolution Tools: Reviews

Chapter 5: Software Evolution Tools Reviews

Introduction

This chapter evaluate a those tools that can help during the studies of software evolution with respect to our defined criteria, in chapter 3 number of tools being identified that are used by the researcher for evaluating their research hypothesis\questions .some tools are freeware, some are commercial and others are open source.

Usually these tools are domain specific and not fulfill the complete operational requirement of particular research. The smooth and accurate execution of research (especially empirical research) highly depends on the tools t. Mostly researcher develop their own tools for extracting empirical data, they write the scripts, customized the available tools, and removed the anomalies from available tool for obtaining results. As discussed above we have list down the different tools that used in evolutionary studies. Next section presents the review on those tools which are commonly used in open source evolutionary studies and some other tools that can be used for extracting empirical data.

The first section provides the briefly overview on the assessment criteria. Second section evaluates the tools capability according to proposed criteria and third section consisted of matrix of evaluated tools according to criteria.

5.1. Tools Assessment Criteria

The motivation behind this work is associated with current empirical study in which we are looking for tool that can extract the architectural element from source code. This assessment criterion basically extracted from the empirical studies conducting process that usually consists of four parts. First, extraction of the data elements or metrics, second visualizing extracted data in proper format. Third, performing analysis. Fourth, reporting the results. The purpose of this assessment is to evaluate the capability of software evolution related tools that may help in predicting or exploring the architecture evolution. In this section we propose a criterion that can help for assessing the capability of the tools. We analyzed the each tools with respect to extrication capability, visualizing, analyzing and platforms supportability perspective. Following section provide the brief information about our assessment criteria.

Extraction This discuss the different extrication aspect of the selected tools, we are interested to analyzed the what kind of information can be extract tool. We also assess; does the tool extract the classes, packages, and links from source code of any software? What type of changes can be found from packages, classes and links, and weather the tool have capability to find the some evolutionary pattern.

Views: This section will consist of inspecting the capability of visualization or highlighting the extracted changes. We inspect how the tools visualize the modified packages, classes and link and other extracted information? How specific tools visualize the deleted packages, classes and link? How the tools visualize the added packages, classes and link etc.

Analysis: This section address, how the selected tool model the evolutionary information? Does the information model in a way that can help to predicting the evolutionary trends/patterns?

Supportability: We evaluate each tool with respect to programming language and programming language supportability.

5.2. Evaluated Tools

This section assesses the existence tools that are mostly used in empirical studies of software evolution.

5.2.1. JFreeChart[32J

FreeChart is free java library that is used for developing wide range of charts, it is developed under the GNU Lesser General Public License (LGPL), this application widely used over the network as client and server side application, the feature that we are looking (capability of extrication the modified, deleted and added packages, classes and links) not provided this Tool. JFreeChart supports pie charts both in 2D and 3D way, horizontal and vertical bar charts, line charts, time series charts, scatter plots, high-low-open-close charts, Gantt charts, combined plots, thermometers, dials and more. JFreeChart can be used in different type of applications, applets, servlets and JSP, it can be can be used on wide range of operating system.

5.2.2. WinMerage [74]

WinMerage is an open source tool that belonged with the family of comparisons and merging tools, it can find the difference between file and folder, difference of changes can be visualize

in different forms WinMerage mostly used for "what has change between project versions , and then merging changes between versions". WinMerage support long range of operating system s such as Microsoft Windows 98, ME, 2000, XP, 2003, Vista, 2008 as well as Linux and UNIX file's Format. It is situated for comparison of file and folder but it can't find any type of difference on package or classes level.

5.2.3. KDiff3[35]

Kdiff3 used for finding out the difference line by line and character by character for two files, it can merge two or three file and directory with automatic merging facility. Kdiff3 find the difference in total number of merge file or new added file and then highlighted that files or directory by using different color codes. Kdiff3 easily work over the GNU/Linux with KDE3, MS-Windows and on apple Mac OSX But it cannot find the difference on packages or class level.

5.2.4. Araxis Merge[7]

Araxis Merge is similar to kdiff3, it can compare the two and there file at a time, and also can merge these file in a single file. Comparison difference should be on the basis of files and directory, it compare and merge "source code, web pages, XML and other text files. Additionally it can compare the binary and images file." File comparison reports can be generated in html, "html slideshow", xml or Unix diff format." It is available for Windows and MAC OS.

5.2.5. ExamDiff[23]

One of great visual comparison utility that can compare the text file, binary file and directories in very efficient way. This utility can work over on long range of windows version such as "98/Me/2000/XP/2003/Vista/7/2008." It can highlight the differences of the lines words or on character level. Feature that creates the difference of ExamDiff from other tools is "no need to specify both filenames", it can compare only taking the directory name. The analysis report will be generated in the form of HTML file. It also provides the functionality of the print and print preview of the reported difference.

5.2.6. Beyond Compare[12]

This tool allows there user to comparison support of the file, folder and directory structure of the different versions of the any specified software. It can also use for merging changes,

synchronizing files and generating reports for different record It is available for windows and Linux with supportability of different languages .it work like ExamDiff and other comparison tools; it can't provide the functionality of architectural element of any software.

5.2.7. Jhawk[34]

Jhawk is java based code analyzer tool that convert the result into CVS, HTML and interactive XML format. It visualize the different type of changes that occurred over the time, Jhawk collect the java source code and analyze for maximum and average cyclomatic complexity on system, packages and classes level. Additionally it can also find the number of Packages in system, classes by packages, Method by classes, all methods in system, all classes in system and total number of LOC. Overall this is very brilliant effort that made ten year ago but this tool cannot find the difference of packages classes and links.

5.2.8. JDiff[33]

This is a java doc that generates the HTML report on difference of two releases of packages, classes, method constructor and filed with respect to removed, changed and added. This is very useful tool to tell what is actually being changed in term of packages, classes and so on. The analysis show Diff is difficult to use for analysis of large source code, difficult to use for higher number of releases and subsystem, have not interactive and user friendly interface, can't extract the difference of links and have not visualizing capability of changes

S.No	Name	Extraction	Visualize	Analysis	Supportability
1	JFreeChart:	This tool generates the wide range of charts from different type of metrics.	Can visualize the generated charts.	Can't analyze the evolution phenomena	Portable for many type of application and system software
2	WinMerage	This tool extracts the difference between file and folder of releases.	Highlights difference only.	Can't generate any type of analysis report.	Provide portability for MS window 98, ME, 2000, XP, 2003, Vista, 2008, Linux and UNIX.
3	KDiff3	This tool can find difference of Lines and character on file and folder level .it can merge the files as well.	Only Highlights the different color code.	Can't generate any analysis report. can't help	Support the many plat form such windows and Osx .
4	Araxis Merge	It compare and merge the binary, image source code, webpages xml and other text file.	In the form of html slideshow", xml or Unix diff format.	File evaluation report can be generated in xml, html etc. can't help	Support the different version of windows operating system and Mac
5	ExamDiff	It can highlight the difference between lines of code and character of the specific software on file and folder level.	This tool can highlight the differences of the lines words or on character level.	The analysis report generated in the form of HTML. Can't help	98/Me/2000/XP/2003/Vista/7/2008.
6	Beyond Compare	This tool provide help for compering the file, folder and directory structure of the software.it can help for merging and synchronizing the different files.	This tool cant highlights the changes.	Generate the different reports. Can't help	Available for windows, Linux.
7	Jhawk	This tool find the maximum and average cyclomatic complexity ,Packages in system, classes by packages, Method by classes, all methods in system, all classes in system and Total number of LOC.	This tool can Visualize the changes.	This tool converts the result into CVS, HTML and XML format. Can't help	Available for different version of windows and Linux.
8	JDiff	This tool can find the difference of two releases for packages ,classes ,constructor ,methods and fields .	Can't provide any graphical presentation for changes.	Can't help.	Available for windows, Linux.

5.3. Summary and Motivation

This chapter assessed the capability of eight different tools that are being used in different study of software evolution. Our aim to shortlist those tools that help to extract the difference of software architectural elements, can analyze and visualize that difference of extracted element and must not be platform dependent. To the best of my knowledge we did not found any single tool that completely extract the architectural element of any software system from source code and then make the comprehensive report on his extracted data from evolutionary perspective. Only JDiff extracts the few architectural elements.

JDiff can extract the difference of Packages, classes' constructor methods and fields but can't extract the links and association between different architectural elements and also have not capability to analyze these changes or change trends of architecture of the any software.

Motivation behind building new tool comes from study of existing evolutionary data extrication tools, these all tool have limitations and cannot used stand alone. we are looking for a tool that have following features.

- Can extract the design and architectural information in term of packages removed, deleted ,modified and there connectivity links
- Can extract the design and architectural information in term of classes removed, deleted ,modified and there connectivity links
- Can visualize the Packages information with connectivity links
- Can visualize the classes information with connectivity links
- Can import the data in MS Exile, XML and HTML format
- Can visualize Evolutionary Trends with respect to
    i. Packages ,Classes, Kilobytes , Line of code

Currently undergrad students working on this idea as their final year project under the collaboration of me and my supervisor.

Chapter 6

Architecture Recovery & Trend Analysis Tool

CHAPTER 6

Architecture Recovery and Trend Analysis Tool: JSource2Design

Introduction

Software evolution is emerging area of software engineering in which researcher studies the nature of software by using different parameter and practices that make the software maintenance easy. Previous chapter describe the different tools for researcher, architect and other stakeholder that involved in software maintenance or software evolution process. Several graphical and command base tools might help to extract the information of different aspect of the software evolution, but no tool can provide the comprehensive support on single platform for extracting evolutionary information, analyzing that information and presenting the result from extracted information on a single place, especially for architectural elements.

The analysis of currently available tools shows, there is a lack of tool that can help for analyzing the architecture evolution for large/small open source product. The tool that used in this study: only find the difference of packages and classes in term of removed, added, and modified. This tool can't able to find the links evolution. Moreover it only applicable on subsystem level comparison for two releases. For solving this problem we lunched this idea as undergrad student project.

Currently two undergrad students working on this idea as their final year project under the co supervision of me and Mr. Muhammad Usman. Prototype version is ready to use in which we can extract the architectural information, can analyze the evolution trends of package, class, method, LOC and from KB. This prototype also provides the facility of architecture recovery from available source code in the form of package diagram.

The remainder of this chapter organized as follows: section 2 present the architecture of tools, section 3 visualizes the visual representation of the tool and section 4 briefly present the key use case scenario from working copy of the prototype.

6.1. Architecture

This section discusses the overall architecture and main components of the developed tool. Figure 5.1 present the high level view of the proposed tool. Major components are discussed below.

Extractor: This component is responsible for analyzing, extracting and storing the different type of information from source code on evolutionary bases, in our case these are packages, classes and there connections. This tool can extract the removed, deleted, and added packages, classes and there connection that changed over the time

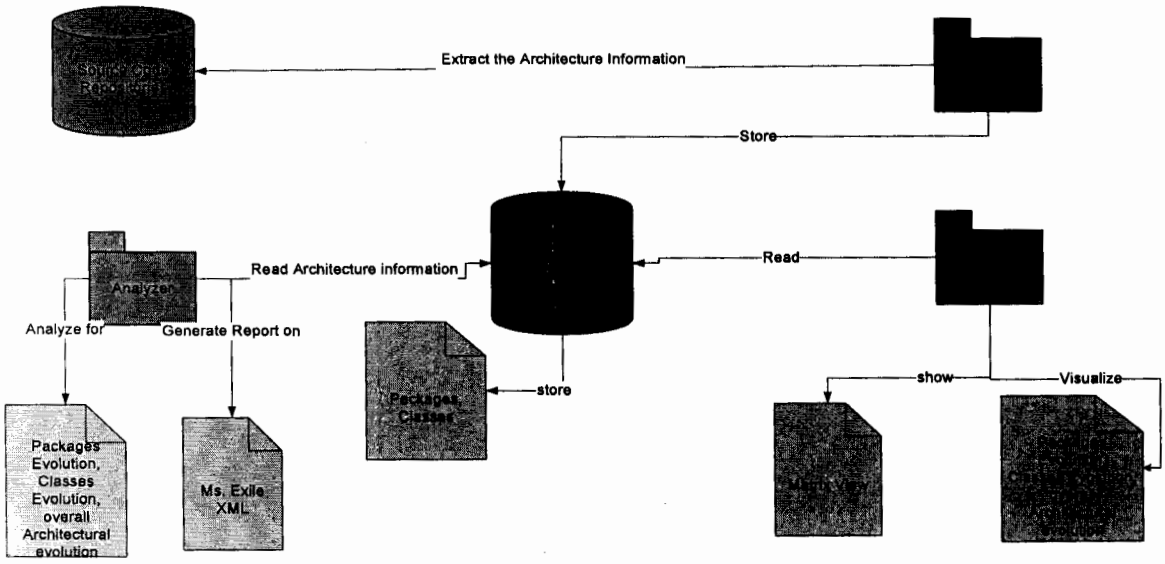


Figure 6.1:JSource2Design architecture

Analyzer read the information from architectural information data base and then analyzes the evolution patterns. It is also responsible for generating different kind of reports in Ms.Exile and XML.

Visualizer component visualize the packages, classes and overall architectural evolution trend by using different graph and matrix. It can visualize the architecture of the software in the form of package diagram by using extracted packages.

6.2. Scenario:

This section described the few use case scenarios of our developed tool. Following are the some major scenarios are proposed.

Use Case 1: Extract Design/Architecture information

1. Extracting Package Detail

i. User choose directory for "select releases" action.
ii. System show available releases.
iii. User selects desired releases.
iv. System verifies the availability of the source files.
v. Repeat following steps until releases available.
vi. User selects the "extract packages information" action.
vii. System Extract the removed, added, modified packages and there connection through "extract packages information" action.

2. Extracting Classes Detail

i. User choose directory for "select releases" action.
ii. System shows available releases.
iii. User selects desired releases.
iv. System verifies the availability of the source files.
v. Repeat following steps until releases available.
vi. User selects the "extract classes' information" action.
vii. System Extract the removed, added, modified classes and there connection through "extract classes information" action.

Use Case 2: Visualize Design /Architecture

1. Visualize Package Detail on system level

i. User choose directory for "select releases directory" action.
ii. System shows all available releases.
iii. User selects the "packages view" action.
iv. System Visualize the all packages and there connectivity links on system level.

2. Visualize classes detail on system level

i. User choose directory for "select releases directory" action.
ii. System shows all available releases.
iii. User selects the "classes view" action.
iv. System Visualize the all classes and there connectivity links on system level.

3. **Visualize Package Detail on sub system level**

i. User choose directory for "select releases directory" action.
ii. System shows all available releases.
iii. User browses the releases directory for subsystem detail.
iv. System shows the releases containing sub systems for desired release.
v. User selects the "packages view" action.
vi. System Visualize the all packages and there connectivity links on sub system level.

4. Visualize classes detail on sub system level

i. User choose directory for "select releases directory" action.
ii. System shows all available releases.
iii. User browses the releases directory for subsystem detail.
iv. System shows the releases containing sub systems for desired release.
v. User selects the "classes view" action.
vi. System Visualize the all packages and there connectivity links on sub system level

Use Case 3: Evolutionary Trend/Pattern Analysis

1. Package Evolution

i. User chooses "Evolution Trend analysis action" action.
ii. System shows the choices to display evolution in packages, classes, KB and line of code.
iii. Users select the action for "Packages evolution trend/pattern" analysis.
iv. System display the graphical presentation of
v. Packages removed trend over the period

a. Packages added trend over the period
b. Packages modified trend over the period

2. Classes Evolution

i. Users choose "Evolution Trend analysis action" action.
ii. System shows the option about packages, classes, KB and line of code option.
iii. Users select the action for "Classes evolution trend/pattern" analysis.
iv. System display the graphical presentation of .

a. Classes removed trend over the period
b. Classes added trend over the period
c. Classes modified trend over the period

3. Evolution in Kilobytes

i. Users choose "Evolution Trend analysis action" action.
ii. System show the option about packages, classes, KB and line of code option
iii. Users select the action for "Kilobytes evolution trend/pattern" analysis
iv. System display the graphical presentation of

a. Kilobytes added trend over the period
b. Kilobytes removed trend over the period

4. **Line of Code Evolution**

i. User choose "Evolution Trend analysis action" action
ii. System show the option about packages, classes, KB and line of code option
iii. Users select the action for "Line of Code evolution trend/pattern" analysis
iv. System display the graphical presentation of

a. Line of Code removed trend over the period
b. Line of Code added trend over the period
c. Line of Code modified trend over the period

6.3. Visualization

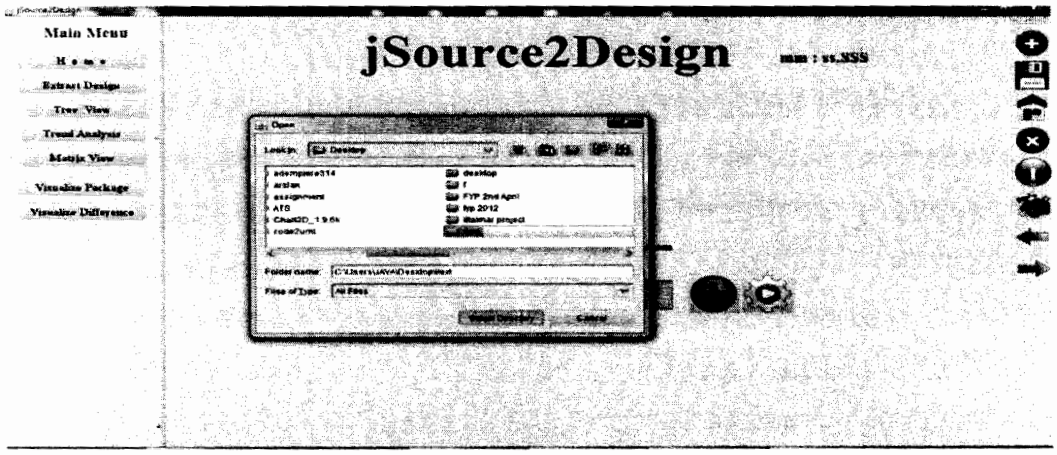
The visualization of the project structure is made up to represent the core functionality of our developed tool. Following are main actions of this software that are shown in different figures.

6.3.1. Data extraction view

Our developed tool prototype currently worked on java base source code. It has the capability to analyze the source code on system or subsystem/module level by choosing the repository from storage location. After that we extract the following information from source code and store in the data base.

- Total number of packages that are modified, added and removed
- Total number of classes that are modified, added and removed
- Total number of links between packages and total number of links classes
- Total number of Methods that are modified, added and removed
- Line of code evolution in entire system and Size of the source code files in kilobyte for each releases

Figure 6.2a and 6.2b show the data extraction view from source code repository on system and subsystem level respectively.



(a)

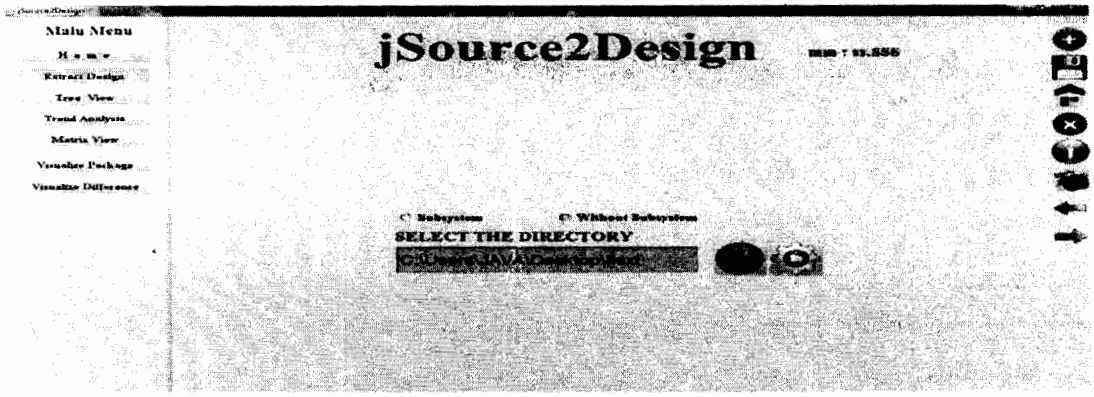


Figure 6.2: Main view of the Source Code Extraction

6.3.2. Trend analysis view

The basic purpose of this component is to analyze the trend analysis of the software evolution. It take the different evolutionary metrics from data base such as packages information, classes information, method information , line of code information and size information and generate the different graph as shown the figure 6.2a,6.2b and 6.2c. moreover it also extract the information about number of subsystem/module ,total number of packages, total number of classes , information about total package's links and classes links, size of the release in KB with total number of line of code. After extracting this information release vise it display in data grade.

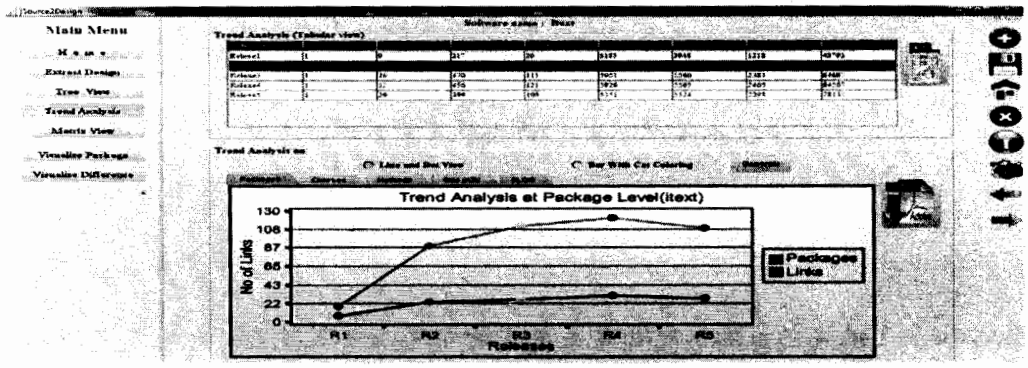


Figure 6.2 a: Packages Trend analysis

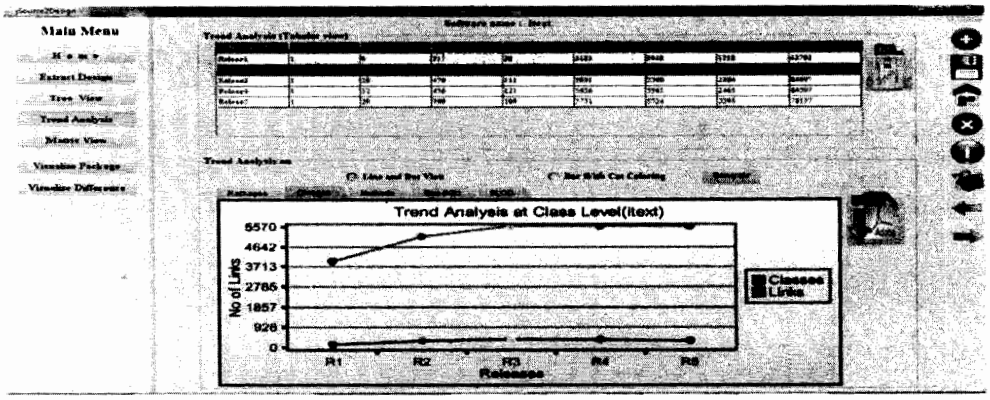


Figure 6.2b: Classes Trend analysis

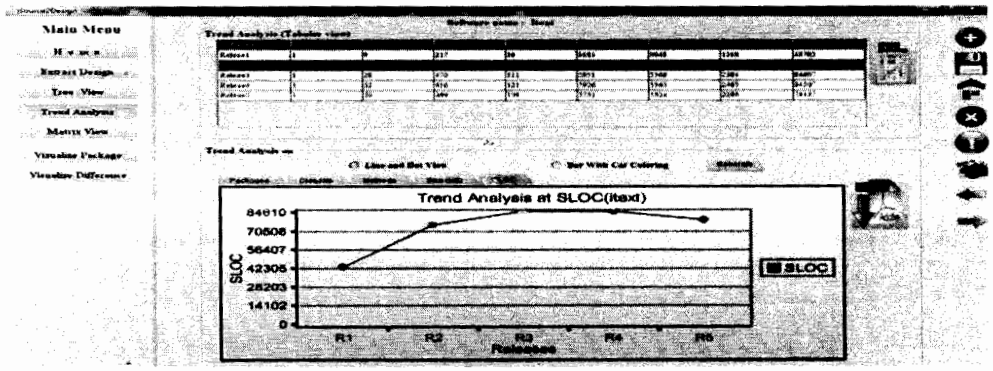


Fig 6.2c: Source Line of code trend analysis

This component also has the capability to generate the PDF and MS excel reports for software evolution.

6.3.3. Design and architecture recovery view

Another purpose of this tool is recovery of the architecture from the available source code. This tool can recover the architecture of the source code in the form of package diagram .Figure 6.3a and 6.3b present the Package view of the source code in the form of packages and matrix. Package view is representation of the linkage of package is pictorial form. Different colors box and lines are used to describe the difference

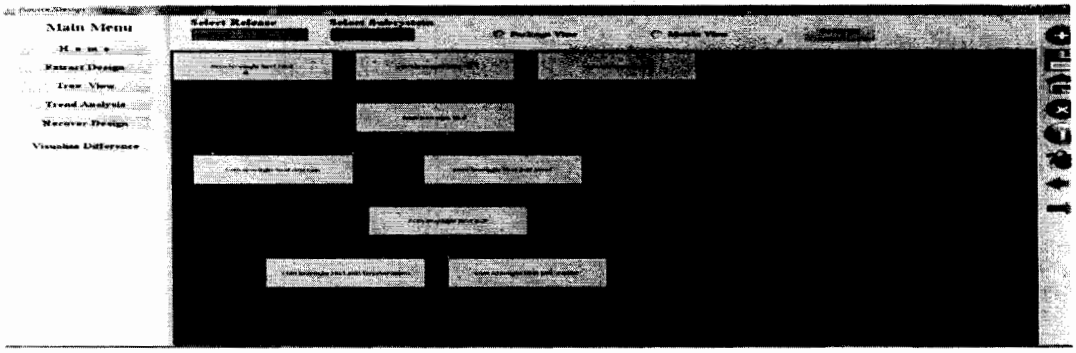


Fig 6.2b: Package diagram from source code

This matrix views shows differences of package linkages in two dimensional arrays. Numbers are assigned to each package and their cross product is presented in the view. Colors are also used to present the difference. "A"(red color box) shows that new link is added, "uc"(green color box) shows that link is unchanged, "n" shows that new possibilities added, "x" shows that no links in old version, "R" shows that removes links.

Fig 6.2b: Package Matrix from source code

6.4. Summary

This chapter present the over view of our developed tool with the help of architecture diagram and snapshot of working copy. To the best of our knowledge from existing literature there is no single tool that can extract the architecture evolution trends for packages and class. JS2D can help the researcher and architects for extract the design, architectural elements of the open source system and predicting or architecture evolution. This tool provides the following features.

- Highlight the difference of package in matrix.
- Highlight the difference of packages removed added and modified on package diagram by using different colors.
- Present the architecture evolution on system and subsystem level.
- Present the Packages evolution
- Present the classes evolution and
- Recover the architecture of software in term of package diagram.

Chapter 7

Conclusions and Future work

CHAPTER 7: CONCLUSION AND FUTURE WORK

Software evolution has been paid attention over last three decades. The result from evolutionary studies can play critical role for software maintenance. Most of software evolution research focuses on overall evolutionary nature of the system or how the particular software behaves on system or subsystem level. Little research has been reported in literature for evolution of software architecture, especially how growth and changes occur in architectural components of the software that are distributed across different parts of system. Moreover there is lack of empirical studies that provides the understanding of how evolution occurs in software architecture, how it is triggers, and how software architectural evolution is supported in software system. This research study investigates the "architectural evolution patterns/trends" in open source ERP.I have extracted architectural elements from three open sources ERPs that consist on over seventy subsystems.

6.1. Contributions

The contributions of this research thesis are:

i. **Major growth is limited to few subsystems.**

The result from this study shows major growth being made in a few subsystems, these result provide consistent support for the applicability of the [18, 27] work, in which they analyze "growth is limited to some subsystems" that make the steep spikes in growth or shrinkage however some time these spikes can be due to the addition or removal of functionality of specific part of the system.

ii. **Overall packages and classes increase.**

The results from this study show that overall size of the selected system increase in term of packages and classes; furthermore the total numbers of the subsystems are also increased in all ERPs as compared to initial releases.

iii. **Dominate change factor is "modification/updating" of the architectural components.**

One aspect of studying the evolution of software is to measure the maintenance effort in software, the result of this study show the maintenance effort put on modification of the packages/classes rather than addition or removal of packages/classes. These results show that

contradiction with [69] work in which he found most of maintenance effort spent on addition of new classes in post initial releases.

iv. Recovering Software Architecture from source code

Software architecture deals with the design and Implementation of the high-level structure of the software, currently our tool provides the facility of generating and visualizing architecture of the software in terms package diagrams from available source code.

v. Analyzing and Visualizing architectural Evolution

We provide a tool to research community that can extract the architectural information, moreover this tool having the capability of analyzing extracted information in different way for recognizing architectural evolution trends/patterns and generate reports from the evolution data.

6.2. Future Work

In future, we plan to apply our tool for studying the architectural evolution on other architectural components; For example links evolution trend in wide range of software types. We will enhance the features of tools for measuring the diverse set of evolutionary phenomena by using metrics such as complexity, method calls, message passing coupling method and inheritance. We are also planning to lunch the open source version of Js2design for research and development community.

References:

[1]. A.Terceiro, M.Mendonça, C.Chavez: "Understanding Structural Complexity Evolution: a Quantitative Analysis", Software Maintenance and Reengineering (CSMR), 16th European Conference on, vol., no., pp.85-94, 27-30 March 2012.

[2]. Ali, S.; Maqbool, O.; "Monitoring software evolution using multiple types of changes," Emerging Technologies. International Conference on , vol., no., pp.410-415, 19-20 Oct. 2009

[3]. ADempiere ERP Available online at "http://www.adempiere.com", Access on 01-10.Feb.2011

[4]. Antoniol, G.; Di Penta, M.; Merlo, E.; , "An automatic approach to identify class evolution discontinuities," Software Evolution, 7th International Workshop on Principles of , vol., no., pp. 31- 40, 6-7 Sept. 2004

[5]. Aoyama, M.: 'Metrics and analysis of software architecture evolution with discontinuity', IWPSE '02 Proceedings of the International Workshop on Principles of Software Evolution, ACM, Pages 103-107 ,ISBN:1-58113-545-9, USA, 2002

[6]. Apache OFBiz™ available at "http:// http://ofbiz.apache.org/", Access date 20-24 August 2012

[7]. Araxis Merge available at" http:// www.araxis.com/merge/", Access date 10 Oct 2011

[8]. Bass L , Paul Clements , Rick Kazman, Software architecture in practice, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1998 .

[9]. Barais, O.; Duchien, L.; Le Meur, A.-F.; , "A framework to specify incremental software architecture transformations," Software Engineering and Advanced

Application 1st EUROMICRO Conference on , vol., no., pp. 62- 69, 30 Aug.-3 Sept. 2005

[10]. Godfrey, M.W.; German, D.M.; , "The past, present, and future of software evolution," Frontiers of Software Maintenance, 2008. , vol., no., pp.129-138, Sept. 28 2008-Oct. 4 2008

[11]. Bennett, K. and Rajlich, V.: 'Software maintenance and evolution: a roadmap'. Proceedings of the Conference on the Future of Software Engineering, Pages 73 – 87. Limerick, Ireland, 2000

[12]. Beyond Compare available at " http://www.scootersoftware.com/". Access date 15 Oct 2011.

[13]. Bouktif, S.; Antoniol, G.; Merlo, E.; Neteler, M.; , "A Feedback Based Quality Assessment to Support Open Source Software Evolution: the GRASS Case Study," Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on , vol., no., pp.155-165, 24-27 Sept. 2006

[14] Brooks, F.P.: 'No Silver Bullet', IEEE Computer, pp. 10-19, 1987

[15] Capiluppi, A.; Morisio, M.; Ramil, J.F.; , "The evolution of source folder structure in actively evolved open source systems," Software Metrics,10th International Symposium on , vol., no., pp. 2- 13, 14-16 Sept. 2004.

[16]. Capiluppi, A.; Ramil, J.F.; Capiluppi, Andrea, and Juan F. Ramil. "Change Rate and Complexity in Software Evolution." Workshop on Empirical Studies of Software. 2004

[17]. Capiluppi, A.; Ramil, J.F.; , "Studying the evolution of open source systems at different levels of granularity: two case studies," Software Evolution. 7th International Workshop on Principles of , vol., no., pp. 113- 118, 6-7 Sept. 2004

[18]. Clemente Izurietaand JamesBieman.:" The Evolution of C FreeBSD andc Linux", ISESE'06, September 21–22, 2006, Rio de Janeiro, Brazil. 2006

[19]. Cook, S.; He, J.; Harrison, R.; "Dynamic and static views of software evolution," Software Maintenance, IEEE International Conference on , vol., no., pp.592-601, 2001.

[20]. D, Romano.; Pinzger, M.; , "Using source code metrics to predict change-prone Java interfaces," Software Maintenance (ICSM),27th IEEE International Conference on , vol., no., pp.303-312, 25-30 Sept. 2011

[21] Dinh-Trong, T.T.; Bieman, J.M.; , "The FreeBSD project: a replication case study of open source development," Software Engineering, IEEE Transactions on , vol.31, no.6, pp. 481- 494, June 2005

[22]. E. Merlo M. Dagenais, P. Bachand, J. S. Sormani, S. Gradara, G. Antoniol "Investigating Large Software System Evolution: the Linux Kernel", Proceedings of the 26 the Annual International Computer Software and Applications Conference (COMPSAC'02) 2002 IEEE

[23] ExamDiff available at "http://www.prestosoft.com/edp_examdiff.asp", Access date 17 Oct 2011

[24] Michael English, Jim Buckley, Tony Cahill, A replicated and refined empirical study of the use of friends in C++ software, Journal of Systems and Software, Volume 83, Issue 11, Interplay between Usability Evaluation and Software Development, November 2010, Pages 2275-2286, ISSN 0164-1212,

[25] Fernandez-Ramil, Juan; Lozano , Angela; Wermelinger, Michel and Capiluppi, Andrea. Empirical studies of open source evolution. In: Mens, Tom and Demeyer, Serge eds. Software Evolution. Berlin: Springer, pp.263–288.2008.

[26] Garlan, D.: 'Software architecture: a roadmap', ACM Press New York, NY, USA, 2000

[27]. Godfrey, M.W., Tu, Q.: Evolution in open source software: A case study. In: Proc. Int'l Conf. Software Maintenance (ICSM), Los Alamitos, California, IEEE Computer Society Press, pp. 131–142, 2000

[28]. Hassan, A.E.; Holt, R.C.; "Studying the evolution of software systems using evolutionary code extractors," Software Evolution, 7th International Workshop on Principles of , vol., no., pp. 76- 81, 6-7 Sept. 2004

[29]. Hassan, A.E.; Holt, R.C.; , "Using development history sticky notes to understand software architecture," Program Comprehension, 12th IEEE International Workshop on , vol., no., pp. 183- 192, 24-26 June 2004

[30]. Hassaine, S.; Gue´he´neuc, Y.; Hamel, S.; Antoniol, G.; , "ADvISE: Architectural Decay in Software Evolution," Software Maintenance and Reengineering (CSMR), 16th European Conference on , vol., no., pp.267-276, 27-30 March 2012

[31]. Hongyu Zhang; Hee Beng Kuan Tan; , "An Empirical Study of Class Sizes for Large Java Systems," Software Engineering Conference, APSEC 2007. 14th Asia-Pacific , vol., no., pp.230-237, 4-7 Dec. 2007

[32] Jfree Charts available at "http://www.jfree.org/jfreechart/", Access date 05 January 2011

[33] Jdiff available at" http://www.jdiff.org/", Access date 05 January 2011

[34]. Jhawk available at "http://www.virtualmachinery.com/jhawkprod.htm", Access date 10 Oct 2011

[35]. KDiff available at "http://kdiff3.sourceforge.net/", Access date 14 Oct 2011

[36]. Israeli, Dror G. Feitelson ,"The Linux kernel as a case study in software evolution", pp 15-25. 2009.

[37]. Lago, P.; Avgeriou, P.; Capilla, R.; Kruchten, P.; , "Wishes and Boundaries for a Software Architecture Knowledge Community," Software Architecture, Seventh Working IEEE/IFIP Conference on, vol., no., pp.271-274, 18-21 Feb. 2008

[38]. Lee; Jeong Yang; Chang, K.H.; , "Metrics and Evolution in Open Source Software," Quality Software, 2007. QSIC '07. Seventh International Conference on , vol., no., pp.191-197, 11-12 Oct. 2007

[39]. Lehman, M.M., Ramil, J.F., and Kahen, G.: 'Evolution as a noun and evolution as a verb', SOCE 2000 Workshop on Software and Organisation Co-evolution, pp. 12-13, 2000.

[40]. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., and Turski, W.M.:'Metrics and laws of software evolution – the nineties view', IEEE Comp Soc, Los Alamitos, CA, USA, 1997

[41]. M. F. Zibran, R. K. Saha, M. Asaduzzaman, and C. K. Roy. Analyzing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study. In ICECCS, pp. 295 – 304, 2011

[42]. M. Kim, D. Cai, and S. Kim "An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution."In ICSE' 11: 33rd International Conference on Software Engineering, 2011.

[43]. Mahmood, Ahmad Kamil, and Alan Oxley. "An evolutionary study of reusability in Open Source Software." Computer & Information Science (ICCIS), International Conference on. Vol. 2. IEEE, 2012.

[44] Madhavji, N.H., Fernandez-Ramil, J., and Perry, D.: 'Software Evolution and Feedback: Theory and Practice' John Wiley & Sons, 2006.

[45]. Medvidovic, N., Taylor, R.N., and Rosenblum, D.S.: 'An Architecture-Based Approach to Software Evolution', 1998

[46]. Melissa M. Simmons; Pam Vercellone-Smith; Phillip A. Laplante; , "Understanding Open Source Software through Software Archaeology: The Case of Nethack," Software Engineering Workshop. 30th Annual IEEE/NASA , vol., no., pp.47-58, April 2006

[47]. Mens, T.; Fernandez-Ramil, J.; Degrandsart, S.; , "The evolution of Eclipse," Software Maintenance, 2008. ICSM 2008.IEEE International Conference on , vol., no., pp.386-395, Sept. 28 2008-Oct. 4 2008

[48]. Michel Wermelinger; Yijun Yu; Lozano, A.;, "Design principles in architectural evolution: A case study," Software Maintenance,. IEEE International Conference on , vol., no., pp.396-405, Sept. 28 2008-Oct. 4 2008".

[49] McIntosh, S.; Adams, B.; Hassan, A.E.; , "The evolution of ANT build systems," Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on , vol., no., pp.42-51, 2-3 May 2010

[50]. Nasser.E, S. Counsell , M. Shepperd","Class movement and re-location: An empirical study of Java inheritance evolution", 2009

[51]. Nasseri, E.; Counsell, S.; Shepperd, M.; , "An Empirical Study of Evolution of Inheritance in Java OSS," Software Engineering,. 19th Australian Conference on , vol., no., pp.269-278, 26-28 March 2008.

[52]. Nasseri.E S. Counsell and M. Shepperd "An Empirical Study of Java System Evolution at the Method Level" Seventh ACIS International Conference on Software Engineering Research, Management and Applications.2009

[53]. Nehaniv, C.L. and Wernick, P.: 'Introduction to Software Evolvability', Third International IEEE Workshop on Software Evolvability, 2007

[54]. Nien-Lin Hsueh; Lin-Chieh Wen; Der-Hong Ting; Chu, W.; Chih-Hung Chang; Chorng-Shiuh Koong; , "An Approach for Evaluating the Effectiveness of Design

Patterns in Software Evolution," COMPSACW, 2011 IEEE 35th Annual , vol., no., pp.315-320, 18-22 July 2011

[55]. Openbravo wiki available at "http://wiki.openbravo.com/wiki/Main_Page", Access date14 July 2011.

[56]. Paulson, J.W.; Succi, G.; Eberlein, A.; , "An empirical study of open-source and closed-source software products," Software Engineering, IEEE Transactions on , vol.30, no.4, pp. 246- 256, April 2004

[57]. Paul C. Clemen, Linda M. Northrop," Software Architecture: An Executive Overview", Technical Report CMU/SEI-96-TR-003 ESC-TR-96-003, February 1996

[58]. Perry, D., Wolf, A.: "Foundations for the study of software architecture," ACM SIGSOFT Software Engineering Notes, Volume 17 , Issue 4, Pages: 40 -52, 1992.

[59]. Ratzinger, J.; Sigmund, T.; Vorburger, P.; Gall, H.; , "Mining Software Evolution to Predict Refactoring," Empirical Software Engineering and Measurement,. First International Symposium on , vol., no., pp.354-363, 20-21 Sept. 2007

[60]. Robles, G.; Amor, J.J.; Gonzalez-Barahona, J.M.; Herraiz, I.; , "Evolution and growth in large libre software projects," Principles of Software Evolution, Eighth International Workshop on , vol., no., pp. 165- 174, 5-6 Sept. 2005

[61]. Robles, G.; Gonzalez-Barahona, J.M.; Herraiz, I.; , "Evolution of the core team of developers in libre software projects," Mining Software Repositories,. 6th IEEE International Working Conference on , vol., no., pp.167-170, 16-17 May 2009

[62]. Rowe, D., Leaney, J., and Lowe, D.: 'Defining systems evolvability-taxonomy of change', Change, , pp. 541-545,1994

[63]. Shang, Jiang, Godfrey, Nasser, Flora. "An Exploratory Study of the Evolution of Communicated Information about the Execution of Large Software Systems,".pp 48, WCRE.2011

[64]. Stewart, K.J., Darcy, D.P., Daniel, S.L.: Opportunities and challenges applying functional data analysis to the study of open source software evolution. Statistical Science 21 PP:167–178,2006

[65]. Suh, S.D., and Neamtiu, I.: 'Studying Software Evolution for Taming Software Complexity', 21st Australian Software Engineering Conference, 2010

[66]. T. Zhu, Y.Wu, X.Peng, Z.Xing and W.Zhao "Monitoring Software Quality Evolution by Analyzing Deviation Trends of Modularity Views", WCRE, 2011

[67] Thomas, L.G.; Schach, S.R.; Heller, G.Z.; Offutt, J.; , "Impact of release intervals on empirical research into software evolution, with application to the maintainability of Linux," Software, IET , vol.3, no.1, pp.58-66, February 2009

[68] Tortoise SVN available at "http://tortoisesvn.net/ Access date 5 January 2011

[69]. Vasa, "Growth and Change Dynamics in Open Source Software Systems",A thesis presented for the degree of Doctor of Philosophy,pp193-194, 2010

[70] Van. And Bosch, J.: 'Design erosion: problems and causes', The Journal of Systems & Software, 61, (2), pp. 105-119,2002.

[71]. Wahyudin, D.; Schatten, A.; Winkler, D.; Tjoa, A.M.; Biffl, S.; , "Defect Prediction using Combined Product and Project Metrics - A Case Study from the Open Source "Apache" MyFaces Project Family," Software Engineering and Advanced Applications, 34th EuromicroConference , vol., no., pp.207-215, 3-5,2008.

[72]. Weiyi Shang; Zhen Ming Jiang; Adams, B.; Hassan, A.E.; Godfrey, M.W.; Nasser, M.; Flora, P.; , "An Exploratory Study of the Evolution of Communicated

Information about the Execution of Large Software Systems," Reverse Engineering (WCRE), 2011 18th Working Conference on , vol., no., pp.335-344, 17-20 Oct. 2011

[73]. Weiderman, N.H., Bergey, J.K., Smith, D.B., and Tilley, S.R.: 'Approaches to Legacy System Evolution', 1997.

[74]. WinMerage available at "http://winmerge.org/", Access Date 10 November 2011

[75]. Wu, J., Spitzer, C.W., Hassan, A.E., Holt, R.C.: Evolution spectrographs: Visualizing punctuated change in software evolution. In: Proc. IWPSE, Kyoto, Japan, IEEE Computer Society Press PP. 57–66.2004

[76]. Xie; Jianbo Chen; Neamtiu, I.; , "Towards a better understanding of software evolution: An empirical study on open source software,", ICSM 2009. IEEE International Conference on, vol., no., pp.51-60, 20-26 Sept. 2009

[77]. Xiaolong Zheng, Daniel Zeng, Huiqian Li, Feiyue Wang, "Analyzing open-source software systems as complex networks, Physica A", Statistical Mechanics and its Applications, vol 24, 15 O, pp.6190-6200, October 2008.

[78]. Yunwen Ye;, "Peer to peer support for the reuse of Open Source Software libraries," Information Reuse & Integration, 2009. IRI '09. IEEE International Conference on , vol., no., pp.284-289, 10-12 Aug. 2009

[79]. Yu, L., Ramaswamy, S., and Bush, J.: 'Symbiosis and Software Evolvability', IT Professional, pp. 56-62. 2008

[80]. Yu.L S. Ramaswamy "Measuring the evolutionary stability of software systems: case studies of Linux and FreeBSD", IET Software, Vol. 3, Iss.1, pp. 26–36.2009.

[81]. Zaidman, A.; Van Rompaey, B.; Demeyer, S.; van Deursen, A.; , "Mining Software Repositories to Study Co-Evolution of Production & Test Code," Software Testing, Verification, and Validation, 2008 1st International Conference on , vol., no., pp.220-229, 9-11 April 2008