FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN · BOLZANO

European Master Software Engineering

A.A. 2010/2011

Empirical Software Measurements:

Project Report

Group Members: *Nicolò Paternoster* [9426]
Carmine Giardino [9099]

0. Outlines

Abstract

Measuring the number of opened bugs is a common activity in software program analysis. Measuring how they can vary over time has been less evaluated previously. This project aims to examine eleven open source projects in order to find statistical differences in early and late stages in terms of number of opened bugs. Moreover, assumptions on maturity properties will be to take in place in view of stability behaviors in fix-bugs activities.

1. Problem Statement

A software system can be described by the software release life cycle. It is composed by discrete phases, which define the software's maturity over time.

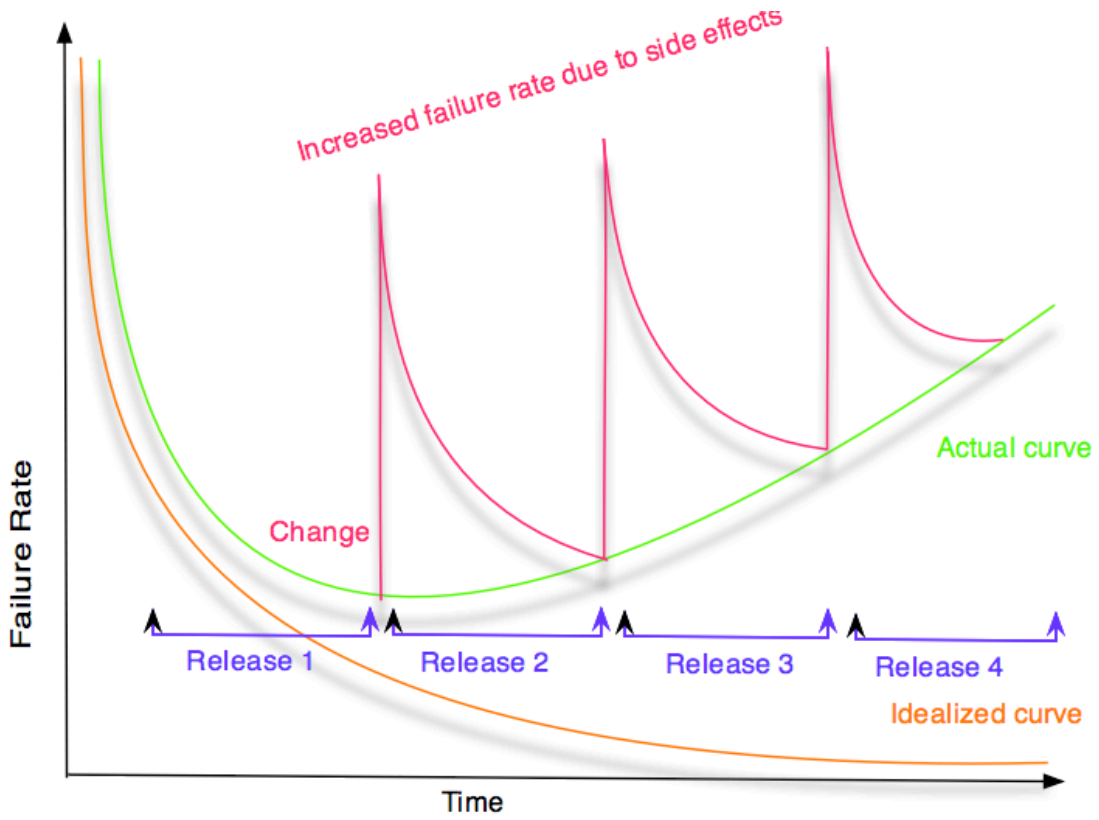

Figure 1. Life Cycle Failure Rate

Updated many times during their life cycle, software systems decrease their reliability introducing new bugs, as described by the actual curve in the graph. On the other hand, analyzing just one release we can focus upon a given time interval of the whole life to understand how the failure rate behaves from an early to a late stage point.

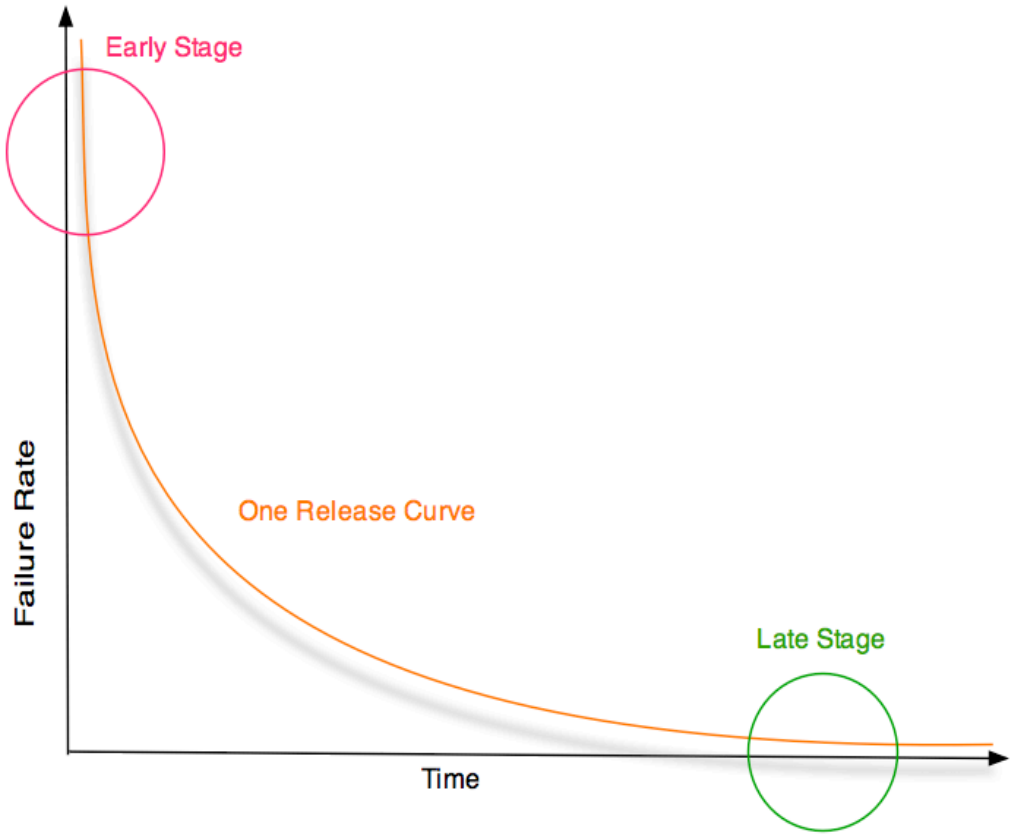

Figure 2 Release Failure Rate

Early stage of a software system is usually defined as a new adoption by the community. It often implies very high failure discovering activity. Late stage can be considered as the period when all the main bugs are fixed and new failures can be found only increasing the time of system testing. Moreover, increasing number of users over time implies increasing number of open bugs, in view of the higher occurrence probabilities of wider operational profile configurations. However, in the study we're going to approach such hypothesis is not assumed. Even if the theory of an idealized curve can be defined as explained, we want to assess the behavior of a system independently from any assumptions. Definitely the tests we're going to mention will be dual-tailed in order to analyze the influence of the stages over the number of opened bugs from a more general viewpoint prospective.

In order to run our research we performed the following steps:

- Establish hypothesis from the research question: state null and alternative hypothesis.
- Determine the appropriate statistical test and sampling distributions
- Specify the error rate
- Gather sample data
- Compute the value of the test statistic
- State the statistical conclusion
- Develop a tool to facilitate data analysis

1.1 Objective of the experiment

Research Question: are there more issues reported in the early stage or in the later stage of Software lifecycle?

1.2 Definition

- **Goals**: investigate if there is a relation between the number of issues opened in the early stage and in the late stage of the project.
- **Purpose:** evaluate the maturity of the software project.
- **Quality Focus:** number of issues opened day-by-day.
- **Perspective:** project manager willing to better understand the evolution of issues reports.

1.3 Context

- **Subjects:** Eleven open source software projects that are running from more than 4 years.
- **Objects:** Issues report on BugZilla.

1.4 Hypothesis Formulation

H0: there is not difference in the number of new issues opened between the early stage and the late stage of the software life cycle.

Ha: there is a difference in the number of new issues opened between the early stage and the late stage of the software life cycle.

1.5 Variable Selection

- **Independent variable:**the main factor of the experiment is the maturity stage of the life cycle (early or late stage)
- **Dependent variable:** the number of issues opened day-by-day

1.6 Co-Factor: Increment

The co-factor we're going to consider in this study is the increment of changes in the number of open bugs over time. The idea concerns to the fact that a project's maturity could be influenced by the fixing activity of bugs other then by the stage phases. The assumption is: a stabilized fixing activity provides very small increments or decrements occurrences. In our particular case, we want to analyze if having a wider range of instability leads to a higher/less number of open bugs and vice-verse. Indeed, we test if the increment variable affects the number of opened bugs and if it has interaction with the main factor (stage).

The subsequent null hypotheses can be formulated on co-factors:
H0: the projects' increment does not significantly interact with the stage period to influence the number of opened bugs.
Ha: the projects' increment interacts with the stage period to influence the number of opened bugs.

2. Experiment Design

In this section is described how the data was collected and which OS projects are taken into account for our analysis.

2.1 Data gathering

We started browsing and exploring all the possible open source bugzilla repositories *[1]* and selected different open source projects hosted on two of them that use the 4.0+ version of Bugzilla.

- Apache Software Foundation *[5]*
- Mandriva *[6]*

This last version of bugzilla (February 2011) has a feature called "*New Charts*" *[3],[4]* that shows charts with the history of issues for each day and allows the export of raw data used to plot the chart in *.csv,* and will replace the *"Old Chart"* feature installed on old systems. Moreover there is the possibility to select the filter "*All Open*" that shows only bugs with an *open* status *[7]* (NEW, ASSIGNED, REOPENED, NEEDINFO and VERIFIED) .
Observe that in order to use this feature login is required.

2.2 Projects Selection

Among the several projects hosted on the two repositories we selected a sample of 11 projects that have enough information and historical data to use in the analysis.
For each project we defined two different time frames of the same length: a first time interval where the project can be consider as new born (early *stage*) and a second interval where the project have a longer lifetime and can be considered more mature (late *stage*).

In the table 1 there is an overview of the selected projects.

	Product Name	Start Date	End Date	Maturity	Time frame length	Distance
1	Mandriva Linux	14-Nov-2007	11-Aug-2008	Early Stage	271d	539d
2	Mandriva Linux	1-Feb-2010	30-Oct-2010	Late stage	271d	
3	Ant	10-Jul-2005	3-Mar-2006	Early Stage	236d	1298d
4	Ant	21-Sep-2009	15-May-2010	Late stage	236d	
5	Apache httpd 1.3	5-Jan-2005	1-May-2005	Early Stage	116d	346d
6	Apache httpd 1.3	12-Apr-2006	6-Aug-2006	Late stage	116d	
7	BugZilla	3-Jul-2006	15-Apr-2007	Early Stage	286d	682d
8	BugZilla	25-Feb-2009	8-Dec-2009	Late stage	286d	
9	Batik	20-Feb-2007	11-Sep-2007	Early Stage	203d	874d
10	Batik	1-Feb-2010	23-Aug-2010	Late stage	203d	
11	FOP	10-Jul-2005	25-Nov-2006	Early Stage	503d	994d
12	FOP	15-Aug-2009	31-Dec-2010	Late stage	503d	
13	Jmeter	1-Jun-2006	11-Jan-2007	Early Stage	224d	1327d
14	Jmeter	30-Aug-2010	11-Apr-2011	Late stage	224d	
15	Lenya	10-Jul-2005	29-Jun-2006	Early Stage	354d	844d
16	Lenya	20-Oct-2008	9-Oct-2009	Late stage	354d	
17	Log4j	4-Aug-2005	3-Jun-2006	Early Stage	303d	1272d
18	Log4j	26-Nov-2009	25-Sep-2010	Late stage	303d	
19	Tomcat 5	20-Dec-2005	10-Feb-2006	Early Stage	52d	409d
20	Tomcat 5	26-Mar-2007	17-May-2007	Late stage	52d	
21	Tomcat 6	1-Mar-2007	1-Feb-2008	Early Stage	337d	759d
22	Tomcat 6	1-Mar-2010	1-Feb-2011	Late stage	337d	

Table 1: Selected projects and relative stage periods.

For each stage we downloaded the .csv file with the raw data of the charts and we parsed it in order to obtain a single csv file for each project with three columns: `Stage, date, Open Bugs`. Where the stage indicates if the data refers to the early (*"e"*) or the late (*"l"*) stage of the project, the date is in the standard BugZilla format (*AAAA-MM-DD*) and Open Bugs is the cumulative number of bug that on that day had an *Open Status.* As a convention, we started counting the number of bugs considering the first day of each stage to have 0 issues. The same operation can be done with any OS project hosted on a Bugzilla 4+ version.

2.3 Data analysis

The first analysis we perform is the Mann-Whitney test, since there are no assumptions about normality distributions and in this case the paired test is inappropriate because the Stage is a period of time arbitrarily chosen. In addition two-tailed test is applied because no assumptions about the difference direction is mentioned. Among the test types "Type I Error" is chosen in order to reject a true null hypothesis. The probability of committing a "Type I Error" is *a* that is the level of significance. We put *a* equal to 0.05. Open bugs have been collected per day. The first analysis we're going to define is through the box-plot chart. It is a convenient way to describe statistical report through five number summaries: the smallest observation, lower

quartile, median, upper quartile and largest observation. In additions outliers are defined as observations numerical distant from the other data.
P-value within Mann-Whitney test:

```
a = Wilcox.test(OpenBugs described by Stage)
if p-value of a >= 0.05
              H0
else
              Ha
```

The second analysis computes the linear regression model and reports intercepts and slopes coefficients. Analysis on slopes can be discussed to determine the maturity level of the project as well. It conducts to the co-factor definition, it being in effect a first derivation description of number of opened bugs in a discrete domain.

The third analysis is the analysis of the increment co-factor. Two-way analysis of variance (Anova) is conducted. We fix the level of confidence at 95%, which means that we reject the null hypothesis when the p-value is lower then 0.05.

The last representation is the interaction plot. It doesn't have the typical meaning of how a co-factor interacts with a main factor, in view of the nature of the co-factor. In our study the increment variable is a numerical factor, which can assume infinite numerical values, indeed no categorization can be defined. However, although the typical interaction between the variables cannot be compute, some other considerations can be examined. Specifically with this interaction-plot we want to represent how varying the increments, the openedBugs-mean changes as well by taking in consideration the stage variable. In this configuration, we can analyze how the mean of open bugs in the early stage is higher or lower than the late stage. On the other hand how an "increment stability" is achieved when the openedBugs mean reaches a given magnitude order. In other words it's possible to take in consideration an increment range for example [-1 to 1], and verify how the late has different variability in the mean magnitude order respect to the early stage. Definitely the benefit of this chart remains since it describes how the number of open bugs in mean corresponds to increments or decrements considering the two different stage status.

3. Software for data report

3.1 Introduction

In order to facilitate the process of data analysis we developed a tool, attached to this project report, which helps the engineer in generating user-friendly reports.
Our software takes as a input a list of *csv* files and with an easy-to-use GUI the user can select which product wants to analyze, and generates a html report with several plots and statistical analysis.
Furthermore data is loaded in such a way that is takes just a few minutes to add a new project to the list.

3.2 Specifications

Our tools uses different technologies and languages:

- Java (1.5 +)
- HTML ,css,js
- R

3.3 System Requirements

The program runs under MacOsX and Unix systems, with Java JRE (1.5+) installed, an active Internet connection, the command-line tool *sips* [8] and a well-configured version of R with some optional packages:

- *stringr*
- *calldata*
- *rid*
- *calibrate*
-

These packages can be easily installed from the R Package Manager menu. A browser is required to see the reports. Firefox is a plus.

3.4 Usage

To run the tool, simply go to the program root folder and type (from the system console) *java -jar it.unibz.esm.jar,* or double-click the jar file.

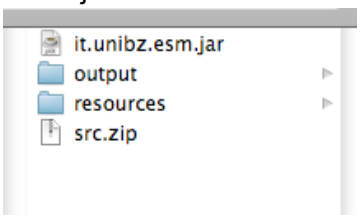

Figure 3 Root folder

Figure 3 shows a screenshot of the program root folder.
Immediately a window will show up listing the available projects (figure 4)

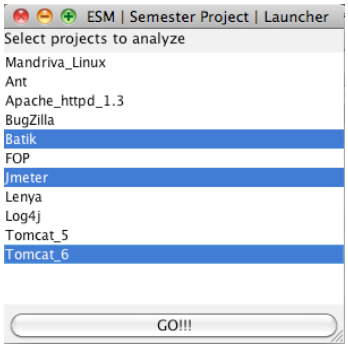

Figure 4 Starting Window

The list of projects are dynamically loaded from a *csv* file (*~/resources/csv/projects.csv*).

```
"ProjectName","EarlyStageStart","EarlyStageEnd","LateStageStart","LateStageEnd","StageLength","PathToCsv"
Mandriva_Linux,2007-11-14,2008-08-11,2010-02-01,2010-10-30,271,1_Mandriva.csv
Ant,2005-07-10,2006-03-03,2009-09-21,2010-05-15,236,2_Ant.csv
…
```

As the user press the *"Go"* button the tool will generate and show an html report containing different plots and statistical data, obtained using R.
For each new report a folder containing all files is created using the timestamp under the ~/output directory as shown in the screenshot below.

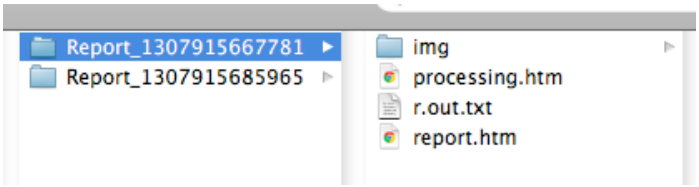

Figure 5 Report Folder

The report.html contains an overview of the analysis, and for each project it shows different blocks of information:

- **Project Overview**

A summary of the main information about the project.

- **Regression line (with a regression table)**

  A plot that shows the number of issues in the early and late stage, fitted with a linear regression line.

- **Box Plot (with quartiles)**

  A box plot containing information of the two stages.

- **Analysis of co-factor (increment)**

  A plot that shows the increment int the two stages.

- **Analysis of variance**

The result of the two-way analysis of variance.

4. Results

In our tool analysis, eleven projects are evaluated to give a more reliable answer to the research question. Here all the results:

	Product Name	P-value
1	Mandriva Linux	2.42E-11
3	Ant	8.90E-52
5	Apache httpd 1.3	7.44E-22
7	BugZilla	1.84E-08
9	Batik	5.83E-45
11	FOP	4.82E-33
13	Jmeter	8.81E-16
15	Lenya	7.54E-90
17	Log4j	5.71E-42
19	Tomcat 5	5.11E-09
21	Tomcat 6	0.0002191935

Table 2: overall results of descriptive statistics and analysis of number of open bugs (Man-Withney test)

As we can see all the projects rejects the null-hypotheses. This is true because all of them have found statistical differences in number of bugs between different stages. In this report for a matter of space we're going to describe just the first of the projects mentioned above, that is Mandriva Linux.

4.1 Main factor analysis

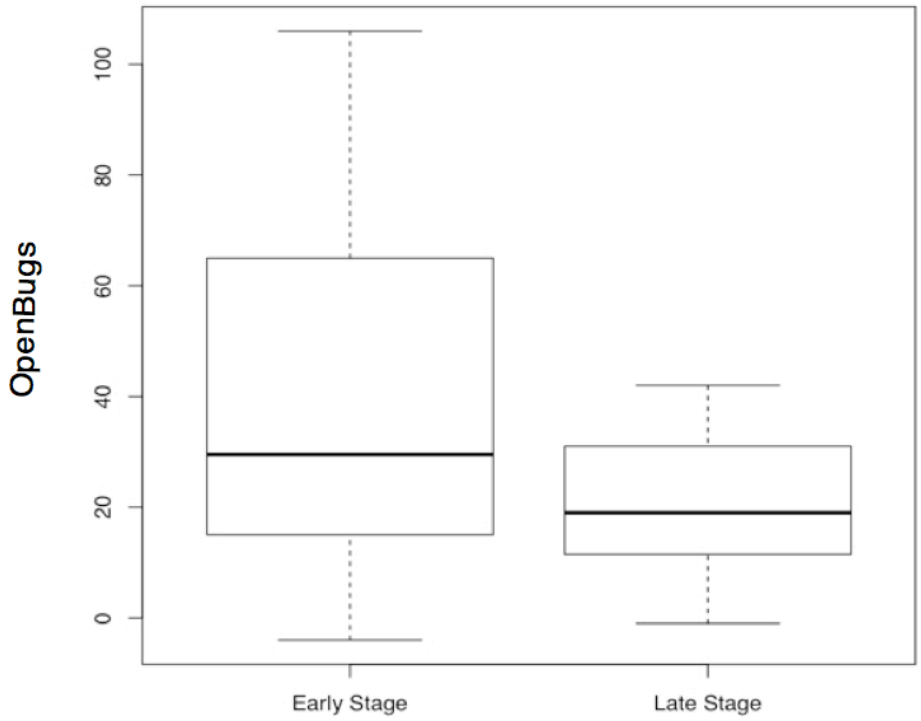

Figure 6 Boxplots

Figure 1 shows the box-plot of the number of open bugs. We can notice that in the early stage the project reports a larger number of open bugs than in the late stage. Specifically, through the quartiles (see Table 3) we can notice how from the early to the late stage all the numerical observations decrease.

Quartiles

```
Early Stage :
 0%  25%  50%  75% 100%
-4.0  15.0  29.5  64.5 106.0

Late Stage :
 0%  25%  50%  75% 100%
-1.00 11.75 19.00 31.00 42.00
```

Table 3: Boxplots Quartiles

The Mann-Whitney test (two-sample Wilcoxon test) reports the p-value equal to 0 (see Table 2), which is significant at the 95% confidence level. Therefore we can reject the null hypothesis H0 and conclude that there is a difference in the number of new issues opened between the early stage and the late stage of the software life cycle.

4.2 Regression Analysis

Knowing that there is a statistical difference between the two stages, we're going to compute regression analysis over the open bugs data-distribution to understand how the slopes change from an early to a late stage period.

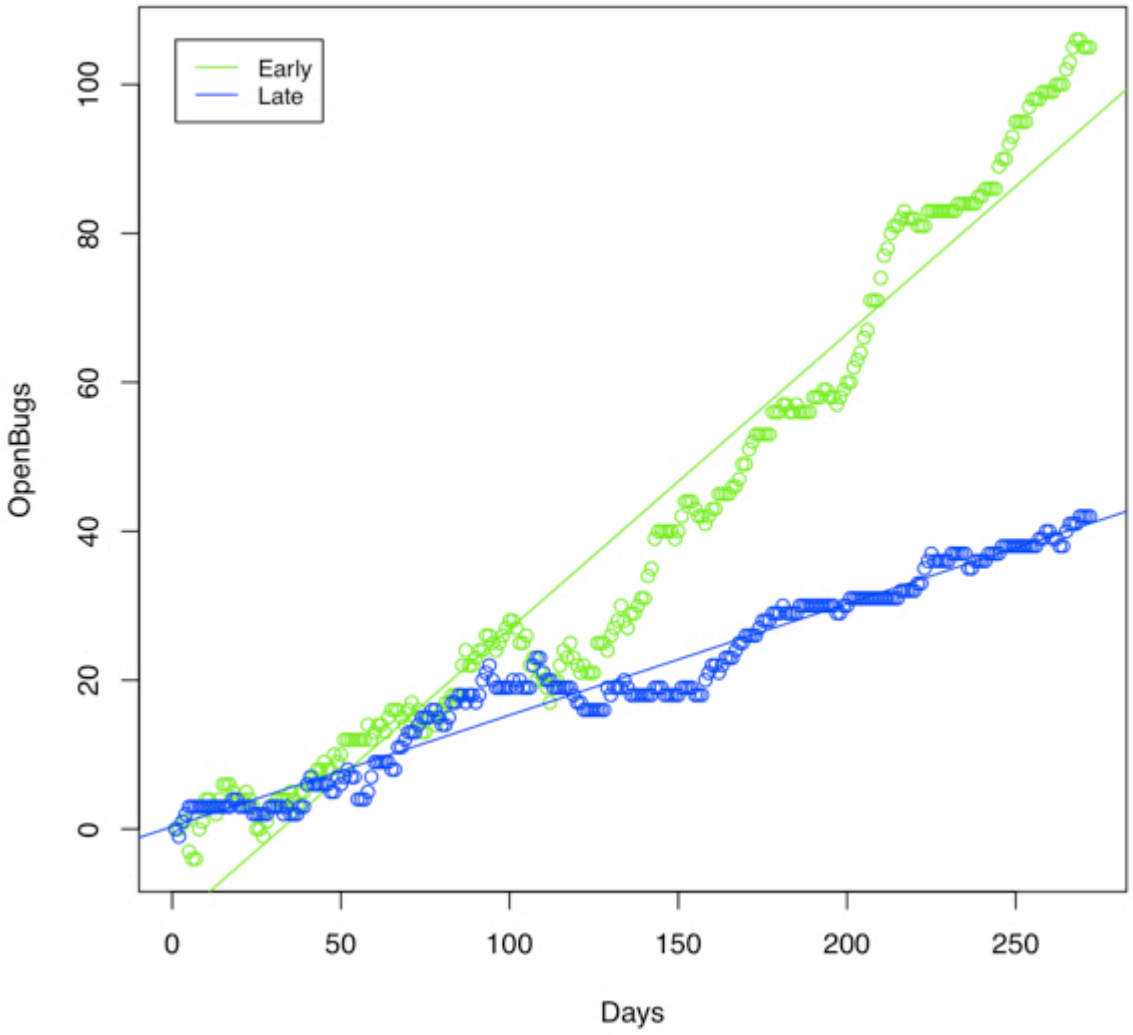

Figure 7 Linear Regression

As we can see, in the early stage the growth of open bugs is much higher then the late stage. Specifically the slope halves from the early stage (0,39) to the late stage (0.14) (see Table 4). Consideration can be taken about the adjusted R-squared. In this case the line well fits the data because the R-squared is equal to 0.95 for both of the regression lines.

Regression table

```
Early Stage :
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -12.772819   0.872010  -14.65   <2e-16 ***
len           0.396364   0.005538   71.58   <2e-16 ***
---
Codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Adjusted R-squared: 0.9498


Late Stage :
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.332537   0.301439   1.103    0.271
len         0.149767   0.001914  78.238   <2e-16 ***
---
Codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Adjusted R-squared: 0.9576
```

Table 4. Regression analysis data

4.3 Co-factor Analysis

Table 5 reports the results of the two-way Anova by Stage & Increment. We can figure out how the increment has a significant effect on the number of opened bugs, although it doesn't interact with the main factor (Stage) statistically speaking. Negative values (decrements) correspond to lower number of OpenBugs on average respect to the positive values (increments), as presented in interpretation discussions later on. This represents a reasonable result if we consider real contexts. Increasing number of open bugs creates problems of management and consequently fix-bugs activities slow down.

Analisys of variance

```
                Df Sum Sq Mean Sq F value  Pr(>F)
Stage            1  57462   57462 99.4742 < 2e-16 ***
Increment        1   3771    3771  6.5281 0.01089 *
Stage:Increment  1    908     908  1.5718 0.21049
Residuals      540 311935     578
---
Codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Table 5. ANOVA of OpenBugs by Stage & Increment

As discussed in the data analysis section, the following graph doesn't have the typical meaning of how the co-factor interacts with the main factor, due to the nature of the increment variable (not categorized). Although this limitation, we can consider some other important aspects. At the first sight, we can notice how the mean of open bugs in the early stage results much higher than the late stage. That is a result that confirms our first alternative hypothesis. But much more important now is how faster "increment stability" is achieved when the openedBugs mean reaches a given magnitude order. To explain this aspect let's take in consideration the increment range -1 to 1, as we can see, the late stage presents low variability in the mean values (from ~20 to ~45) respect to the early stage (from ~15 to ~20).

In other words, when we step from an increment value to another, in a late stage we have more control over the amount of open bugs on average then in an early stage.

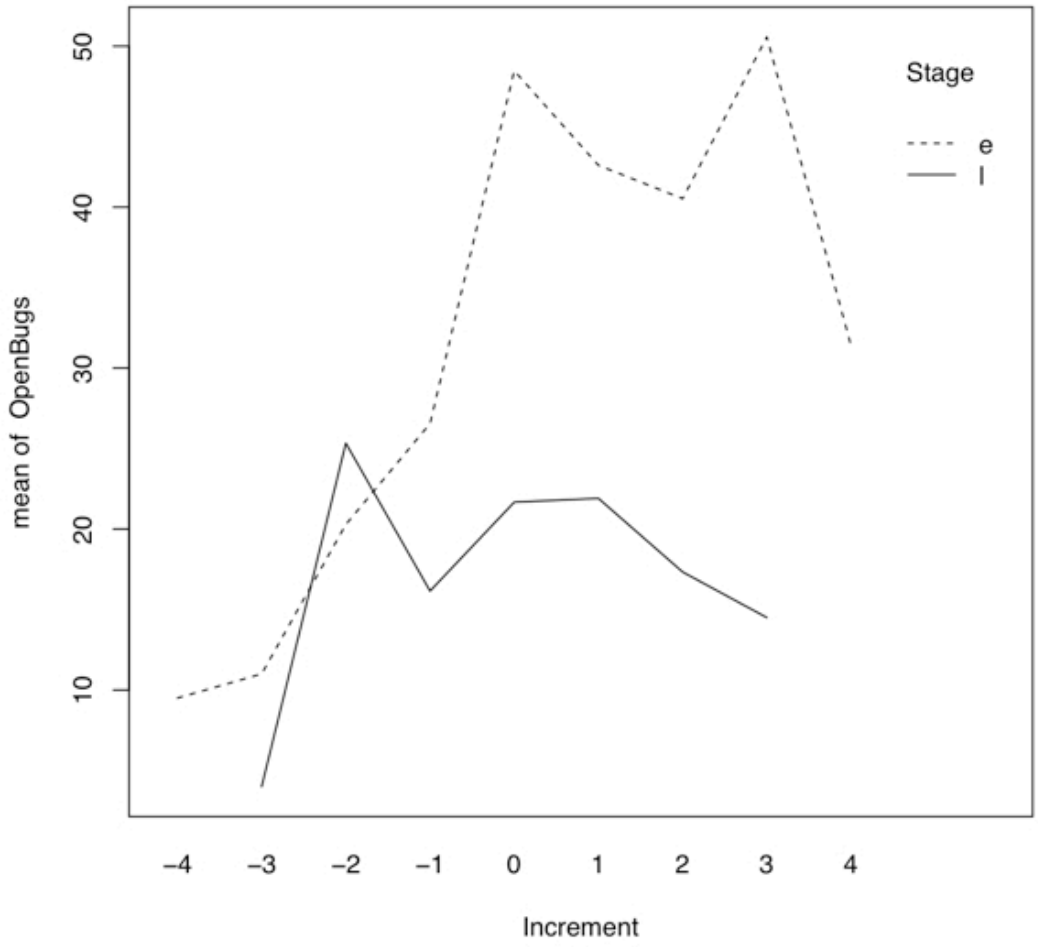

Figure 8 Interaction plot

5. Conclusion

In the reported project, we can conclude that the late stage provide higher number of open bugs then a more mature later stage. Moreover there is an increasing instability of bug fixing activities when a given threshold is overcome. Indeed, as the analysis of variance can show, the “velocity” of how bugs are opened influences the number of bugs itself. In other words the developers can manage stably bugs when they are under a certain order of magnitude.

6. Links

1. http://www.bugzilla.org/installation-list/
2. http://www.r-project.org/
3. https://issues.apache.org/bugzilla/chart.cgi
4. https://qa.mandriva.com/chart.cgi
5. https://issues.apache.org/bugzilla/
6. https://qa.mandriva.com/
7. https://issues.apache.org/bugzilla/docs/en/html/lifecycle.html
8. http://sourceforge.net/projects/sips/

7. Appendix

7.1 Class diagram

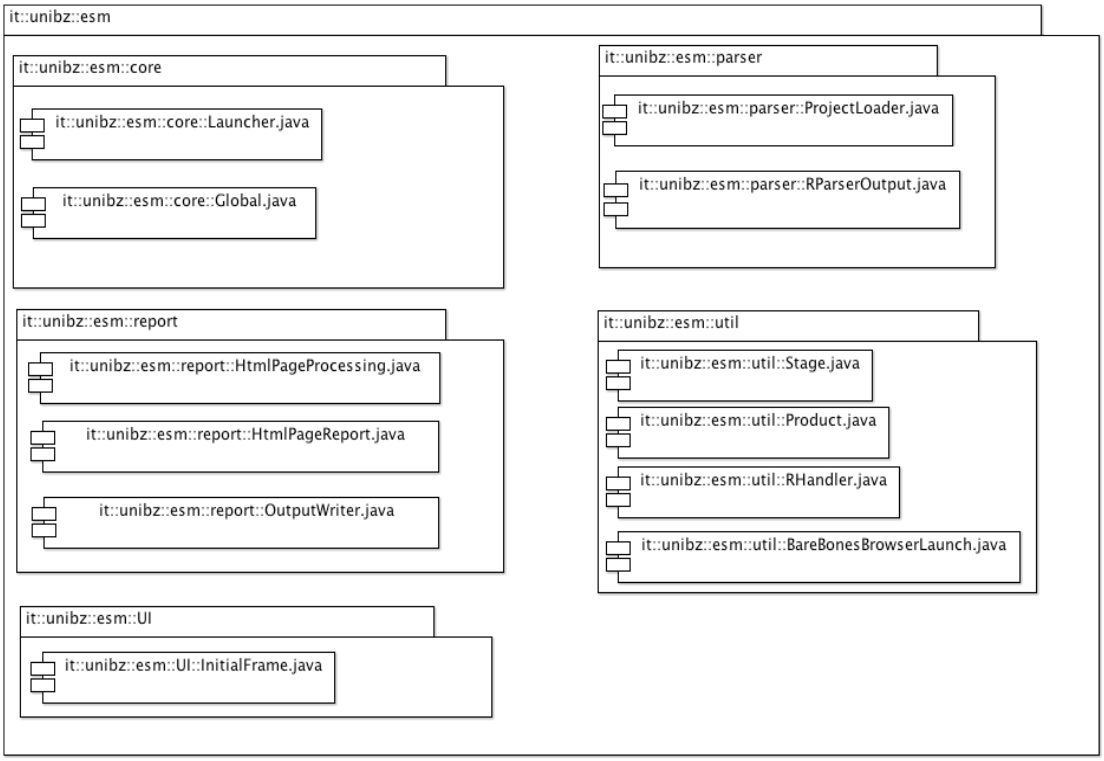

Figure 9 Class Diagram