Software metrics to predict the health of a project ?

Vincent Blondeau, Nicolas Anquetil, Stéphane Ducasse, Sylvain Cresson, Pascal Croisy

Software metrics to predict the health of a project?

An assessment in a major IT company

Vincent Blondeau[1,2] Nicolas Anquetil[1]
Stéphane Ducasse[1]

[1] RMod team, Inria Lille Nord Europe,
University of Lille, CRIStAL, UMR 9189,
59650 Villeneuve d'Ascq, France
{firstname.lastname}@inria.fr

Sylvain Cresson[2] Pascal Croisy[2]

[2] Worldline,
Z.I Rue de la Pointe
59139 Noyelles-lès-Seclin, France
{firstname.lastname}@worldline.com

Abstract

More and more companies would like to mine software data with the goal of assessing the health of their software projects. The hope is that some software metrics could be tracked to predict failure risks or confirm good health. If a factor of success was found, projects failures could be anticipated and early actions could be taken by the organisation to help or to monitor closely the project, allowing one to act in a preventive mode rather than a curative one. We were called by a major IT company to fulfil this goal. We conducted a study to check whether software metrics can be related to project failure. The study was both theoretic with a review of literature on the subject, and practical with mining past projects data and interviews with project managers. We found that metrics used in practice are not reliable to assess project outcome.

Keywords Project health, Closed-source, Data-mining, Interviews

1. Introduction

Typically, IT companies projects are managed by software processes. By following these processes, project teams create a lot of data: process metrics, bug reports, source code, continuous integration artefacts, production metrics, social network communications, etc. With the emergence of big data methodologies, these companies hope data science, and especially statistics, could help them evaluate their project health. Healthy projects will speed up the success of the company, while unhealthy ones can lead to its failure. To achieve this goal, one major IT company asked us to find correlation between data related to its projects and projects health. The hope is that the organization could follow the project's evolution and take preventive actions to prevent project failure. Finding the right metrics in the whole data set is challenging and doing it ahead of time even more.

Studies have already been conducted in this field. But, as they usually consider open-source projects and closed-source projects from other companies, they could possibly not apply to our case because of the different environment reigning in the major IT company.

In this paper, we make three contributions. First, we present the results of a literature review on project success predicted by data mining. Second, we experimented with more than 10 project metrics on 50 real world, closed-source, projects to find links. Third, by doing interviews with project managers, we found indicators that could be linked to project health.

The rest of this paper is organized as follows. In Section 2, we expose the problem. Section 3 describes the methodology followed by our solution and Section 4 presents and discusses the results. Section 5 is dedicated to the related works, and the conclusion will be presented in Section 6.

2. Problem Description

The improvement of the quality of projects is an issue in IT companies. That is why a major firm asked us how they can monitor project health pro-actively. They would like to use past project data, like metrics on software, bugs, budget, production issues, performance, team communications,... and provide alerts for projects by mining them.

In all this data, picking the right set of metrics is a major concern. Many metrics exists around projects and depending on the environment, they could be inadequate.

IWST '15, July 15 - 16, 2015, Brescia, Italy .

ACM 978-1-4503-3857-8/15/07...$15.00.
http://dx.doi.org/10.1145/2811237.2811294

In closed-source software, Nagappan et al. [8] choose to identify the parts of software that are most likely to fail. As metric to measure failure, they used the number of bugs. Bug counts are not uniformly distributed across different software components, so the discrepancies might reveal some development defects and impact the health of the project.

Zimmermann et al. [11] consider the number of post-release bugs to measure the success of the project. The post-release bugs are the number of bugs listed after the release of the application to the client. This metric tests the efficiency of the project team which has to respect all the needs of the client and to deliver a reliable application.

Other IT companies can use different metrics to measure project team efficiency. Project health is related to three items: the budget, the time, and the contents. The three indicators have to be respected. If one is not, the project health will certainly decrease. To avoid this, companies proceed to software testing accordingly to the ISO 9126 standard [6]. There are generally three phases. First, qualification, is a step made by the company where project team test the software to solve the main bugs. Second, acceptance, is the next phase. Test are made by the client inside a development environment. Third, production testing, is made by the end users of the client. This way, project team prevents the likely failure of the project and ensure its quality.

In open source, as the environment is different and as there are no metrics related to budget or time, other considerations are made. Capiluppi et al. [3] and Capiluppi and Fernandez-Ramil [2] think source code quality, including its maintenance, is important for the open-source software. Facilitating maintenance can be considered as improving the health of the project.

3. Solution methodology

In this part, our methodology consisted of a review of the literature on project success predicted by data mining, experiments with more than 10 project metrics on 50 real world projects, and interviews with project managers from the major company.

3.1 Literature review

We hoped previous work could have already found relationships between project success and project metrics. We started from defining the domain of our review: we extracted some keywords and searched them in specialized databases like *Google scholar*. "Cross-project", "Success", "Failure" are some keywords we used. We found papers studying open-source projects and closed-source projects.

Step by step, we followed the references and the citations of each paper to build a network of interesting papers. We found studies about project success, but they were not using project metrics to define success. So we discarded them and kept the essential ones. We will discuss the results in 4.1.

3.2 Data mining

We also conducted a statistical analysis on projects. These projects follow the same development processes. Monthly, project leaders fill Excel files containing information on their project about bugs encountered, budget already spent and the one remaining for the rest of the project.

We used an Excel parser to convert the Excel data into a Moose [1] model. Moose is a Pharo based extensive platform for software and data analysis. This platform, based on meta-modelling principles, offers multiple services ranging from importing and parsing data, to modelling, measuring, querying, mining, and building interactive and visual analysis tools.

The usefulness of using such a model is to have all the data accessible in one place. But the model creation was not easy due to the multiple files to process and the different versions of the Excel sheets. Once the data was in the Moose model, we conduct the statistical analysis with R [9].

R is an open-source platform to apply statistics tools and algorithms on a large set of data. It is widely used by statisticians to perform data analysis. We used a bridge to call R functions directly from Moose.

Excel files are filled by project leaders and no automatic validity check is made, so some of them do not fill in the sheets correctly. For instance, the budget or the number of bugs reporting can be erroneous. The project can still be in a build phase, *e.g.,* not delivered to the client. In this case, the number of post release bug is not set in the sheet, then Excel file of the project will be discarded.

In the Excel sheets, we have 12 project metrics available for each month:

1. Bugs are recorded and categorized in terms of:
   - **Seriousness**: critical, major, minor. A critical bug impedes the usage of a functionality of the application whereas a minor bug can be a misplaced button in the user interface
   - **Testing phases**: qualification, acceptance, or production steps. These metrics correspond to the number of bugs found in each phase. The qualification testing phase is done by the company employees while the other steps are led by the client. The production testing is realized in real conditions with the end-users.
2. **Delta between the number of qualification and acceptance bugs**: for the experiment, we calculate this metric as a difference of both previous metrics. As the qualification and acceptance tests aren't made by the people of the same company, we expect a delta between both values. If the delta is positive, there are more qualification bugs, which is good because it is the project team and not the client who found the bugs. If the delta is negative, more acceptance bugs are found than qualification ones. This will be damaging for the project because of the decrease of the client confidence.
3. Budget

- **Predicted project budget**: it is the budget (in *men-*days*) set at the beginning of the project. For each month an estimate of the budget is done depending on the progress of the project and the number of developers working on it during the month.
- **Realized project budget**: it is the effective budget spent during the month to develop the project. It is known only at the end of the month, whereas the predicted project budget is determined at the beginning of the month and should be spent during the month.

4. Slippage
   - **Delta between predicted and realized budget** (*men-*days* of slippage): it measures the number of *men-*days* that the project deviates in the month. If the value is positive, the project has lost days during this month, else it has "saved" some days.
   - Whether there is slippage or no slippage: there is slippage when the project has at least one month in slippage. This metric is related to the first slippage metric. If the delta between predicted and realized budget is positive, there is slippage, otherwise there is no slippage.
5. **Project name length**: it is the number of characters in the project name. It is intended as a placebo metric. We will compare all results to this metric to see if the name can have an impact on project success.

We also added the number of intermediate releases in a project and the number of months in slippage at the end of the project.

For this company, the project health is related to client delivery. We know that a project is failing if the application is delivered too late to the client. A project is a success if the client has the requested application in time, within budget, with all desired features implemented.

As project leaders consider slippage as the most important metric to assess the health of their projects, we used the three metrics related to it *i.e.,* # months in slippage, # days of slippage, and if there is slippage or not. We decided to compare these metrics to the metrics related to bug number and budget data described above.

3.3 Interviews

To complete the project study, we interviewed project managers of the company to get their feeling on what impacts project health *i.e.,* success or failure. We wanted to know what are their problems during project development, how they detect them and how they resolve them.

The interviews lasted one hour and were decomposed in two parts. In the first, we presented the research topic to the interviewee. In the second, in an open discussion, we try to get all the managers experience on project success and failure.

We also interviewed project managers whose projects we did not analysed. We interviewed four managers, among them, two projects they lead are successful.

4. Results & Discussion

In this section, we describe and discuss the results we obtained from the literature review, the data mining, and the interviews with the project managers.

4.1 Literature review

We surveyed the literature to find studies on correlations between project success or failure, and project metrics. From these findings, we extracted the four relevant papers.

Capiluppi et al. [3] attempt to identify which part of a system should be refactored first. They think files changed the most are prone to be changed again. So a refactoring in advance on these files will increase project health for the future. They studied 62 releases of ARLA, an open-source implementation of the Andrew File System. ARLA is mostly written in C. The authors correlate the number of changes for each source file with four code metrics:

1. Number of functions
2. Cyclomatic complexity [4] sum on functions
3. Cyclomatic complexity average on functions
4. Number of lines of code.

They concluded that 50% of the files that changed most have the highest cyclomatic complexity and carried the highest number of functions.

In another study, Capiluppi and Fernandez-Ramil's [2] goal was to find metrics identifying regressions after refactorings. They used eight open-source software projects in C++ to correlate four code metrics at function level: fan-in (number of incoming calls to the function), fan-out (number of outgoing calls from the function), number of lines of code, and cyclomatic complexity.

They found that no metric alone is a good predictor of regressions. Moreover one has to determine for each individual project which combination of metrics could be used.

Nagappan et al. [8] described how they apply statistical methods to predict bugs after client delivery. They mined five C++ Microsoft applications, including Internet Explorer 6 and DirectX. Nagappan et al. correlated 31 code metrics with post-release project failure data. They used: module metrics (number of classes or functions), functions metrics (number of lines, parameters, called functions, calling functions, and cyclomatic complexity,...), and class metrics (number of classes, superclasses, subclasses, and classes coupled).

By doing statistical analysis, they found, for each project, a set of metrics related to post-release bugs. This set is changing from one project to the other. As project failure has to be anticipated, the ideal is to find an unique set of metrics suitable for any project. This unique set does not exist. So it will be impossible to detect the failure of a project before it happens. It will be not productive for a project manager to know, *at posteriori*, if its project fails or succeed.

They found the same results as the previous study, *i.e.,* no metric can be a good predictor alone. In some cases, the predicting metrics seem more accurate if they have a metric

value in common *e.g.,* the metric representing the language used is the same.

It is possible in cases to create a statistical model from one project and have good results by applying it to another project. But it is rare and there is no way to know in advance if it is going to work.

However, Nagappan et al. use only source code metrics, other studies that correlate project metrics will be complementary.

Zimmermann et al. [11] had for goal to predict class defect from both metrics from source code and project environment. They extracted data from 12 products both closed-source from Microsoft projects, like DirectX, Windows Core File system manager, SQL Server 2005 and Windows kernel, and open-source like Apache Tomcat, Eclipse, Firefox. Several versions of each system were used for a total of 28 datasets. On each version, 40 metrics were gathered.

Concerning source code metrics, Zimmermann et al. used variables ranging from churn (i.e., added, deleted, and changed lines) to cyclomatic complexity. As project environments metrics, they take for example: the domain of the application, the company developing the project, the kind of user interface, the operating system used, the language used, the number of developers. For each metric, they computed median, maximum and standard deviation at project level.

Their empirical study gave results: on 622 cross-predictions between project tested, only 3.5% of the couples can predict each other. For instance, some open-source software projects (Firefox, Tomcat, Eclipse,...) are strong predictors for some closed-source projects but do not predict the other open-source projects. Some other open-source projects cannot be predicted by any of the projects in their study. On the closed-source side, they found some projects such as File System that can predict other closed-source projects. However, they also found some projects such as Internet Explorer, Windows Kernel, and DirectX that do not predict other projects. They also found some systems that are predicting each other.

To summarize this section, it seems possible to find, for a given project, metrics that allow one to makes predictions *a posteriori*. But finding, *a priori*, a unique metric or a unique combination of metrics that can be applied to all projects seems unlikely.

However, development environments between companies and open-source are different. Where one is driven by the money and the time, the other is more independent. We decided to try the experiment with company data and metrics.

4.2 Data mining

Based on the metrics described before (see section 3.2), we analyse their correlation with slippage. We want to know whether any metric from Excel files influences slippage which is our definition of project health.

We used data from 44 projects during the past 3 years. Earlier data was not available. Altogether, we have 1076 Excel files that we can use but only 91 were mined due to data redundancies during the year. Therefore, on these projects, only 19 are useful for our analysis.

By the value of their metrics, several projects can have a great influence on the sample. These outliers have to be identified then removed. Statistical methods advise to take out these kind of extreme values to have a better sample to analyse. We carried a Principal Component Analysis (PCA) to identify these outliers. The PCA algorithm extracts from the initial variables, principal components. They are linearly uncorrelated variables and represents concisely the initial set of variables. The benefit of this analysis is to identify the groups of variables and which one are related to the health metrics.

There are two kinds of representations of the values.

The first representation, figure 1, shows the impact of each individual (project) of the sample on the two first components. The first principal component is put on the abscissa and the second on the ordinate. Each point or cross is the projection of the metrics of a project on the two first principal components. In this way, we can determine the outliers. As shown, the projects represented by crosses are far from the other projects (at the left of the figure). There are the extreme values. We moved them aside to conduct the following studies.

The second representation shows the contribution of each metric of the sample to the two first principal components. We will detail it later.

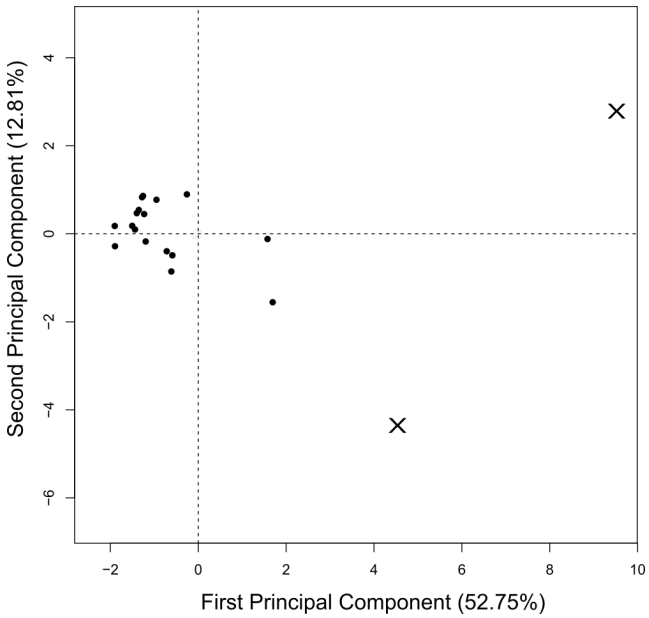

Figure 1. Projection of the project metrics on the first and the second principal components

To have a first intuition, we correlate all metrics described in section 3.2 and whose principal characteristics are detailed in Table 1.

Table 1. Statistical summary of the metrics used for the analysis

Metric	Min	Mean	Median	Max
# Qualification bugs	0	20.3	4.0	81
# Acceptance bugs	0	18.3	1	193
# Production bugs	0	8.1	0	120
# Critical bugs	0	12.4	3	101
# Major bugs	0	17.5	11	92
# Minor bugs	0	16.9	7	81
# Total bugs	0	46	25	274
Δ Qualif. & Accept.	-112	1.9	0	55
Predicted project budget	31	881	411	4700
Realized project budget	63	947	432	5210
# Months in slippage	0	38.9	34	80
# Men*Days of slippage	-60	65.8	32	510
# Intermediate releases	6	18.7	18	31
Project name length	6	18.9	19	32

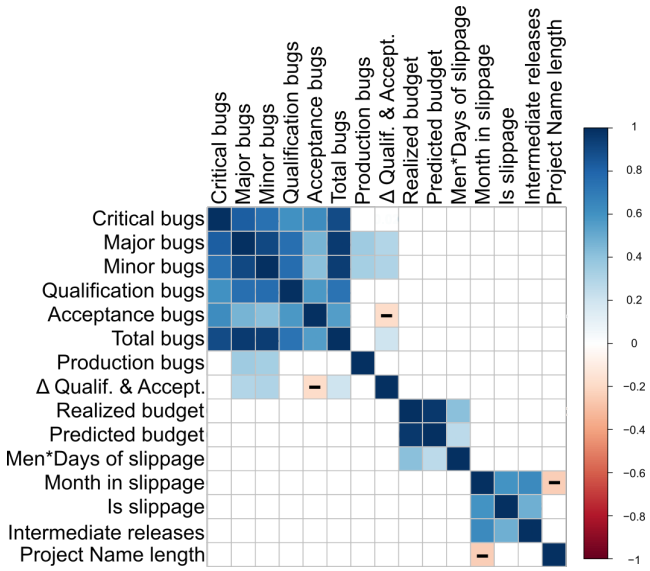

Figure 2. Correlation matrix between each metric of the sample

Figure 2 is a correlation matrix of all metrics we analysed. The crossing of one column and one row shows the correlation value between both variables. It highlights whether there is a linear dependency between 2 variables. If the value is close to 1 or -1, the variables are linked. In this case, it is likely that the evolution of one will change the other. The variables are independent if the values are close to 0. The value is negative if variables are correlated but evolve in opposite directions. As the sample of project is small, we use the Spearman correlation. On the matrix, a lighter coloured value mean that the correlation is closer to 0 (no correlation), a darker coloured one is closer to 1 or -1 (correlation). The correlations values are detailed in Figure 4 of Appendix A.

For instance, in Figure 2, the blue square between the variables representing the total number of bugs and the number of qualification bugs means that both variables are correlated, this is expected. If a minus is in the square, the correlation is negative. For more precisions, the correlations values are detailed in Figure 4 of Appendix A.

This matrix shows three blocks of correlated variables. First, we can infer a strong correlation between all kind of bugs (the darker square at the top left of the matrix) except the number of bugs in production. These bugs are not strongly correlated to the others. It can be due to the fact that it is the final user who found them. The final user didn't take part into the project requirements elicitation step of the project. It is a new eye on the project. Another explanation is that project just began its production, the end-user didn't find any bug yet. So it seems natural that the number of bugs found is not correlated to the other ones. The metric representing the delta between the qualification and acceptance bugs is lightly correlated to the other ones. The correlation is not strong enough to be considered.

Second, we can also see, in the middle block, correlations between the budget variables: the realized budget and the predicted budget. The number of $men * days$ between the initial and final budget seems also correlated to these budget metrics but not to the slippage ones. It might be due to the fact that the bigger a project is, the more difficult it is to predict the budget. A long project is more likely subject to deviations.

Third, it seems also the slippage metrics are significantly correlated together as foretold. Except the $men * days$ of slippage, which is not linked to the others. However, the number of intermediate releases seems correlated to these slippage metrics. It might be the decomposition in group of functionalities that is difficult to determine by the project managers.

Finally, our placebo metric seems lightly correlated to the number of month in slippage. So longer the name, less months of slippage the project will have.

However, as shown by the matrix, there is no link between the 3 groups of variables *i.e.,* the bugs, the budget, and the slippage.

In the light of this analysis, we can conclude that there is not link between the slippage and any other studied variable.

Figure 3 displays the result of the PCA. We see two axes, the first follows the abscissa and the second the ordinate. The abscissa aggregate non-production bugs, the ordinates aggregate slippage and slippage and production bugs.

The budget values have no impact in the 2 components.

Each arrow is representing a variable, the larger the arrow the more this variable is correlated to these two principal components. If the arrow is in the same direction that a principal component, *e.g.,* the first component and the arrow representing the number of critical bugs, both variables are correlated. On the contrary, if the arrow is orthogonal to

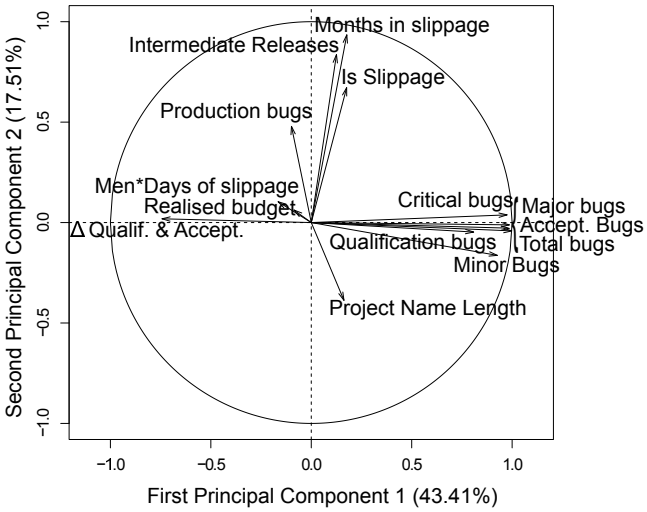

Figure 3. Impact of each metric on the sample

the component, the variable representing the arrow is not correlated to the component, *e.g.,* the bug component and the number of slipping months.

In our analysis, the first principal component synthesize 43.41% of the variability of the sample and the second 17.51%. It means that almost 60% of the sample is resumed by these two principal components.

We can see four groups of metrics from the PCA. First, we have the metrics linked to the bugs at the right of Figure 3. Second, we have the metrics representing the slippage on the top (except the delta between the initial and realized budget). We have also in this group the number of bugs revealed in production.

Third, we have the project name length. It seem negatively correlated to the slipping metrics. So the shorter the project name is, the more slippage it will have and more the project will possibly fail.

Fourth, we can suppose that the other variables (the budget realized, the delta between predicted and realized budget, and the delta between the number of qualification and acceptance bugs) make the group of the non-influential variables.

In the second group, the number of bugs presence means that this metric is increasing, the slippage ones will increase too. It can be explained by the fact that if the project is slipping the final user will find more bugs in the application.

As we have the metrics on the intermediary releases, we conducted the same experiment. In the Excel files, only 59 can be used for analysis among the 720. Actually, it is more difficult to get all the metrics for the intermediary releases because the good filling of the Excel sheets is more complex. After removing outliers, we found the same results than with the projects.

To summarize, the correlations we found are quite trivial. Like the papers from the literature survey, we were are not able to link the bugs and the budget metrics we used with the slippage metric, we considered as an indicator of project health.

4.3 Interviews

As the data analysis and literature review did not show any significant link between project health and project metrics, we conducted 4 interviews with project managers from the company.

In these interviews, they identified the following root cause of project failure:

- Delay at the beginning of the project: if the client decides to begin the project later, the project team is ready to work but not paid by the client. So the company spend money and the relationship with the client will deteriorate.
- Collaboration between the team and the client: if the team and the client know each other well, the collaboration will work fine and the project is more unlikely to fail.
- Team cohesion: if the members of the project team support each other, the cohesion is stronger and the project has significantly more chance to succeed.
- Understanding the specifications: if the project team understand that the client says and succeeds to transcribe it in is own technical language, the project will progress easier.
- Knowledge of the functional concepts: if the project team knows the concepts of the client, the project has more chances of success.
- Change of the framework during the development: if the technical tools or the framework, that the project team uses, change, it will cost more to the project.
- Experience with the used frameworks: a team with experience on tools or frameworks will be quite capable of doing the project faster.
- Bypass the qualification tests: if the team does not test its application before delivery to the client, the client will be unhappy because some functionalities will not work and some tension in the project team will appear.
- High number of bugs listed by the client: as a consequence of the previous item, the client will find more bugs in the application.

Among these root causes, we found by the mining that production bugs are linked to failure. But for the other root causes of failure, mining the project artefacts will not give the necessary metrics.

5. Related Works

This section details literature review of others studies that already tried to predict project success or failure. They are around social coding, *i.e.,* how the project members interact inside their team.

Wolf et al. [10] study the link between team communications and the result of the integration process after merging of the developed software parts. It is based on the

IBM dataset, Jazz. They studied 1 288 build results, 13 020 change sets and 71 000 comments on 47 different team in 4 months. They used 8 metrics representing the exchanged informations.

The authors found no unique measure of social network that predicts if the integration process is a success or failure.

Hu and Wong [5] examine the influence of social metrics on defect prediction. On six releases of NetBeans and seven of Eclipse, they studied the relations between developers thanks to nine metrics on commits. They studied the impact of these metrics on after release defects.

The authors found that the developers relationships metrics are not correlated to the number of after release defects.

It seems that the mining of social coding metrics should not bring an explanation to project failure or success.

Menzies and Zimmermann [7] reference the progress of the predictive analysis applied on software projects. They precise that it is possible to make studies from various data. However, it is impossible to reach conclusions from a project and apply them to all. As they said: "But more recently, the goal of analytics has changed — the research community has accepted that lessons learned from one project cant always be applied verbatim to another." The research heads towards local methods, *i.e.,* be applied to only one project.

6. Conclusion

An major company asked us to found metrics that predict project success or failure. We conducted a study to check whether software metrics can be related to project failure. The theoretical study of literature shows that the metrics extracted from a project cannot be used on another one. The mining of data we have done on company project highlights there is no link between project metrics and data. Moreover, the interviews we conducted shows that the metrics linked to success cannot be found by mining project data.

As all these studies intervene *a posteriori* on projects, it seems random for a new project to know which metric or set of metrics uses to assess success. Predictive analysis will not work well if it is not possible to know *a priori* which statistical model use. In this case, there is no utility at all to mine them to predict the health of a project.

Predicting success or failure requires some sort of understanding of whether customer requests are correctly addressed by the project team. The developers and the project manager are creating their own benchmark that will determine whether they succeeded, and no metric seem required.

To go in depth, we plan to do a survey to better understand the real indicators that are taken into account to assess success or failure of a project.

A. Appendix

	Critical bugs	Major bugs	Minor bugs	Qualification bugs	Acceptance bugs	Total bugs	Production bugs	Δ Qualif. & Accept.	Realized budget	Predicted budget	Men*Days of slippage	Month in slippage	Is slippage	Intermediate releases	Project Name length
Critical bugs	1	0.82	0.74	0.6	0.62	0.89									
Major bugs	0.82	1	0.9	0.75	0.47	0.96	0.36	0.29							
Minor bugs	0.74	0.9	1	0.76	0.42	0.94	0.34	0.3							
Qualification bugs	0.6	0.75	0.76	1	0.58	0.74									
Acceptance bugs	0.62	0.47	0.42	0.58	1	0.55		-0.19							
Total bugs	0.89	0.96	0.94	0.74	0.55	1		0.21							
Production bugs		0.36	0.34				1								
Δ Qualif. & Accept.		0.29	0.3		-0.19	0.21		1							
Realized budget									1	0.97	0.41				
Predicted budget									0.97	1	0.26				
Men*Days of slippage									0.41	0.26	1				
Month in slippage												1	0.59	0.63	-0.25
Is slippage												0.59	1	0.48	
Intermediate releases												0.63	0.48	1	
Project Name length												-0.25			1

Figure 4. Correlation matrix between each metric of the sample

References

[1] Muhammad U. Bhatti, Nicolas Anquetil, and Stéphane Ducasse. An environment for dedicated software analysis tools. *ERCIM News*, 88:12–13, January 2012. URL http://ercim-news.ercim.eu/images/stories/EN88/EN88-web.pdf.

[2] A. Capiluppi and J. Fernandez-Ramil. A model to predict anti-regressive efforts in open source software. In *23rd IEEE International Conference on Software Maintenance*, oct 2007. URL http://oro.open.ac.uk/8243/.

[3] Andrea Capiluppi, Alvaro Faria, and J. F. Ramil. Exploring the relationship between cumulative change and complexity in an open source system evolution. In *Proceedings of the 9th European Conference on Software Maintenance and Reengineering*, 2005. URL http://oro.open.ac.uk/12330/.

[4] Maurice H. Halstead. *Elements of Software Science*. Elsevier North-Holland, 1977.

[5] Wei Hu and Kenny Wong. Using citation influence to predict software defects. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 419–428, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL http://dl.acm.org/citation.cfm?id=2487085.2487162.

[6] ISO. International Standard – ISO/IEC 9126-1:2001 – Software engineering – Product quality. Technical report, ISO, 2001.

[7] T. Menzies and T. Zimmermann. Software analytics: So what? *Software, IEEE*, 30(4):31–37, July 2013. ISSN 0740-7459. .

[8] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th International Conference on Software Engineer-*

ing, ICSE '06, pages 452–461, New York, NY, USA, 2006. ACM. ISBN 1-59593-375-1. . URL http://doi.acm.org/10.1145/1134285.1134349.

[9] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/.

[10] Timo Wolf, Adrian Schroter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 1–11, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3453-4. . URL http://dx.doi.org/10.1109/ICSE.2009.5070503.

[11] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100. ACM, 2009.