Software Metrics in Students' Software Development Projects

Pekka Mäkiaho, Timo Poranen, Ari Seppi

Abstract: *A software metric is the measurement of a particular characteristic of a software or the measurement of a software project and process. In this paper we study how student project managers and team members observe metrics, like working hours, number of test cases, requirement statuses, regular reporting, and number of code commits, and which metrics they consider most important.* The metrics that the teams reacted most often were team's working hours and requirements statuses. These metrics were also considered the most useful by the project managers. *We propose a method to calculate defect rate of reporting metrics and combine this defect rate with the way the project teams used the metrics. It seems that when the teams were motivated to use metrics, observed them and reacted on base of the metrics, they also had less defects on reporting the metrics.*

Key words: *Software metrics, Project metrics, Student projects, Unit testing, Continuous integration, Project management*

INTRODUCTION

A software project can be called succeeded if it is completed on-time and on-budget and all the requirements are fulfilled as specified [1,2,8,13]. The surveys made by Standish Group [3] show that the projects very rarely fulfil these success criteria. Standish Group is an organization that publishes a survey every two year in which it follows the success of software projects mainly in US and European companies. The results show that during the year 2012 only 39% of the projects succeeded. The project was assessed to be failed, if it was terminated or if the software was never deployed. The failure rate of the studied projects was 18%.

Project management consists of five phases: initiating, planning, executing, monitoring plus controlling and closing [7]. One of the most important reason that a project fails is poor reporting of the project's status [10]. Because of the poor reporting, the management does not know the state of the project and thus does not execute the right actions. So it is failed on the monitoring and controlling phases.

In the next section software metrics are introduced with more details. Then data gathering method is explained and after that the obtained data is analysed. The last section concludes the work.

CompSysTech'15, June 25-26, 2015, Dublin, Ireland
Copyright © 2015 ACM. ISBN 978-1-4503-3357-3/15/06...$15.00
http://dx.doi.org/10.1145/2812428.2812473

SOFTWARE METRICS

A software metric is the measurement of a particular characteristic of a software or the measurement of a software project and process. Goodman [5] defines software metrics to be "the continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products".

Software metrics can be categorized into *controlling* and *predicting* metrics: controlling metrics are related to software processes and predicting metrics to software products [9]. An example of a controlling metric is the average time used for bug fixing, and an example of predicting metric is the total number of lines of code. The controlling metrics can also be divided into *process and project metrics* depending on whether they are measuring the software process, and thus supporting strategic decisions, or whether they are measuring an individual project and supporting tactical decisions [4]. In this paper we use terms *product metric, project metric and process metric.*

DATA GATHERING

The data was collected from the courses Software Project Management (SPM) and Project Work (PW) from the School of Information Sciences, University of Tampere during academic year 2013-2014 [6]. The projects started in September and ended in March.

The PW course was a BSc level course and 46 students participated in that. On the MSc level course, SPM, there were 41 students. In the courses, students formed project groups of 6-8 members, so that the project managers came from the SPM course and the developers from the PW course. Main learning goal of the courses is to give students an experience on a managed and structured software development project with a real client. Main outcome from the project is runnable software that fulfils client's reasonable requirements.

It should be emphasised that teams were relatively inexperienced. The managers had experience at least from one student project where they acted as developers, and most of the developers were participating in their first project.

The project groups had a mandatory set of deliverables with given deadlines. The deliverables included software, documentation, weekly reports and six formal review meetings with the course's supervisor.

Weekly reports were sent to all stakeholders including course supervisors. Supervisors updated a statistics page [12] in which the data was published. The next statistics were shown on monthly bases: used hours, date of deliverables and reviews, requirements with statuses, commits to the version control system, number of (unit) test cases and the number of passed (unit) test cases.

The projects were given instructions to report the statuses of the requirements by using six values: New (a requirement has been proposed), In progress (the group has committed to implement the requirement), Resolved (implemented), Feedback (under testing), Rejected (deleted) and Closed (requirement is implemented and it has been tested that the implementation is correct). Some teams used Redmine or Jira project management tools to maintain requirements statuses.

On top of these, the students filled three Moodle questionnaires with questions on previous unit testing experience, usage of metrics in their projects and the challenges faced on unit testing. The first questionnaire was filled in November when the implementation was started, the second in the middle of the projects (January) and the last in April when the projects should have been finished.

The course offered students access to an Ubuntu Linux continuous integration server where Jenkins [14] and SonarQube [15] services had been set up. Jenkins is a widely used Java based continuous integration server where build and test scripts (jobs) can be launched automatically when changes occur in the version control system. The result of the job is usually a build result and/or report of the test runs and metrics.

SonarQube is a platform for continuous code quality observing. It ties together several tools allowing code quality analysis for multiple languages.

The workflow for creating the quality observation setup for a project began after the group had something that could be built or analyzed. The members of the group could then contact the server administrator who created accounts for Jenkins and SonarQube. Using those accounts the groups could then configure the Jenkins job to analyze their project with SonarQube.

SOFTWARE AND PROJECT METRICS IN STUDENT PROJECTS

An overview of projects is shown in Table 1. More detailed data can be found from course's statistics page [12]. In Table 1, it is given project number, duration in weeks, final hours, final requirements statuses, number of code commits, number of the passed unit test cases from all the unit test cases and number of the passed test cases from all the test cases. The values on the Final requirements –column tells how many requirements there were on each state (New, In progress, Resolved, Feedback, Closed, Rejected).

Table 1. Overview of projects.

	Duration	Hours	Final requirements	Commits	Passed unit test cases	Passed test cases
1	26 weeks	820	3/0/0/2/14/1	104	1/1	10/12
2	26 weeks	1190	0/0/0/0/54/2	447	16/16	32/32
3	28 weeks	1247	0/4/12/0/2/8	378	2/2	5/5
4	26 weeks	1250	26/6/0/8/32/1	189	12/12	35/37
5	28 weeks	918	2/0/1/4/23/0	120	7/7	56/65
6	27 weeks	906	21/18/18/0/18/3	153	0/0	?/?
7	25 weeks	953	0/0/0/0/51/36	89	14/14	16/16
8	26 weeks	1167	0/0/8/3/1/2	76	0/0	11/11
9	25 weeks	1000	4/13/2/0/0/0	153	0/0	0/10
10	28 weeks	932	0/0/0/29/3/15	195	0/0	9/37
11	30 weeks	1245	33/6/6/0/0/0	59	0/0	0/0
12	25 weeks	810	0/1/0/4/8/4	25	0/0	0/0
13	27 weeks	1049	0/0/17/0/0/6	139	0/0	0/0

FINDINGS FROM METRICS

The weekly reports were almost always sent on time, before Monday midnight. During the course only one project was late with or skipped more than one weekly reports. Three projects were late with a single report. The rest 9 projects sent the reports always on time.

Looking at the monthly working hours [12], we have left out the September as the projects started after the middle of September. Also the statistics from December and January are not comparable as some of the project teams had a holiday between the semesters. Leaving these (and one non-reported month from a project) out, the average number of working hours was 163.9 hours per month. Maximum number of hours was 445 done by project 8 on February and the minimum monthly hours were 101 by project 6 at November.

To see how the projects kept the schedule, we chose some of the deliverables from the reporting: 5 reviews and the final test report. The preliminary analysis was to keep on the 27th of September, 2013 with course supervisor and the client (optional); the project plan review's deadline was the 11th of October; three separate reviews with the client and the supervisor were set to November, December and January. The test report was to return by the 7th of March. It was not always easy to find a suitable time for the meetings with the project group, client and course supervisor (having 12 other groups) with the limited number of the meeting rooms. Thus we decided that if the meeting was held 10 days or less after the deadline it was considered to be held on time. On the average, the projects were late with 3.7 out of the 6 chosen deliverables. It is not surprising that with the 1st delivery (Preliminary analysis) there were least delays (2/13) and with the last delivery (Test report) there were most (10/13).

On average, the projects had 39 requirements during their life cycle (from 14 to 47). One could assume that in the end of the project the status of a requirement is either closed (average number of closed requirements was 16), rejected (average 6) or new (average 7) if the implementation never started. However, on the final state of the projects, there were other statuses reported too: in progress (average 4), resolved (average 5) and feedback (average 4). In the students' projects it may have been a known decision to publish some requirements without testing and thus the statuses were left to be resolved. However, after the project has been delivered, the status should never be *in progress* or *feedback* (testing). When either of these two statuses existed, it was considered an error in reporting.

The number of all test cases in a project varied between 0 and 65 and the number of the unit test cases between 0 and 16. Only 7 groups out of the 13 reported reasonable values every month. The number of reported commits to the version control system in the final product varied between 76 and 447. Two groups skipped the reporting of commits during 2 months or more.

Only two groups made unit tests to be run on Jenkins. This is explained by the project groups often having little or no experience in unit testing or metrics analysis. This was emphasized in groups who worked on languages other than PHP, for which the course material provided ready examples. But even the PHP groups had problem if they used a framework that required some additional setup to work with unit tests, for example.

Of the two groups who had unit tests run on Jenkins, the first got test coverage of 20.5%, the other only 2.5%. But low percentage of the latter is partially explained by their project being a continuation from earlier project that did not have unit tests at all.

In project 10 neither the developers nor the project managers did have any previous experience about unit testing. They however were one of few groups that actually used metrics, one of the project managers observed SonarQube reports and took action when needed. The project group did still have the common problem that when nobody is very experienced the basic functionality requires so much work that the quality related issues are often ignored just to get something done.

In the current setting only three of the 13 groups had more than 20 SonarQube runs which can be considered as some kind of minimum for any kind of metrics usage.

Not that surprising result was that the groups (projects 2, 4, 7 and 10) that had specific plans about metrics usage, also had the most Sonar runs and unit tests.

One of the problems for the metrics generation were diversive project platforms. PHP applications were supported the best, some groups tried Android analysis, but encountered problems. C# analysing was not available because it would have needed Windows server to be properly configured [11].

FINDINGS FROM MOODLE QUESTIONNAIRES

In the Moodle questionnaires, it was asked if the members saw the metrics important, which metrics were observed, and which were considered as the most important ones. On top of these, project managers were asked, if they used the metrics for making decisions, i.e. whether they reacted on the basis of metrics and if they had a formal process to observe metrics.

The questionnaires provided interesting viewpoint to what metrics the project groups really observed and what metrics the project members saw the most important. These are shown in Table 2.

Table 2. Metrics observed and seen most important.

Metric	**Observed by number of groups**	**Reported to be most useful**
Working hours	12	7
Requirements/User stories	11	8
Commits/Code revisions	6	2
Code warnings (from Sonar)	6	1
Test cases	3	1
Lines of code	3	1
Number of bugs	2	0
Code coverage	2	0
Function points	1	1
Rules compliance index	1	1

Almost all groups observed working hours (12 groups), after that the most observed metrics were requirements (11) and the number of commits (6). Product metrics were not so well observed; violations were observed by 6 groups, unit test count were observed by 3 groups.

For the managers the working hours were clearly the most used metrics as they were the most important for passing the course and easiest to react (informing the course personnel with lacking hours that they need to do more hours). The second most reacted metrics were the requirements as the end deliverable also strongly affected the grade. As for the product metrics some groups reacted on them, but mostly they were not used. One comment was that the group reacted to the issues that caused changes in product metrics not to the changes in metrics themselves.

The unit testing material provided unit testing examples with PHP only so groups using other languages had difficulties in implementing the tests as most of the course participants had little coding experience. Even for the groups using PHP creating unit test cases for different frameworks not used in the examples posed a considerable challenge.

On Jenkins server some groups faced the obstacle that their project language was not supported, this was the case with C# projects. Even if the language itself was supported the frameworks created additional difficulty as not all the frameworks were supported on the Jenkins server when the course began and getting them to work required some debugging from the group.

DEFECT RATE AND METRICS USAGE

In Table 3, we have compared how the project teams themselves used metrics and how well they reported. *Defects on reporting* here is a sum of different factors multiplied by a number which tells how important we saw this factor on the course. The first factor of defect was if the weekly reports were sent on time. If all reports were on time, the value of *Delivery of reports* is 0; if 1-2 reports were late, the value is 1; value 2 is given if 3 or more reports were late or missing; multiplier is 1.

Table 3. Groups, reporting and own metrics usage.

Project# /Multiplier	1	4	3	1		0.1	1	1	2	
Factor (values)	Delivery of reports (0, 1)	Requirements (0,1,2,3)	Test cases (0, 1, 2)	Commits	**Defects on reporting**	Number of metrics observed	Metrics considered important (-1, 0, 1)	Reaction to metrics(0,1)	Formal process(0,1,2)	**Metrics usage**
1	0	1	0	0	4	2	1	0	0	1.2
2	0	1	0	0	4	5	0	1	1	3.5
3	0	2	0	0	8	5	0	1	1	3.5
4	0	3	0	0	12	5	1	0	1	3.5
5	0	1	0	0	4	3	-1	0	1	1.3
6	0	3	2	0	18	2	0	0	0	0.2
7	1	1	0	0	5	7	0	1	2	5.7
8	0	2	2	0	14	5	0	1	2	5.5
9	0	3	1	0	15	4	1	1	2	6.4
10	0	3	1	0	15	9	-1	1	1	2.9
11	2	3	2	1	21	4	-1	0	0	-0.6
12	1	3	0	1	14	2	0	1	0	1.2
13	0	2	1	0	11	4	1	1	1	4.4

Requirements has value 0 if the requirements were reported monthly and the total amount of the requirements was consistent (not decreasing) and at the end of a project there were no requirements in wrong states (feedback or progress). Value 1 is given if there were no inconsistencies in the total number and 1-10% of the requirements were in wrong state or if there were inconsistencies during one month but at the end of the projects all statuses were correct. Value 2 is given if there were inconsistencies on the total amount and 1- 10% of the states were wrong. Value 3 is given in all the other cases. The multiplier of the requirements-factor is 4.

Test cases gets value 0 if unit test cases and the other test cases were reported monthly and the values were reasonable; 1 is given, if the values were missing less than from 2 months or the values were not reasonable; number 2 is given in all the other cases. Multiplier for test cases is 3.

Commits-factor gets value zero if all the values were reasonable and the numbers are missing maximum from one month; value 1 is given in all the other cases. The multiplier for commits-factor is 1.

Metrics usage is also a sum of different factors with given weights. The weights were given on the base of our own knowledge and experience on project management. *Number of metrics observed* is the number of the metrics the group observed. It is given multiplier 0.1. *Metrics considered important* gest a value -1 if the project members answered that they did not see metrics important, 1 is given if the metrics was seen important and the value 0 if the answers were contradictory or missing. Multiplier for the metrics considered important is 1.

Reaction to metrics gets a value 1 if the project reported that they had actions on the project on the basis of the metrics; otherwise, the value is zero. Multiplier of the reaction to metrics is 1.

Formal process -factor gets a value 1, if the metrics were observed systematically but no formal process was planned; value 2 is given, if there were a formal process and it was followed; otherwise the value is zero. Multiplier for the formal process -factor is 1.

In Figure 1, each project is put on the diagram so that the value of defects sets the place on the y-axis and the value of the x-axis is got from the metrics usage -value.

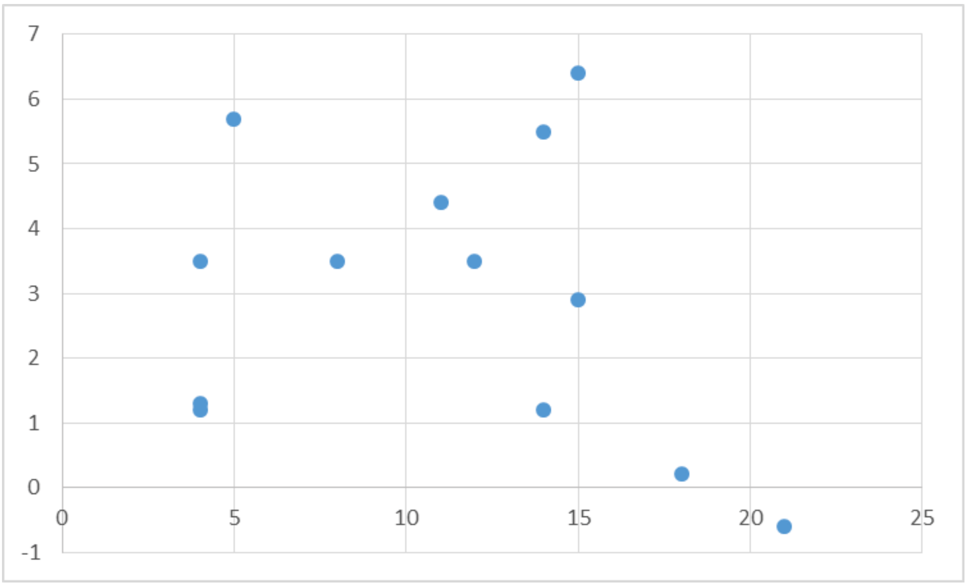

Figure 1. Defects on reporting (x-axis) vs Own metrics usage (y-axis).

There were 4 projects which had the defect-value 5 or less; one of those had also the second highest own metrics usage. Actually this project was selected the best project of the year. There were two projects which had very little own metrics usage (value less than one). These projects had also the highest values on defects, and they had problems to deliver the project.

It would be interesting to research further, if there is a dependency between the reporting of metrics and the metrics own usage in the project. All the projects seemed not to follow this pattern: there were two projects which reported conscientiously even if they did not use metrics themselves. These 'rule follower' projects got grades from 4 to 5. Two projects also had a high own usage and more than average value on defects (projects 8 and 9). The grade of these projects were 4.

CONCLUSIONS AND FUTURE WORK

As far as Jenkins server and metrics generation are concerned, the students had lots of difficulties setting up their projects so in the future some kind of automatic metrics generation setup with unit test templates could be considered.

It seems that there is a relation between how much the projects used the metrics themselves and how well they reported the metrics. As the poor reporting is one of the reasons why the projects fail [1], it could be worth of researching further whether the reporting level could be increased by teaching the project groups how they could better utilise the metrics generating tools and also take advantage of observing metrics.

REFERENCES

[1] Attarzadeh, I. and Ow, S.H. Project Management Practices: The Criteria for Success or Failure. Communications of the IBIMA, 1. Pages 234-241. 2008.

[2] Blaney, J. Managing software development projects, PMI Seminar/Symposium (Oct 7–11, 1989). Pages 410–417.

[3] The Standish group, CHAOS Manifesto 2013. The Standish Group International, 2013.

[4] Fenton, N. Software Metrics - A Rigorous Approach. Chapmann & Hall, London, 1991.

[5] Goodman, P. Practical Implementation of Software Metrics, McGraw Hill, London, 1993.

[6] Mäkiaho, P. and Poranen, T. (editors) Software Projects 2013-2014. University of Tampere, School of Information Sciences, Report 27 (2014).

[7] A Guide to the Project Management Body of Knowledge, 5th edition. Project Management Institute, 2013.

[8] Rook, P. Controlling software development projects. Software Engineering Journal - Controlling software projects archive. 1 (1), Jan. 1986. Pages 7-16.

[9] Sommerville, I. Software Engineering, 9th edition. Addison-Wesley, 2010.

[10] Why Software Fails. http://spectrum.ieee.org/computing/software/why-software-fails. Referred 31.3.2015

[11] SonarQube, C#-plugin. http://docs.sonarqube.org/display/SONAR/C%23+Plugin. Referred 31.3.2015.

[12] SIS-Projectwiki, statistics page 2013-2014. http://www.sis.uta.fi/~tp54752/project-statistics.

[13] Weitz, L. How to implement projects successfully, Software Magazine, 9 (13), 1989. Pages 60–69.

[14] Jenkins CI, https://jenkins-ci.org/. Referred 31.3.2015.

[15] SonarQube, http://www.sonarqube.org/. Referred 31.3.2015.

ABOUT THE AUTHORS

PhD student Pekka Mäkiaho MSc, School of Information Sciences, University of Tampere, Finland, E-mail: pekka.makiaho@uta.fi.

…………………………………………………………………………

University lecturer Timo Poranen, PhD, School of Information Sciences, University of Tampere, Finland, E-mail: timo.t.poranen@uta.fi.

…………………………………………………………………………

PhD student Ari Seppi, MSc, School of Information Sciences, University of Tampere, Finland, E-mail: ari.seppi@gmail.com