Evaluation of Students' Modeling and Programming Skills

Birgit Demuth, Sebastian Götz, Harry Sneed, and Uwe Schmidt

Technische Universität Dresden
Faculty of Computer Science

Abstract. In winter semester 2012/13 we started an empirical study evaluating modeling and programming skills in a software project course. We acquired comprehensive data on both modeling and programming activities by means of source code metrics and a survey focused on modeling of 34 successfully completed software projects. In this study we divided student teams by their basic skills and interest in object-oriented software development into groups of project teams and compare survey results as well as source code metrics and the derived team productivity for each group. In the conducted statistical evaluation we detect significant differences between these groups. This conclusion basically firms up Robert France's hypothesis that expert modelers are also good programmers.

1 Introduction

In almost every computer science program students have to learn object-oriented (OO) techniques in software development. Besides programming in an object-oriented (OO) language such as Java, object-oriented modeling topics constitute an important subject matter in teaching. The issue is how to integrate modeling into a software engineering curricula as discussed at software engineering education conferences. So Robert France stressed that modeling should be developed alongside programming [1]. His hypothesis is that expert modelers are good programmers.

Our teaching approach in software engineering courses includes modeling with UML and programming with Java. In a first introductory course we introduce undergraduate students to OO analysis and OO design including using selected design patterns as well as to OO programming including UML2Java transformation based on small applications. In the subsequent project course students have to implement a middlesize application in a work-sharing software development process. Although the topics of the courses have basically not changed over the years we experimented with different didactic approaches in the introductory course. The underlying issue is how should modeling and programming intertwine to educate both modelers and programmers. The idea of our *UMLbyExample approach* [2] is that we already visualize Java code by UML class, object and sequence diagrams at the beginning while in parallel teaching the OO basics in Java. We could show that merging of modeling and programming yields better results both in modeling and programming. In this respect, we

	real projects	simulated projects
passed students	13 (**externals**)	12 (**internals**)
failed students	not intended	9 (**failed**)

Table 1. Team classification with number of teams per group of student teams and used term for the group.

demonstrated the validity of Robert France's hypothesis. However, our observations only included small exercise applications. Our intuitive observations in the following software project course confirm the hypothesis that programmers with good modeling skills produce better quality programs than those with weaker abstraction skills. More empirical studies are needed to investigate the quality of larger programs according to the modeling skills of the programmers.

The remainder of this paper is structured as follows. In Sect. 2 we describe our method to perform the empirical study, classify data used for our evaluation and explain the data obtained of the 34 software projects. In Sect. 3 we present the results of the evaluation of the collected objective and subjective data. The threats of validity are discussed in Sect. 4. Finally we summarize in Sect. 5 our lessons learned and give an outlook to further studies.

2 Method and Data Used for Evaluation

In winter semester 2012/13 we started an empirical study in our software project course[1]. We obtained comprehensive data based on a survey and metrics of 34 successfully realized software projects. Each project team consisted of 5 to 6 students. Project teams were organized by members with similar basic skills in modeling and programming based on exam results in the previous introductory software engineering course. In our software project course about which we reported in [3] we ensure an intensive supervision during the whole software life cycle of the students' applications. In the subsequent evaluation of the realized software projects, we divided the 34 teams into three groups under consideration of their basic skills and their interest in software development (Table 1). Then we analyzed the student modeler and programmer skills for each group. Basic skills in OO software development were estimated by the students' achieved grades in the exam of the introductory course. Generally we discriminate between passed and failed students. *Passed students* passed the exam. *Failed students* failed the exam in fact, but they failed the exam only marginally. That means they got involved in the software engineering course (i.e., students, which almost passed the exam, have been granted participation, too) but they had only a very basic knowledge. All other failed students were not permitted to take part in the project course. Besides the classification of the students by their basic skills in OO software development we asked them if they were interested in *real projects* in industry (i.e. to perform the course *externally*). We offered carefully selected

[1] http://st.inf.tu-dresden.de/teaching/swp2012/

	objective data	**subjective** data
quantitative data	source code metrics team productivity (TP)	survey (T1,T2)
qualitative data	survey (PR)	survey (DP, UML, MC)

Table 2. Classification of the data used for our evaluation (cmp. Table 3).

project tasks of comparable complexity provided by regional software companies. Students could apply for those projects. Typically students that are interested in software development step up to the plate of a real task. However, only students who passed the exam had the chance to take part in a real project. All other passed students received a constructed task–a web shopping application [3]–and therewith worked in *simulated projects* performed *internally*.

To test the hypothesis of Robert France, we performed (1) a guided online survey with the teams of the project course and (2) analyzed the software artifacts provided by each group. That is we collected and examined qualitative and quantitative data, both of which we further divided into subjective and objective data. Most of the survey's questions are qualitative, though there are quantitative questions, too. For example, the teams were asked to estimate their real effort per development phase in weeks. This information is in one sense subjective because the data's accuracy obtained by the students is varying, but in the other sense quantitative. Another question relates to the number of framework packages used, where the answer was extracted by the students from their code and, hence, is considered objective. In all other cases, the information gathered by the survey can be characterized as subjective, whereas the information gathered by examining the code is of objective nature. Table 2 summarizes this classification. The mapped questions are explained in detail in the following and summarized in Table 3 and metrics in Sect. 2.1.

Each team filled out an online survey at the end of the project course. To improve the quality and expressiveness of the answers given by the teams, the respective responsible tutor–who knows well about how the team performed and, hence, is able to distinct correct from fake answers–guided the team in completing the survey. As a consequence, we achieved a response rate of 100%.

Question	Answer Scale	Number of Options
(DP) Design Patterns	Boolean	10
(UML) UML Application	Boolean	7
(PR) Package Reuse	Boolean	14
(MC) Model/Code Consistency	Interval	1–10
(T1) Time Planned per Phase	Integer	Number of weeks
(T2) Real Time per Phase	Integer	Number of weeks

Table 3. Answer scale and number of options of relevant survey questions.

Metric	Range
(COMPL) weighted average program complexity	0 ..1
(QUAL) weighted average program quality	0 ..1
(#OP) Number of Object Points	integer
(#STMT) Number of Java statements	integer
(#FP) Number of Function Points	integer

Table 4. Used source code metrics

The survey comprised 76 questions in total, though only six are of interest w.r.t. Robert France's hypothesis that are listed in Table 3 and explained in detail in the following Sect. 2.1 and 2.2. The remaining questions are intended to improve the overall quality and organization of the project course. For example, by asking for the used development environment or whether the team considered the lecture as helpful for the project course or not.

2.1 Objective data

For quantitative data (metrics) we analyzed the source code of every student project using Harry Sneed's SoftAudit tool [2]. We used Sneed's source code metrics [4] listed in Table 4 to compare groups of student projects.

A further relevant metric characterizing students' skills is their *team productivity (TP)*. For this purpose we required that all students log their spent time in hours for the project. Thus we could determine the total hours of a team over the whole life cycle of its software project (#PH). The team productivity is evaluated as geometric mean of three parameters each as a quantity metric (#OP, #STMT, #FP) per hour as follows:

$$TP = \sqrt[3]{\#OP/\#PH \cdot \#STMT/\#PH \cdot \#FP/\#PH} \tag{1}$$

As qualitative data we considered and investigated the *Reuse* of Java packages provided by the SalesPoint framework. The idea is that we adopt the habit of reusing existing classes instead of inventing new ones. This requires you to know where to look for reusable components (i.e. you must understand the scope and structure of the class libraries you are using). Because of the diversity of used frameworks in real projects, however, we only considered reuse in SalesPoint applications. For each package provided by the Salespoint framework, every team had to specify in the online survey whether they used the package or not.

2.2 Subjective Data

The following five questions providing subjective data have been examined.

(DP): We asked which design patterns have been used by a team. The options to answer this question are a fixed amount of design patterns, which could be

[2] http://www.anecon.com/

Group	COMPL	QUAL	#OP	#STMT	#FP	#PH	TP
externals (bioshop)	0,49	0,62	5411	1411	356	1020	1,24
externals(rest)	0,53	0,57	1967	671	214	1005	0,38
internals	0,48	0,67	4208	1549	117	1020	1,6
failed	0,53	0,63	3440	1322	40	859	0,55

Table 5. Group metrics.

marked as "have been used" or "not used". The considered design patterns were singleton, template method, strategy, state, object adapter, class adapter, factory method, iterator, composite and observer. We selected these patterns as answer options, because only those had been taught in the introductory course.

(UML): We asked in which activities the students applied UML. We considered seven activities: brainstorming, analysis, design, implementation, test, documentation and communication, where the application of UML could be answered with either "yes" or "no".

(MC): We asked for the students impression of model/code consistency. This question, in contrast, did not provide Boolean options, but asked for an assessment on a scale from 1 to 10, where 1 refers to a low degree of fulfillment and 10 to a high degree.

(T1, T2): We asked for the planned and real time spent per development phase and did not provide any predefined options. Instead the number of weeks spent per phase was meant to be provided as an answer. We intended to ask for the time spent in addition to the originally planned effort, which can be derived by interpreting both questions in parallel.

3 Evaluation of the Survey

In the following we present the results of the evaluation.

3.1 Evaluation of Objective Data

Here we list the estimated source code metrics, the team productivity as well as the degree of the reuse of SalesPoint packages.

Metrics. Table 5 summarizes all considered source code metrics and the estimation of the team productivity for each group of student projects (cf. Table 1). In evaluating the data we noticed significant differences within the real projects. Three of the real projects implementing a health-food shop (bioshop) using the SalesPoint framework, produce significantly different metrics than the rest of the real projects (rest). Therefore we had to distinguish between the two groups of real projects. The metrics were estimated for each student project. The *group metrics* listed in Table 5 are estimated by the median of all projects of this group.

The bold printed data in Table 5 show the "best" groups relating to their respective metrics. It could be assumed that externals achieve the best results.

	externals	internals	failed
Design Patterns used (max: 10)	19.23	28.33	22.22
Framework packages reused (max: 14)		61.29	79.36
Application of UML per phase (max: 7)	45.24	48.36	44.44
Model/Code consistency (max: 10)	62.31	62.50	27.78

Table 6. Survey results by group in percent.

However, the internals achieved the lowest program complexity and the highest program quality. According to ISO-9126 software engineering product quality standard scores of more than 0.6 mean "good" quality [3]. Therewith almost all groups achieved good software quality. Internals also showed the highest team productivity. Although the bioshop teams had the highest quantity metrics (at least #OP and #FP) and used at least partly the SalesPoint framework as the internals they could not achieve the same scores as the internal teams. The explanation for this fact and the results of the rest teams is that they had to deal with a real customer and therewith more strict test and maintenance requirements. A further observation is that students in real projects had a higher learning curve (represented by their induction value in Table 7). Failed students showed that the software project course helped them to improve their modeling and programming skills. For example they achieved a higher team productivity than the rest teams. A reason for this fact is that in several failed teams students left and had to be assigned their tasks in the project to the other students.

Reuse of Salespoint Framework Packages. The investigation of the amount of framework packages reused by type of group was expected to show that passed students reused more packages than failed students. The second row in Table 6 shows the average number of packages reused by type of group. For this analysis real projects have been omitted, as these teams did not (or minimally) use the Salespoint framework and, hence, the question was not applicable to these teams. Surprisingly, the survey shows a significantly higher degree of reuse for failed teams, which reused ca. 80% of all provided packages, whereas passed teams only reused ca. 60%. This might be due to the higher abilities of passed students, which more often decided for an own, customized solution instead of reusing (i.e., extending) the provided standard solutions.

3.2 Evaluation of Subjective Data

In the following the results gathered by the survey (cf. Table 6) will be presented.

Application of Design Patterns. The first row of table 6 depicts the average number of design patterns used per group. The teams with real tasks made the least usage of design patterns. These teams applied less than 20% of the patterns

[3] http://www.anecon.com/downloads/System_Assessment_-_Harry_Sneed_02.pdf

	Induction	Analysis	Design	Implement-ation	Mainte-nance
externals	0.462	0.269	0.192	1	0.846
internals	0	0.083	0.25	0.917	0.917
failed	0.111	0.333	0.222	1	1.222

Table 7. Average Error in Time Estimated per Phase by Group in Weeks.

taught in the introductory course. For internals, those teams, which passed the exam, are clearly superior to those that failed (ca. 28% versus 22%).

Application of UML per Activity. The third row in Table 6 shows the number of activities, where UML has been used in percent of all seven activities (cf. Section 2.2). The expected result was that internals made the strongest usage of UML, followed by externals and, finally, by failed students. The collected data confirms our intuitive observation. Internals utilized UML more often than externals, which is due to the increased induction and customer communication efforts of externals. Moreover, the survey reveals that internals are superior to failed teams in terms of UML usage and, thus, supports our hypothesis.

Model/Code Consistency. The fourth row of Table 6 depicts the degree of consistency of models and code for internals, externals and failed teams as perceived by the students (i.e., these numbers do not rely on artifact analyses). We expected the internals to be superior to the other groups, followed by externals and, finally, failed teams. The reason for internals to outperform externals is the higher complexity of real tasks in comparison to simulated tasks, which increases the difficulty for external passed students to keep their models and code consistent. As Table 6 shows, our expectations are confirmed by the survey.

Difference Between Planned and Actual Time per Development Phase. The investigation of the difference between the planned and actual time required by each team per team confirms our intuitive observations, too. The results are shown in Table 7. Externals spent, in comparison to internals and failed teams, the most additional time in the induction phase. This is due to the additional effort required to work with real world frameworks. Failed teams spent the most additional time in the maintenance phase. This indicates a lower quality of their code in the implementation phase, which confirms our hypothesis.

4 Threats to Validity

The greatest threat to the validity is that the source code metrics of four real tasks were not estimated. Three of these projects were written in Ruby for which we had no software evaluation tool. The fourth project was indeed a Java project, but it was developed in a model-driven way: large parts of the code were generated. Furthermore, we excluded the web GUI code from our

evaluation. Our internal evaluation showed that around 50% of the project code is GUI code (JSP, JSF and JS tags). Therefore it can be assumed that the absolute team productivity is significantly higher than the estimated one listed in Table 5. Notably, all other teams used the same programming language (Java). Using exam results as an indicator for expertise is another threat to validity, as some students are more eligible to a continuous assessment and show much less competency in exams than they actually have. Yet another threat to validity is the effect of social interaction in project teams, which highly depends on the personalities and, thus, is hard (if not impossible) to be taken into account. Finally, the guidance in answering the online survey could have influenced the answers given, due to social effects between the teams and the tutor.

5 Conclusion

We conducted an empirical study evaluating modeling and programming skills in a software project course and evaluated comprehensive data capturing both modeling and programming activities based both on a survey and source code metrics of 34 successfully completed software projects. As a result of the statistical evaluation of the data, we detected significant differences between students of different qualification in their basic skills and their interest (in terms of their engagement in a real instead of a simulated task) in software development. However, we could also show that our teaching approach leads in most cases to "good" program quality including such model requirements as reuse of frameworks, model/code consistency and use of design patterns. This conclusion confirms the hypothesis that expert modelers are also good programmers.

Furthermore we learned that the use of a common application framework constructed for teaching purposes helps to improve the program quality. In the next course we plan to supervise the software development process in a more rigerous way and to evaluate the UML design metrics in addition to the Java source code metrics. The use of the SoftAudit tool for conformance checking and quality measurement allows tutors to guide the whole life cycle of the projects in a more systematic manner.

References

1. Bézivin, J., France, R.B., Gogolla, M., Haugen, Ø., Taentzer, G., Varró, D.: Teaching modeling: Why, when, what? In Ghosh, S., ed.: MoDELS Workshops. Volume 6002 of LNCS., Springer (2009) 55–62
2. Demuth, B.: How should teaching modeling and programming intertwine? In: Proc. of the 8th Educators' Symposium, ACM (2012)
3. Zschaler, S., Demuth, B., Schmitz, L.: Salespoint: A java framework for teaching object-oriented software development. Science of Computer Programming (2012)
4. Sneed, H.: Software in Zahlen. Carl Hanser Verlag München (2010)