Research plan: Software Quality in Education

Procedures, didactics, and tools

Sylvia Stuurman, Harrie Passier, Lex Bijlsma, Harold Pootjes

August 28, 2017

1 Introduction

One of the aims of education in the field of software engineering is to teach students to value software quality. We would like students to be able to evaluate software with respect to quality, we would like them to be able to produce software with quality, and we would like them to know how to optimize software quality. Software quality is composed of characteristics of the software; those characteristics can be categorized according to the standard software quality framework [10].

We discern three ways to produce and guarantee software quality:

- design and code in such a way that the software adheres to software design principles (abstraction, coupling and cohesion, decomposition and modularization, encapsulation, separation of interface and implementation, sufficiency, completeness and primitiveness, and separation of concerns) [4]
- refactor designs and code when one finds shortcomings during inspection
- test code, to check for bugs and mistakes (often in a process of constant refactoring and testing)

There are various problems with respect to software quality in education, and these problems might be related. For instance, learning to program is very difficult for many students, in particular when they do not receive explicit problem solving strategies [8]. One of the design principles, abstraction, is very hard to grasp for many students [14]. Abstraction may be learned, but most teachers explain their subject at the highest level of abstraction [17]. Many students are not interested in software quality, for instance, in testing [1].

We focus on two problem areas.

Lack of enthusiasm for software quality On the one hand, students find it difficult, or perhaps boring, to pay attention to software quality. Testing their code is tedious; students work towards a result and are happy when their program seems to 'work'. It seems superfluous to pay much attention to a good design, and to maintainable code [1]. Finding ways to stimulate students' enthusiasm for software quality is one of the goals of our research.

Lack of systematic problem solving strategies Explicit guidance in problem solving is needed when one teaches complex tasks [12]. There is a need for systematic problem solving strategies for producing flexible designs, maintainable code, and for finding test cases. Most teachers, for instance, focus on the syntax, on knowledge [8] and do not gradually teach abstraction [17]. Students should initially be required to use these strategies, and, once they are mastered, the strategies take on the character of a tool to be employed when useful or necessary: experts do not use the step-by-step procedures that novices need [6].

Online education In a setting of online education, both problems become even more pregnant. It is more difficult to create enthusiasm for software quality online, and because students do not see their tutors working on a problem (with a chance that they learn problem solving strategies implicitly, and that they learn implicitly to pay attention to software quality), explicit systematic problem solving strategies are needed more.

1.1 Enthusiasm of students

The lack of explicit systematic problem solving strategies might be one of the reasons that students often do not show enthusiasm to pay much attention to software quality. Another probable cause is that they are used to be asked to deliver something that 'works': they simply pay attention to what is asked, and not more. A third cause might be that students simply do not know which quality goals they should set themselves, and how they can check whether they reached these goals. Students often do not know exactly what 'a correct program' means (when a program compiles, they consider it 'relatively correct', for instance) [13].

Teachable problem solving strategies Problem solving strategies should be teachable. Strategies should students really help to move forward. An example of a strategy that is often too difficult for students is the rule to write functions by first specifying them, using a specification language. In practice, students first write the program code and try to derive their specification from their code, to satisfy the teacher. This is understandable, because a specification has, in general, a higher abstraction level than a program, and abstraction is something that should be learned gradually [17]

Authentic tasks Authentic tasks are indispensable when teaching students complex tasks, and they may also help them to warm to paying attention to quality. Students

found it much easier to define what a correct program is when the assignment was about their own family tree for instance ('My uncle should be there') [17].

Make it more fun We would like to make paying attention to software quality more fun, for instance, by finding as much bugs as possible in the software of other students, or by providing tools that take the tedious parts out of their hands.

1.2 Systematic problem solving strategies

Learning how to design and program according to general principles can be seen as a complex task, and the same applies to refactoring and testing. Learning complex tasks should be based on real-life authentic tasks [20]. Students should be provided with support and guidance while solving such a task. That guidance should tell students how to recognize an acceptable solution, and should provide guidance to the solution process: procedural information is required when one offers authentic tasks to enable complex learning [12].

This means that one should provide systematic problem solving strategies for design, programming, refactoring and testing tasks when one teaches students how to produce software with quality. These strategies preferably take the form of direct, step-by-step instruction [26].

Systematic problem solving strategies can have several forms, which may range from rules of thumb to very precise steps, (almost) as a computer algorithm. An example of a strategy in the form of rules of thumb is the Unified Process for designing software using UML as graphical notation [15]. An example of a strategy in the form of precise steps is the procedure to design precise XML content models using the regular expression language [21, 22]. Examples of guidelines somewhere between these two extremes are the guidelines how to design functional programs [9] and the method to design thread-based algorithms [3].

Very important as well, is to explicitly state quality goals, and to show students how they can check whether they have reached these goals.

Developing problem solving strategies for design, programming, refactoring and testing has more benefits, aside from the fact that it will help students in learning to solve tasks. When you would like to model intelligence in a computer program, for instance with the purpose of automatically grade assignments or assist students while elaborating exercises, one needs to [5]:

- have knowledge about the domain,
- be able to reason with that knowledge,
- and have knowledge about how to direct or guide that reasoning.

This means that, once you have explicit problem solving strategies for a certain domain to your disposal, you have one of the necessary ingredients for (intelligent) tools that will guide students or assist teachers within that domain.

Therefore, systematic problem solving strategies have our interest from three perspectives (see figure 1):

- Strategies per se - How can we solve a particular class of problems? Which steps should be taken, what domain knowledge is needed, and what knowledge is needed to guide reasoning? In many areas of computer science, no procedural guidelines are available, or they have the form of rules of thumb. We would like to make them more precise, and make them explicit.
- Teaching - How can we incorporate these procedures in our teaching material? Does this lead to better learning results of our students? Do they produce better software products? Do students have a better understanding of the subject?
- Tools - How can we use these strategies to create tools? Can we use these tools to support the learning processes of our students?

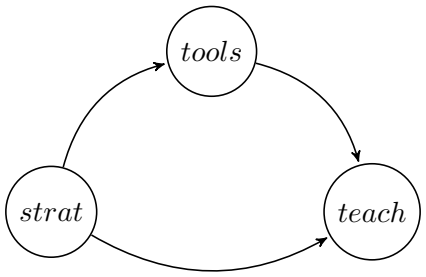

Figure 1: Relationship between procedures, tools, and learning

Because of the lack of explicit systematic problem solving strategies in many areas of computer science, our goal is to develop strategies where they lack, to make rules of thumb more precise, to use these strategies as part of our learning material and didactics, and to implement these strategies, if possible, in the form of tools supporting the learning processes.

In the remainder of this proposal, we explain how we envision our research in the fields of the design of software, refactoring software, and testing software. For each subject, we propose some research questions as a starting point.

2 Design

Of all topics for which software engineering practitioners need training after their formal education, software design and patterns score the highest [16]. This study dates some time back, but it is probable that software design and patterns still are important topics for software engineers. With respect to programming, we know that the level of understanding after introductionary courses is much lower than we would hope.

Many errors are outside the domain of construction, i.e. thin application domain knowledge (besides fluctuating requirements, communication problems, ...). Misunderstanding the design is a recurring theme. The better the application area is understood, the better the architecture is. The solution might be to pay explicit attention to domain modeling.

Good modeling and prototyping is much more effective then unit testing. 'The earlier an error is inserted, the more entangled it becomes in other parts'. 'If you want to improve your software, don't just test more but develop better'.

The master course Design patterns teaches students how to design software that is flexible with respect to future changes: design for change. In the course, design patterns are used to show how one can reach flexibility in object oriented systems.

In previous research, we found that students have several problems in learning to design for change [25]: they find it difficult to use the strategy for problem analysis that we teach, and they find it difficult to use design patterns to create flexible software (instead, they often try to fit in as many patterns as possible).

Our research on this subject aims to improve the course.

Another direction for our research on software design is based on the fact that there is a trend in programming languages to support more than one programming paradigm. In many cases, one can choose a functional solution as well for an object oriented solution. It is difficult to choose the 'right' solution, because of this choice.

2.1 Research questions

Problem analysis The approach that we teach to analyse the problem does not seem to work: students find it difficult to apply, and we cannot see signs that it helps them in their design. We would like to create an alternative approach, loosely based on domain-driven design. It would be interesting to find out whether such an alternative strategy helps students better.

Guidelines Students struggle when having to apply design patterns for a given problem. Are we able to offer more detailed guidelines? Are guidelines on how to refactor a design, using patterns, a valid option?

Tool support To come closer to tool support, with immediate feedback while designing, there are various problems to solve. Are we able to automatically detect patterns? Are we able to translate our guidelines into UML diagram conforms to one of those strategies? Can we fund 'buggy solutions' for some problems (solutions that are often given by students, but that are false)?

Motivating examples It seems difficult for students to grasp the goal they are supposed to try to achieve (design software with quality); instead, they try to apply as many design patterns as they can, because they think that that is what the teachers want to see. Does it help to start the course with some motivating examples? What are good examples?

Functional versus object oriented Can we derive guidelines to choose between a functional or an object oriented approach? Perhaps the key is to link such a choice to the place in the architecture. When state is an integral part (the user interface for instance), object orientation might be the better choice, whereas one could choose a functional style for places that can be designed without state (the domain).

3 Refactoring program code

One of the areas that lacks systematic problem solving strategies is programming. In contrast with older methods [28, 7, 29], modern textbooks typically lack guidelines [2], with one exception: 'How to design programs', which is a textbook on functional programming [9]. The result of this lack of procedural guidelines is that learning to program is hard, and so is teaching. One of the solutions is to have the teacher perform live programming (which almost always involves refactoring) [2]. Another solution is to develop explicit guidelines.

One of the possible approaches is to provide guidelines on how to create code that works, together with guidelines on how to refactor 'naive' solutions. The purpose of refactoring is to create code that conforms to the design principles that exist in the field of software engineering [4]. These design principles are typically goals that we would like our students to achieve when coding or designing code. Guidelines for refactoring, therefore, could help students to achieve those learning goals [23].

One of the complications with refactoring is that many programming languages support several programming paradigms. Scala for instance, is both fully functional and object oriented. JavaScript has many possibilities to program in a functional style, and has also support for an object-oriented style. An important kind of refactoring, is, therefore, to refactor code from a more procedural style towards a more functional style.

3.1 Research questions

We would like to explore the following questions:

Functional style A pure functional style has many advantages compared to a style that relies on maintaining state, for instance with respect to testing. Refactoring JavaScript to a more functional style is therefore a research subject: can we detect code that can be refactored automatically? Can we refactor automatically?

Detect the need for refactoring Related to the previous subject is the question of how to detect code that might profit from refactoring in general. There is a relation with code quality.

Metrics for JavaScript Metrics are needed both to validate whether refactored code has better quality than the code before the refactoring, and to detect code that might benefit from refactoring. JavaScript is a difficult language to use metrics, because it has procedural, object-oriented and functional characteristics, and its

object-orientation is object-based instead of class-based. We would like to create metrics that work for JavaScript, to measure code quality.

Code quality We use the Swebok design principles to evaluate code [23], but we could also use a validated model for assessment of code quality [24]. We would like to explore how these two evaluation models compare, and also whether it is possible to create more specific refactoring guidelines using the model for code quality.

Understandability of refactored code Refactored code is more easily maintainable, but sometimes, refactored code is less easy to understand then 'naive' code (for instance because of the higher level of abstraction). A lesser degree of understandability obviously has an impact on changeability, which may explain the fact that in a recent study, the positive effect on changeability of refactored code was very small, and the effect on understandability negative [11]. We would like to explore whether we can find guidelines for documentation that add to the understandability of refactored code, and check how students perform in understanding and changing refactored code versus 'naive' code.

Positive effect on understanding programming The 'hunch' is that refactoring using explicit guidelines helps students in their understanding of programming. The level of understanding of students, after introductionary courses, is, in general, rather low [27, 18, 19]. We would like to investigate whether explicit guidelines for refactoring indeed improve the understanding of students with respect to programming.

Guidelines for automation If we are able to express our guidelines in such a way that a computer may detect whether a specific guideline may be applicable, we are closer to the realization of a tool for assistance for students while programming.

4 Testing

In education, writing tests is often something that comes last (or does not come at all). Teachers find it difficult to pay attention to writing tests because they feel that would take precious time from teaching, for instance, programming, and students try to focus on getting the program to work. When they are forced to test, they find it difficult to decide which test cases to use.

With respect to writing tests, there are, therefore, two main questions: how do we get students (and teachers) to consider testing as part of programming, and how can we help students decide which test cases to use.

4.1 Research questions

Feedback on tests Are we, for a specific function, able to automatically derive a test set that covers all possibilities? Are we able to compare the test sets that a student gives with such a test set?

Tests supporting refactoring An important step in refactoring is specifying tests. Are we, for a specific function, able to automatically derive a test set that covers all possibilities? Are we able to compare the test sets that a student gives with such a test set?

Automatically derive test cases Can we derive test cases for functions in JavaScript automatically? Would it help is we would use TypeScript? Would it help if we would specify pre- and postcondition?

Adjust tests after refactoring Refactorings often have an impact on the test set. How can we provide feedback on adjusting test code, and can we automatically adjust test code after a refactoring?

Tests as integral part of programming How can we stimulate teachers and students to regard testing as an integral part of programming? Would it help to make a game of finding bugs in code of other students?

5 Relevant Conferences and journals

5.1 Conferences

ITICSE , `http://www.sigcse.org/events/iticse`, Conference on Innovation and Technology in Computer Science Education, This conference brings together delegates from all over the world to address pressing issues in computing education.

Koli Calling `http://www.kolicalling.fi`, Koli Calling is one of the leading international conferences dedicated to the exchange of research and practice relevant to the scholarship of teaching and learning and to education research in the computing disciplines. Koli Calling aims to publish high quality papers that combine teaching and learning experiences with solid, theoretically anchored research.

CSECS `http://www.csecs.org`, Computer Science and Education in Computer Science, This is a refereed conference event. Papers in a wide range of Computer Science research areas ranging from Software Engineering to Information Systems Security are considered for presentation and publication in our proceedings. Computer Science Education topics range from introduction and evaluation of computing programs, curricula, and online courses, to syllabi, laboratories, and teaching and pedagogy.

ICCSE `http://ieee-iccse.org`, International Conference on Computer Science & Education

ICER `http://icer.hosting.acm.org`, ICER is an annual international conference sponsored by ACM and its SIGCSE special interest group. The conference is focused specifically on the computing education research discipline that is, the study of how people come to understand computational processes and devices, and how to improve that understanding.

Frontiers in Education , http://fie-conference.org, a highly-respected major international conference focusing on educational innovations and research in engineering and computing.

CSEE&T , http://conferences.computer.org/cseet/, an international peer-reviewed conference, publishes papers in all areas related to software engineering education, training and professionalism.

ICSE -SEET , http://2016.icse.cs.txstate.edu/educationTraining, Software engineering, education and training,

CSERC ,an international forum for researchers with interests in all aspects of computer science education.

5.2 Journals

TOCE , http://toce.acm.org, ACM Transcactions on Computing Edication, covers diverse aspects of computing education by publishing papers with a scholarly approach to teaching and learning, a broad appeal to educational practitioners, and a clear connection to student learning

IEEE Transactions on Education , http://www.ewh.ieee.org/soc/es/esinfo.html, Educational research, methods, materials, programs, and technology in electrical engineering, computer engineering, and fields within the scope of interest of IEEE.

Computer Science Education http://www.tandfonline.com/action/journalInformation?show=aimsScope&journalCode=ncse20, Taylor & Francis Online, high-quality papers with a specific focus on teaching and learning within the computing discipline that are accessible and of interest to educators, researchers, and practitioners alike.

International Journal of Qualitative Studies in Education , http://www.tandfonline.com/action/journalInformation?show=aimsScope&journalCode=tqse20, The aim of the International Journal of Qualitative Studies in Education (popularly known as QSE) is to enhance the practice and theory of qualitative research in education, with 'education' defined in the broadest possible sense, including non-school settings.

JEE Journal of Engineering Education, http://www.asee.org/papers-and-publications/publications/jee, serves to cultivate, disseminate, and archive scholarly research in engineering education.

References

[1] E. G. Barriocanal, M.-Á. S. Urbán, I. A. Cuevas, and P. D. Pérez. An experience in integrating automated unit testing practices in an introductory programming course. ACM SIGCSE Bulletin, 34(4):125–128, 2002.

[2] J. Bennedsen and M. E. Caspersen. Revealing the programming process. ACM SIGCSE Bulletin, 37(1):186–190, 2005.

[3] A. Bijlsma, H. Passier, H. Pootjes, and S. Smetsers. Methodical concurrency design in education, part i: Race conditions. Technical Report TR-OU-INF-2015-1a, Department of Computing Sciences, Open Universiteit, 2015.

[4] P. Bourque and R. E. Fairly, editors. Guide to the Software Engineering Body of Knowledge version 3. IEEE Computer Society, 2014.

[5] A. Bundy. The Computer Modelling of Mathematical Reasoning. Academic Press, 1983.

[6] N. Cross. Expertise in design: an overview. Design Studies, 25(5):427 – 441, 2004.

[7] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors. Structured Programming. Academic Press Ltd., London, UK, UK, 1972.

[8] M. de Raadt. A review of australasian investigations into problem solving and the novice programmer. Computer Science Education, 17(3):201–213, 2007.

[9] M. Felleisen. How to Design Programs: An introduction to programming and computing. MIT Press, 2001.

[10] ISO/IEC. ISO/IEC 25010, system and software quality models. Standard, International Organization for Standardization, 2011.

[11] S. Kannangara and W. Wijayanayake. An empirical evaluation of impact of refactoring on internal and external measures of code quality. arXiv preprint arXiv:1502.03526, 2015.

[12] P. A. Kirschner, J. Sweller, and R. E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational psychologist, 41(2):75–86, 2006.

[13] Y. B.-D. Kolikant. Students' alternative standards for correctness. In Proceedings of the first international workshop on Computing education research, pages 37–43. ACM, 2005.

[14] J. Kramer. Is abstraction the key to computing? Communications of the ACM, 50(4):36–42, 2007.

[15] C. Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition). Prentice Hall PTR, Upper Saddle River, NJ, USA, 2009.

[16] T. C. Lethbridge. Priorities for the education and training of software engineers. Journal of Systems and Software, 53(1):53–71, 2000.

[17] R. Lister. Concrete and other neo-piagetian forms of reasoning in the novice programmer. In Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114, pages 9–18. Australian Computer Society, Inc., 2011.

[18] L. Ma, J. Ferguson, M. Roper, and M. Wood. Investigating the viability of mental models held by novice programmers. ACM SIGCSE Bulletin, 39(1):499–503, 2007.

[19] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. ACM SIGCSE Bulletin, 33(4):125–180, 2001.

[20] M. D. Merrill. First principles of instruction. Educational technology research and development, 50(3):43–59, 2002.

[21] H. Passier and B. Heeren. Modeling XML Content Explained. Technical report, Department of Information and Computing Sciences, Utrecht University, 2011.

[22] H. Passier and B. Heeren. Modeling XML content models explained. In Proceedings of the MCCSIS. IADIS, 2011.

[23] H. J. Passier, S. Stuurman, and H. Pootjes. Beautiful javascript: How to guide students to create good and elegant code. In Proceedings of the Fourth Computer Science Education Research Conference (CSERC), pages 65–76, New York, NY, USA, 2014. ACM Digital Library.

[24] M. Stegeman, E. Barendsen, and S. Smetsers. Towards an empirically validated model for assessment of code quality. In Proceedings of the 14th Koli Calling International Conference on Computing Education Research, pages 99–108. ACM, 2014.

[25] S. Stuurman, H. Passier, and E. Barendsen. Analyzing students' software redesign strategies. In Proceedings of the 16th Koli Calling International Conference on Computing Education Research, pages 110–119. ACM, 2016.

[26] J. J. van Merriënboer and P. A. Kirschner. Ten Steps to Complex Learning, a systematic appraoch to four-component instructional design. Taylor & Francis, New York, NY, USA, second edition, 2013.

[27] J. L. Whalley, R. Lister, E. Thompson, T. Clear, P. Robbins, P. Kumar, and C. Prasad. An australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies. In Proceedings of the 8th Australasian Conference on Computing Education-Volume 52, pages 243–252. Australian Computer Society, Inc., 2006.

[28] N. Wirth. Program development by stepwise refinement. Communications of the ACM, 14(4):221–227, 1971.

[29] E. Yourdon and L. L. Constantine. Structured design: Fundamentals of a discipline of computer program and systems design. Prentice-Hall, Inc., 1979.