Teaching Advanced Software Design in Team-Based Project Course

Stan Jarzabek
Dept. of Computer Science
National University of Singapore
Singapore
stan@comp.nus.edu.sg

Abstract

Skillful design remains one of the critical success factors in long-lived software projects. Design fundamentals have been established and are pretty stable. How do we teach design in-the-large to equip our graduates with design skills relevant to a plethora of changing software technologies and emerging new application domains? Today programs are built on top of functionalities provided by software platforms. Most often, developers extend existing systems rather than develop from scratch. Programming with application program interfaces (API) that allow newly written code to call middleware or existing application software has become a norm in software industries. While the details of API mechanisms heavily depend on a specific platform or application, the principles behind API design are universal, and can be taught in project courses designed for that purpose. Working knowledge of API design principles helps students faster adapt to new and changing technologies. In the paper, we describe a teaching methodology and 10 years of experiences teaching advanced design in a team-based software engineering project course. Our course builds around fundamental concepts in API design and use.

1. Software Project Courses[1]

Abstraction, decomposition or separation of concerns are common sense principles to tame the complexity of problem solving in all engineering disciplines, including software development. We can understand their importance in the theory, but it is the practice that transforms the knowledge into the skill.

Introductory assignment-based courses in undergraduate Computer Science curricula expose students to software design principles. But we often find that enforcing principled software development in small assignments is artificial and ineffective: First, assignment problems are too small to justify the need for principles. Students know that small programs can be effectively "hacked", so they do not experience the real need and value of applying design principles. Second, in assignments, specific design issues are considered one at the time, while in the reality, developers deal with a web of many inter-dependent design concerns.

It follows that students can experience and appreciate the benefit of design principles and "best practices" only in a frame of larger projects than assignments. Team-based project courses create an opportunity to teach software principles in the situation where application of principles is truly needed and beneficial.

Many types of project courses are taught in university curricula, each focusing on training some specific development skills:

1) Industrial attachments in which students work on real-world problems in industrial settings.
2) Project courses in which students work on problems in various application domains under supervision of faculty members, experts in a given domain. Sometimes such courses build on real-world problems provided by industry.

[1] Disclaimer: This paper contains only author's opinions and does not represent the opinion of NUS.

978-1-4673-5140-9/13/$31.00 © 2013 IEEE CSEE&T 2013, San Francisco, CA, USA

3) Project courses in which students learn advanced software design principles and apply them in their projects. As faculty members need scrutinize in detail design artifacts to provide feedback to students, such projects must be supervised by faculty members specializing in software engineering, and well-versed with problem and solution domain students work with.
4) Projects developed from scratch versus projects in which students extend existing software.
5) Projects based on a specific software platform such as .NET, JEE, service, mobile device or Facebook.

Team work, communication and writing skills can be trained in all of the above project courses. Other skills are quite difficult to accommodate in the frame of a single project course. For example, in project types 1) and 2) the goal is to expose students to the reality of fuzzy, ill-defined and changing requirements. Fuzzy requirements and software design are not only the two hallmarks of software development, but also hard and wicked problems [1] that are difficult to teach in the frame of a single course. Project courses that expose students to fuzzy and changing requirements tend to be less structured and rigorous than courses that teach students application of design principles. When teaching application of design principles (project course type 3), we should give students sample design sketches, and lots of detailed feedback on their initial attempts to refine the design. Supervisors need be intimately familiar with a problem domain and design solutions to provide effective guidance for students. This may be quite difficult in projects types 1) and 2).

In this paper, we describe a team-based project course focused on software design we have been teaching at the National University of Singapore. Our course builds around fundamental concepts in API design, independently of a specific software platform (.NET, JEE or Facebook) that is being used. We describe an overall course framework, motivation, teaching methodology, and 10 years of experiences teaching the course. We hope that our approach and experiences can be useful for others who teach or plan to teach courses that focus on advanced design.

2. Course Overview

In undergraduate Computer Science programme at the National University of Singapore (NUS), students take a general software engineering course in the 2nd year of study. We have experimented with various approaches to teaching project courses. Although our students participated in the industrial attachment program and did projects proposed by faculty members, industry surveys were consistently signaling problems related to weak development, problem solving and communication skills, and generally slow start in industrial projects. When exposed to real world pressures, our students did not know how to take advantage of what they had learned. Students tended to perceive software engineering principles as obstacles rather than tools that can help them better complete the project work.

To address these problems, in 2000 we introduced a new team-based project course CS3215 to teach design principles in problem-based way, through architectural concepts and iterative development process. CS3215 was offered as a one-term, intensive course, with modular credits equivalent to two regular courses. We reported initial findings teaching CS3215 at CSEE&T conference in 2005 [6]. Since then, the course evolved into two-term course CS3201/CS3202, shifting from Java to C++ as an implementation language.

The two-term course formula introduced more lectures and tutorials to expand formal treatment of software design principles. Our course builds around concepts of architectural design and application program interfaces (API) [14][12]. Students learn to justify design decisions from architecture to detailed implementation level. Initial analysis and architectural design is followed by iterative development. In addition to applying design principles in

practice and on larger scale, we emphasize program reliability (assured by reviews and testing), reusability, extensibility and quality documentation.

Exceptional individuals are born with a sense of what makes a good design. Most of us must see examples of good design before we can come up with good design on our own. Therefore, we teach design by example, providing students with design sketches and then much feedback on their attempts to refine the design.

Working in teams of six, students do modular decomposition, architecture design and learn to specify component interfaces. This know-how is addressed in lectures and practiced in tutorials, and is a prerequisite for team work. We individually assess students' understanding of design principles and methods required in the project to ensure that all the students understand basic design methods and can effectively contribute to the team work. This teaching strategy considerably reduced conflicts in teams.

Having set up preliminary software architecture, students follow up with implementation (10-15KLOC) applying agile, iterative development model. At the end, students present their projects, we evaluate project report and test students' programs using automated testing tool with 500 test cases. Students read *The Mythical Man-Month* by F. Brooks [3] and present an expose on the topic of own choice.

3. The Focus on Application Program Interfaces (API)

Today programs are rarely built from scratch. They are written with reuse of services provided by the underlying software platform. Sometimes, new code must extend existing systems. Application program interfaces (API) allow newly written code to interact with the middleware or existing application software. Programming with APIs has become a norm in software industries. Developers embarking on a new job in a company often start by learning APIs of software technologies used in that company. APIs can be complex, imperfect and poorly documented. The benefits of reuse via APIs are often hindered by steep learning curve to master APIs. Superficial understanding of APIs is not sufficient to develop robust software on top of those APIs. In case of errors, developers need insider's knowledge of what's behind APIs to be in the position to pinpoint and rectify the problem.

While the details of API mechanisms heavily depend on a specific platform or application, the principles behind API design are universal, based on concepts of modularization and information hiding proposed by Parnas in 1972 [14]. These concepts led to Abstract Data Types (ADT) [12], and then to the development and wide-spread adoption of Object-Oriented programming languages. Component-based platforms (.NET™ or JEE™) and service-oriented architectures rely on APIs.

Our students learn about program interfaces in programming courses (public interfaces to classes), study Abstract Data Types (ADT) in data structures courses, and learn how to hide changeable design decisions behind public interfaces in level-2 software engineering course. These courses are prerequisites to CS3201/CS3202. In an advanced design course, students should learn how to apply these concepts together, on the larger scale. For that, students should design and use application program interfaces (API) not only to hide implementation decisions, but also to interface system components that are developed by different members of a team.

Many students face a great deal of problems to apply interface concepts learned in earlier courses, in a project situation of a bigger scale. In preliminary courses, students deal with one, relatively small ADT at the time. In the project, students must perceive APIs for a web of interrelated components. In addition, the design of APIs must take into account the needs of multiple clients interfacing to a give component. The scale of the project and the complexity of information that must be hidden behind APIs creates a close to real-world challenge for API design.

4. Selecting a Platform and a Model Problem for the Course

It is tempting to base a project course on one of the popular platforms used in industry to let students work with real production APIs. At NUS, we expose students to Facebook and mobile device platforms/applications in a number of project courses. These courses have a great educational value, and are liked by students. Still, we see some drawbacks of selecting an industrial platform for advanced design course. First, rather than spending time learning platform-specific development strategies and APIs, we would like to focus on fundamentals shared by all platforms. Struggling with details (and often imperfections) of platform mechanisms, students may not see "the forest from the trees", i.e., design and API fundamentals shared by all platforms from specifics of a given platform. Second, working with an industrial platform, students learn how to use APIs, but they can't practice the design of APIs. Project course creates an opportunity to expose students to the interplay between API design and use.

For these reasons, rather than using an industrial platform, we decided to base our project course on a system whose componentization involves non-trivial APIs. We selected a Static Program Analyzer (SPA) for the course. SPA helps programmers understand code during maintenance by answering queries about various program properties. For example:

Q 1. Which procedures are called by procedure "P"?
Q 2. Which variables have their values modified in some procedure called by "P"?
Q 3. Is there a control flow path between statements at lines 20 and 620?
Q 4. Which assignments can be affected (in terms of data flow) by statement at line 5?

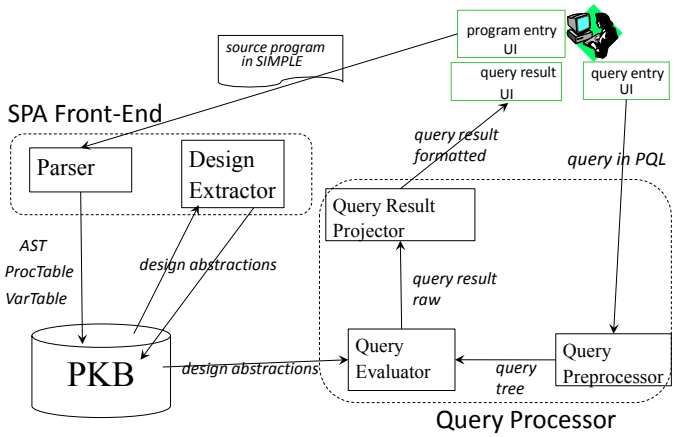

Figure 1. Static Program Analyzer: Component Architecture

An interesting class of questions about programs can be answered automatically based on static analysis of source code. SPA Front-End (Figure 1) parses source programs, and computes program design abstractions such as Abstract Syntax Trees (AST), symbol tables, procedure call graphs, Control Flow Graphs (CFG), and information about variable usage (modifications and references). These design abstractions are stored in the Program Knowledge Base (PKB). A developer writes questions about source program properties in an SQL-like semi-formal Program Query Language (PQL). SPA's Query Processing sub-system checks queries for correctness and evaluates them. Query results are presented to the developer who can refine queries or ask yet other queries to produce complementary program views.

The following examples of PQL program queries formalize questions Q1 – Q6 about program properties:

procedure q; assign a; while w; *//these are declarations of variables denoting program entities (e.g., procedure, while loop or assignment) that can be referred to in queries*
Q 1. Select q such that Calls ("P", q),
Q 2. Select v such that Modifies (q, v) and Calls ("P", q)
Q 3. Select BOOLEAN such that Next*(20, 620)
Q 4. Select a such that Affects (5, a)

The difficulty of an SPA project can be easily scaled up or down to suit the students' background and the required workload of the course.

5. Course Structure

Students complete SPA analysis, architectural design with API specifications and prototyping (shown as iteration 0 in Figure 2) in the first term (CS3201), and continue with iterative development in the second term (CS3202). The SPA problem and design methods to be used in the course are described in a 100-page Project Handbook [8]. The Handbook is used by both instructors teaching the course and students taking the course.

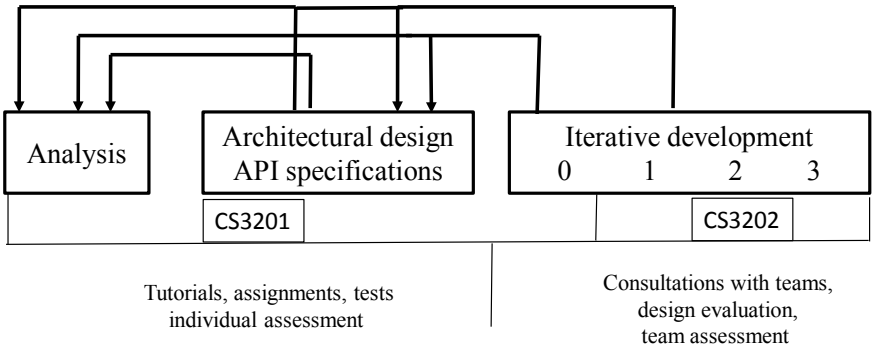

Figure 2. Course activities

During initial SPA analysis and architectural design, we conduct conventional lectures, assignments and tutorials. Lectures introduce students to the SPA problem and explain design methods for the project. After briefly reiterating on design principles in general, students are shown examples of application of principles in the context of SPA.

Initially, students work in groups of three, but we assess their knowledge of a problem domain and design methods individually in two tests. This ensures that all the students are prepared for follow up development, and can contribute to team work. Not all difficulties with course concepts can be spotted in written assignments and tests. Interactive tutorial sessions allow us to observe how students think and directly address the sources of problems. Problems such as API discovery and specifications are ideal topics for tutorial sessions.

Students form teams of six during architectural design. They implement an SPA prototype and then proceed to iterative development.

Once communication in a team is established through PKB API, teams split into groups of three. PKB group works on SPA Front-End, while PQL group develops a Query Processor. Both groups participate in PKB design, playing roles of API designers and users.

During iterative development, teams meet weekly with their course instructors who now play roles of supervisors. Supervisors are intimately familiar with the problem domain and solution, so they can effectively play roles of SPA domain and design experts. Supervisors scrutinize design and code artifacts, students get immediate feedback on their progress, refining and improving their design throughout project iterations.

At the end of the course, students present their project. We evaluate their design solutions and test the conformance of students' programs to requirements using our regression testing tool comprising 500 test cases.

6. Teaching Methodology

6.1 Iterative Development Process

Industries report many benefits of iterative development. Developers can scale difficulties, gradually investigate solutions, and early identify errors through frequent system integration.

To observe the above benefits, iterations must be properly planned. Having completed initial problem analysis, architectural design and prototyping, students are in the position to plan development iterations. Iterations slice SPA horizontally, so that at the end of each iteration students complete a mini-SPA that implements a subset of final SPA functionalities. A mini-SPA is operational and can be tested against SPA functional specifications. Horizontal slicing ensures that all major components of SPA Front-End and Query Processor undergo unit and integration testing in each iteration. This allows students to validate the semantics of APIs and other design decisions early in the project.

6.2 API Design

During problem analysis and architectural design students analyze the responsibilities of SPA subsystems and components. They understand how SPA Front-End and Query Processor communicate via design abstractions stored in the PKB. For example, SPA Front-End needs API operations to create design abstractions such as abstract syntax trees (AST), symbol tables, procedure call graph or control flow graph (CFG). On the other hand, to evaluate queries such as Q1 – Q6, Query Evaluator needs operations to traverse AST and CFG, and extract information from PKB required for query evaluation.

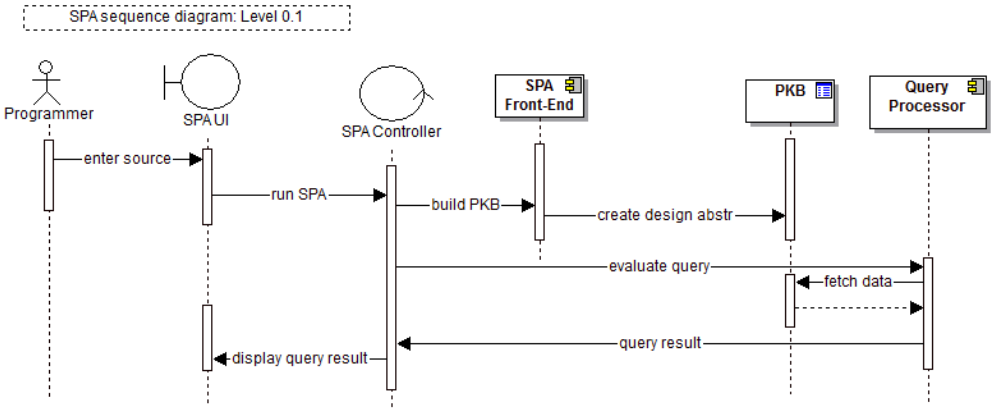

Figure 3. Top-level sequence diagram for SPA subsystems

SPA subsystems and components have different requirements regarding PKB APIs. It is a non-trivial task for students to discover what APIs are needed, and then to specify them. We advise students to work on API discovery incrementally, considering each interface between SPA component (e.g., Parser) and specific design abstraction in PKB (e.g., AST) separately. Students produce small samples of APIs, get feedback and then proceed with the rest, integrating partial APIs into a complete set of PKB API for the project.

Interactions between SPA components and design abstractions can be shown as UML sequence diagrams. Students explode a diagram of Figure 3 into diagrams showing interactions between specific SPA components and design abstractions. For example, Parser and AST,

Parser and symbol tables, Query Evaluator and AST, Query Evaluator and CFG, etc. Such diagrams ensure that students consider requirements for design abstraction API in respect to all its clients. Diagrams are useful in planning stages of API discovery, and in monitoring the progress of API design work.

Students discover API for each interaction between an SPA component and a design abstraction working in pairs. For example, one student simulates Parser actions for a source program fragment, while the other student suggests suitable API operations for AST. The dialog takes form of discussion and negotiations. We conduct such API discovery dialogues during tutorials and give students an assignment to continue the exercise. Communication among team members, interfacing of system components, integration testing, documentation, and project planning is based on those APIs. Initial API operations students come up with are specific to program situations that students analyze. In follow up analysis, students generalize and simplify operations.

During the API discovery exercise, many partial APIs are developed independently by student pairs. Therefore, we ask students to look through their documentation and unify API documentation. Each team must agree on API documentation standards in terms of solutions and notational conventions. To kick off API discovery and specification, we feed students with examples of quality API specifications. Students submit samples of their API documentation and based on feedback work out full blown PKB APIs.

To see the interplay between API design and use, students switch their roles between API designers and users. In that way students start appreciating the importance of writing documentation that is understandable for others. Throughout the course, communication among team members, interfacing of system components, integration testing, documentation, and project planning is based on APIs.

6.3 Justifying Design Decisions

Any non-trivial design problem has many solutions. Experienced designers do not jump to the first design solution that seems to make sense. Not many students entering our course have a habit to consider design alternatives. The need to consider design decisions is hardly justified in small assignments. Project courses create a better opportunity to train this important skill. We show students how to consider design alternatives and evaluate trade-offs among them in the view of the required qualities of a software system, and taking into account the impact a given design decision may have on other design decisions.

Criteria for evaluating design decisions in SPA project are simplicity, reliability, reusability and performance of the Query Processor. Students apply the following steps in a variety of design contexts:

1) Describe the design problem under consideration
2) Identify, write down and prioritize goals to be met by relevant design solutions
3) Consider alternative design solutions to the design problem under consideration
4) Evaluate each alternative design solution:
   a) Analyze how well the design solution satisfies identified goals; its strengths and weaknesses
   b) Analyze any other implications of a given design solution
5) Analyze trade-offs among design alternatives to justify your choice of the design solution.

7. Evaluation of Students' Projects

Students write a final report in which they document project plans, development process, architectural and detailed design decisions. Each team is given one hour to present their work. We grade students based on the quality of their design, the ability to evaluate design decisions

and argue for their design choices in the view of the stated quality attributes - reusability, extensibility and the efficiency of a query evaluation strategy.

We also assess the scope of the implementation, conformance to requirements, and the reliability of code. For that we test student programs with a regression testing tool *AutoTester* that runs 500 test cases. *AutoTester* captures information about failed test cases, exceptions, crushes and timeouts (when the time taken to evaluate a query does not meet the time limit we set). Students are shown test results and can give explanations of what went wrong if they know the reason. In case of minor errors, we give students opportunity to fix errors in limited time and then rerun their programs.

We run clone detectors [2][11] to spot possible cases of code plagiarism. In 10 years, we have found cases of documentation plagiarism, but have not discovered cases of code plagiarism. We believe code plagiarism in the course is difficult for two reasons: First, as supervisors work closely with students, using somebody else's unfamiliar code would be inevitably spotted during consultation session. Second, it is quite difficult to plagiarize only some part of SPA designed by other team. Because of many dependencies among SPA components, the integration effort would be too high to make plagiarism worthwhile.

8. Tools for the Project

Students use Visual Studio IDE with a standard C++ compiler. They use CppUnit [16] for C++ for unit testing, and our *AutoTester* for integration and system testing. In earlier offerings of the course, students used Concurrent Version System (CVS) [17], which was replaced by Subversion (http://subversion.apache.org) and Mercurial (http://mercurial.selenic.com). Students use Doxygen [18] and public domain UML modeling tools of their choice.

9. Course Experiences and Evaluation

Despite heavy load, students like the course. As supervisors working closely with students throughout the year, we observe huge difference in student's ability to tackle the design issues in the overall context of requirement analysis and implementation at the beginning and at the end of the course. In student's surveys we read many positive comments about the educational values students draw from the course. The industry perception of our students' development skills improved a lot over years, however we implemented many other changes in our curriculum as well as introduced yet other types of project courses, so it is not possible assess the specific impact of any single innovation in our curriculum in an objective way.

Teaching by example with frequent feedback is an effective way to get students started with design-in-the-large. Examples given to students may include sketches of software architecture, API specifications, and also illustrations of how to apply design techniques. Some students follow exactly examples to create design solutions for the whole project, while others learn from examples, but then innovate, experiment with ideas and propose their own versions of design techniques. In both cases it is most important that students clearly understand the rationale and motivation for design techniques. While lectures cover the theory of design principles, it is face-to-face discussions between teams and instructors that spark understanding. Especially at the beginning, students err often and need lots of feedback and constructive discussions to get on the track.

During architecture design, students describe APIs of major software components at abstract level. Initially, students experience a great deal of problems to move around in the space of abstract concepts. Despite background in identifying class interfaces and in ADT theory, students find it difficult to perceive, analyze and describe design problems and solutions at abstract level in the project situation. Students fail to see a benefit of implementation-independent system descriptions. They would like to jump to implementation, and talk in terms of specific data structures and their implementation. Relevance of what they learned about

class interfaces and information hiding to the design-in-the-large situation is not clear to them. Tutorials in small groups are effective in helping students get comfortable with capturing ideas in abstract yet precise way.

Once students know how to express problems abstractly, we show them why it is important to consider, analyze and evaluate alternative design decisions. Most students do not have a habit to do so when they start our course. They tend to assume that the first solution that occurs to them is the only right one. We found that this attitude can be changed if students see specific examples of situations when simple-minded (or overly complex) design decisions conflict with qualities their software should satisfy. Another eye opener and motivator is when students see that one design decision often must be considered in the context of other mutually dependent design decisions. In SPA, the performance of query evaluation and maintainability of the SPA offer many examples of situations where not considering alternative design decisions may lead to problems.

In one-term project course CS3215, we had two assignments to provide teams with feedback on their API design. We knew that only some of the students in a team would really get the concept, but the frame of a one-term course did not allow us to assess students' proficiency with API design individually.

In two-term project course formula CS3201/CS3202 we can give more attention to the process of API discovery. Tutorials allow us to initiate students in applying techniques described above. We incorporated elements of individual assessment to ensure that all the students are proficient with design techniques to be applied in the project and can contribute to team work during development.

10. How to Use our Courseware

SPA as a model problem for the course can be easily scaled by changing the source language, design abstraction definitions, and PQL rules. Design methods described in this paper remain valid and relevant to any other than SPA programming problem that offers rich APIs. Our courseware includes 100-page Handbook [8], lecture notes, assignments, and tutorial questions. Materials in source format can be requested from the author.

11. Related Work

The course is based on fundamental concepts of modularization and module interfaces laid by Parnas [14] and Liskov's [12]. Parnas introduced modularization criteria and the principle of information hiding, while Liskov unified the concept of modules with data types and proposed module interface descriptions in the style of abstract data types. Among textbooks, Sommerville [15] and Ghezzi et al. [4] provide comprehensive treatment of design fundamentals and techniques our course is based upon.

In [6], we described course motivation, structure, and initial experiences. In this paper, we described the teaching methodology, and experiences after changing form on-term, intensive course, to two-term course with regular workload in each of the two terms.

12. Conclusions

We described a model problem, teaching methodology and 10 years of experiences teaching advanced design team-based project course. The course aims at teaching the theory and practice of design fundamentals, exposing students to problems of the design in-the-large. We emphasize the design of application program interfaces (API) and justifying design decisions. In our experience, design is best taught by example, giving students frequent feedback and opportunity to discuss design issues in face-to-face tutorial and consultation sessions.

We believe students should take as many project courses of different types as it is possible to get ready for work in an industrial environment. Each type of a project course offers its own unique values. Ideally, students would take a project course focused on application of design principles first. This course should limit factors that make application of principles particularly difficult (such as fuzzy requirements or constraints imposed by integration with existing programs). Equipped with sound knowledge of principled design, students could better face other challenges of real-world software development addressed in follow up project courses.

Search on the WWW reveals a number of universities that teach software design courses. Information available online shows that most such courses include project work. We trust that those who teach or plan to teach design courses may find our experiences reported in this paper useful.

Acknowledgments

Eng Pin Kwang helped establish tool infrastructure for the course during initial years. Markus Kirchberg, Khoo Siau Cheng and Damith Chatura Rajapakse taught the course in alternative terms and contributed many ideas and improvements. Damith came up with innovative versions of the course that included release of software developed by students to the public domain. Kee Yong Ngee implemented advanced features into the *AutoTester* making it much more useful for the students and for instructors. Xue Yinxing, Ashwin Nanjappa, Cristina Carbunaru and many other teaching assistants have been involved in the course and the author thanks all of them for contributions. Thanks are due to Katsuro Inoue and Toshihiro Kamiya for letting us use CCFinder for plagiarism detection.

References

[1] Armarego, J. Advanced Software Design: a Case Study in Problem-based Learning," Proc. 15th Conf. on Software Engineering Education and Training, CSEET'02, Covington, USA, Feb. 2002,

[2] Basit, H. A., Jarzabek, S. "Data Mining Approach for Detecting Higher-level Clones in Software", *IEEE Trans. on Soft. Eng*., July/August 2009 (vol. 35 no. 4) pp. 497-514; Published online January 2009

[3] Brooks, F. *The Mythical Man-Month: Essays on Software Engineering*, McGraw Hill, Addison Wesley

[4] Ghezzi, C., Jazayeri, M. and Mandrioli, D. *Fundamentals of Software Engineering*, Prentice Hall, 2002

[5] Jacobson, I., Booch, G. and Rumbaugh, J. The Unified Software Development Process, Addison-Wesley, 1999

[6] Jarzabek, S. and Eng, P.K. "Teaching an Advanced Design, Team-oriented Software Project Course", *18th Int. Conference on Software Engineering Education and Training (CSEE&T),* IEEE CS, April 2005, Ottawa, pp. 223-230

[7] Jarzabek, S. (editor), "Teaching Software Project Courses" (special issue, 13 papers), Forum for Advancing Software Engineering Education, Vol 11, No 6, http://www.fase.sunderland.ac.uk/, June 2001

[8] Jarzabek, S. *Software Engineering Project*, Pearson Education Asia Pte Ltd, 2012

[9] Jarzabek, S. and G. Wang "Model-based Design of Reverse Engineering Tools", *Journal of Software Maintenance: Research and Practice*, No. 10, 1998, John Wiley & Sons, pp. 353-380

[10] Jarzabek, S. "Design of Flexible Static Program Analyzers with PQL," *IEEE Transactions on Software Engineering*, March 1998, pp. 197 215

[11] Kamiya, T., Kusumoto, S., and Inoue, K. "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code", *IEEE Trans. Software Engineering*, 2002, 28(7): pp. 654-670

[12] Liskov B., and Guttag, J. *Abstraction and specification in program development*, MIT Press, 1986

[13] Robillard, P. "Teaching Software Engineering through a Project-Oriented Course," Proc. Conf. on Software Engineering Education, CSEE'96, 1996, pp. 85-94

[14] Parnas, D. "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972 pp. 1053 – 1058

[15] Sommerville, I *Software Engineering,* (8th edition) Addison Wesley

[16] CppUnit. http://sourceforge.net/projects/cppunit.

[17] Concurrent Version System. http://www.cvshome.org.

[18] Doxygen. http://www.stack.nl/~dimitri/doxygen.