System requirements analysis with ICONIX process case study: group project peer-assessment tool

Qingxiong Ma
University of Central Missouri

ABSTRACT

To enhance the process of teaching team skills for software engineering students, it is important to have an effective self- and peer-assessment tool for team-based projects. This tool not only provides prompt feedback and help students achieve learning and performance goal, but also helps faculty in grading students' team-based projects. In attempt to develop such an online assessment tool, students in software engineering class are expected to do an on-going project based on the documents provided. Instead of using UML modeling technique in the system requirements analysis, ICONIX process is adopted. ICONIX Process is UML Use Case driven but more lightweight. It uses only four UML based diagrams in a four-step process that turns use case text into working code. The milestones in the four-step process are: requirements review, preliminary design review, detailed design review and deployment. The focus of this study is on the system requirements analysis part by using techniques covered in the first of the four steps in ICONIX Process. Specifically, from this analysis use cases are identified, a domain model is produced and some prototype GUIs are made. Teaching notes, assignments, evaluation, and analysis are provided.

Keywords: Group project, self- and peer-assessment, Software engineering, ICONIX process, Requirements analysis, use-case-drive

INTRODUCTION

To teach team skills and competences, the faculty in computer technology programs frequently requires students to do some team-based projects [3, 6] and very often guided self- and peer-assessment is used in the team-based projects to enhance the teaching effect [2, 4]. However, due to lacking of a powerful, but easy to use assessment tool, the effectiveness of this strategy is significantly affected. In practice, the assessment was conducted manually by an individual faculty with paper-based instrument. The quantity of evaluation data set is tremendous, and the calculation is complicated. It is tedious and time-consuming for faculty to do analysis. As a result, it is difficult for students to get a prompt feedback to improve their team performance. If multiple assessments are conducted (which is preferable), it will be very difficult and challenging for faculty to manage. Thus, these issues make this strategy less practical to implement in class. Therefore, an online assessment system that can automate the data collection and analysis process will be highly expected.

In attempt to develop such an online assessment tool, students in software development class are expected to do an on-going project. This study focuses on the system requirements analysis by using techniques covered in the first step in ICONIX Process. Specifically, students are expected to achieve following learning objectives:

1. Learn the guidelines of writing functional requirements and apply the knowledge in a specific business case.
2. Learn how to create a domain model from the case description.
3. Learn how to describe the system usage via use cases, which are identified based on functional requirements, domain objects, and GUI prototype.
4. Learn UML modeling language conventions and a create use case diagram.

ICONNIX Process

According to Rosenberg and Stephens [8], UML is “way too big” to use in software design because people can do 80% of their modeling with 20% of the UML. Instead, they proposed a use case driven but more lightweight approach -- ICONIX Process. This approach claims to be a minimalist (core subset of UML), streamlined. It focuses on area that lies in between use cases and code. The ICONIX Process uses only four UML based diagrams in a four-step process that turns use case text into working code. The general idea of ICONIX Process is presented in a diagram (Fig. 1 in Appendix). It is composed of dynamic and static workflows, which are highly iterative: one might go through one iteration of the whole process for a small batch of use cases, all the way to source code and unit tests. The milestones in the four-step process are: requirements review, preliminary design review, detailed design review and deployment. The ICONIX Process is well suited to agile projects, where swift feedback is needed on such factors as the requirements, the design, and estimates.

Requirements definition in the ICONIX process suggests three types of requirements: functional requirements, domain modeling, and behavioral requirements. Functional requirements define what the system should be capable of doing; Domain modeling makes sure understanding the problem space in unambiguous terms; Behavioral requirements define how the user and the system will interact. It is recommended that starting identify all the use cases with a GUI prototype when exploring the requirements. The deliverables from requirements analysis includes, but not limited to, use cases, a domain model, and some prototype GUIs. The

milestone in the requirements analysis is "Requirements Review", which assures that the use case text matches the customer's expectations.

TEACHING METHOD

Course Description and Student Background

The course "Software Engineering" is required for Master of Information Technology program at a local university. Students taking this class must be admitted to the program of Master of Information Technology. The class size is about 20 to 25 students. Majority of students are international students from Asian countries. Only a small portion of them have background of computer related degree or working experience. Many of them have background of other fields such as mechanical, electric, electronic, or chemical engineering. System analysis & design is a pre-requisite course, but only about a quarter of students have taken it before.

The course emphases both theoretical knowledge and hands-on skills. The theoretical part discusses a collection of methods which embody a systematic approach to the software development process, from requirements elicitation and analysis, through specification and design, to implementation, integration, testing, maintenance, and retirement of software. A variety of concepts, principles, techniques, and tools are presented, encompassing topics such as software processes, project management, people management, software requirements, system models, architectural and detailed design, user interface design, programming practices, verification and validation, and software evolution [7].

The hands-on part is used to enhance the theoretical learning by applying the knowledge learned in real life project – analyzing business problem and providing solutions. All students in the class are required to do a project. As part of the project, students are expected to develop a complete set of requirements and specifications for a peer-assessment system to enhance team skill learning process in their group project.

Teaching Materials

Blackboard course management system is used to manage all teaching materials, assignments, students' work submissions, and grades. In the class, students are provided with problem case description, paper-based instrument for self-peer-assessment, and the rubrics[1]. The rubrics used in the instrument was developed based on previous research [2, 4, 9] and department faculty brain-storming. Students are encouraged to clarify and ask questions about requirements from other faculty and students in the department who had experience of group projects.

The instruction of system requirements analysis is lecture based. Three types of system requirements technique are discussed: functional requirements, domain modeling, and use-case based behavioral requirements. In the class, each method is covered and students are expected to understand each technique and guidelines as well as notations of use case diagram. During the instruction, step-by-step examples are demonstrated. The lecture notes for each technique are provided below. The guidelines for object domain modeling, use case modeling, and requirements review are largely based on the ICONIX process [8]. Only some changes were made based on the author's instructional experience and students' feedback.

[1] The case description, instrument, and rubrics are available upon request.

Functional Requirements

A functional requirement is related directly to a process a system has to perform. Generally requirements focus on what a system should do, rather than how it should do it. A requirements definition must satisfy several needs. Following are a few guidelines for students to keep in mind as they document systems requirements [1, 5].

1. Define the requirements from the developer's perspective. Make sure they are complete and understandable by developers.
2. Use the active voice and keep statements short. Avoid long narrative paragraphs.
3. Write requirements at a consistent level of detail throughout the document. Regarding to the level of granularity, write each requirement in a way that can be tested.
4. Avoid multiple requirements aggregated into a single statement. Conjunctions like "and" and "or" in a requirement suggest that several requirements have been combined.
5. Make the requirement concisely and avoid redundancy. The multiple instances of the requirement may cause modification difficulties and inconsistency issues.
6. Requirements should be complete or no necessary information should be missing, but this completeness must be within a limit or scope and cannot be over specified.
7. Requirement must be modifiable and traceable. Requirements are uniquely labeled and structurally listed.

Domain Modeling

In ICONIX Process, a domain model for a project defines the scope and forms the foundation on which to build the use cases. It shows graphically how all these different terms relate to each other. The domain model uses generalization (is-a) and aggregation (has-a) relationships to show how the objects relate to each other. The domain model also provides a common vocabulary to enable clear communication among members of a project team. Some practical steps are listed below:

1. Go through the high-level requirements identified in functional requirements and highlight nouns and noun phrases. Extract them and list them all to have a primary list.
2. Identify and eliminate duplicate terms and unnecessary items in the list. For instance, some example synonyms are: feedback and comments, group and team etc. Some other terms may be simply fall outside the scope of the project such as department, course, and project.
3. Create a domain model only with objects having aggregation (aka has-a) relationship. Notice that a few of the domain objects don't match up with anything else. Put these together over on the right tentatively.
4. Identify and add implicit domain objects that weren't in the requirements, but instead they are derived from personal understanding of the problem domain.
5. Add generalization relationships in the domain model. A generalization relationship is one in which one class is a "kind of" some other class; generalization is often called is-a relationship.

Use Case Modeling

In ICONIX process the domain object model forms the foundation of the static part of flow, while the use cases are the foundation of the dynamic part (Fig. 1 in Appendix). The static part describes structure; the dynamic part describes behavior.

Use cases describe the way the user (Actor) will interact with the system and how the system will respond. The ICONIX Process suggests to explore the requirements by starting with GUI prototypes or screen mock-ups. Behavior requirements detail the user's actions and the system's responses to those actions. For majority of systems, this interaction between user and system takes place via screens, windows, or pages. Often it's easier for the customers and end users to relate to a visual aid. These visual aids can be any form of GUI prototypes and screen mock-ups. They present a sequence of screens as they will appear to the users within the context of the usage scenarios being modeled. It's also important that the screen mockups include details about the various buttons, menus, and other action-oriented parts of the UI. Some important use case modeling guidelines are listed below.

1. Write use case by using the functional requirements as the primary source
2. Write use cases in the terms from the domain object model.
3. Write use cases in a noun-verb-noun sentence structure with active voice.
4. Make sure that use case text is not too abstract. Expand use case by describing event/response flow in sequence from both aspects (the user and system).
5. Write use cases by using GUI prototypes and screen mock-ups, and make sure that a use case accurately reflects the GUI.
6. Write use cases in two-paragraph style. First paragraph describes the main successful scenario; the second paragraph describe optional flows and exceptions.
7. Refer boundary classes (e.g., screens) by name.
8. Remove everything that is out of scope.
9. Organize use cases and actors in use case diagrams. The Use Case diagram gives the overall picture of the systems behavior.

Requirements Review

After all requirements identified, a formal review of them should be conducted with project clients and other interested parties. The focus of this review is to verify all information and requirements make sense. All relevant parties understand and agree with the requirements identified. This session will ensures that the system as described genuinely matches up with the requirements. Below are the requirements review guidelines.

1. Make sure the domain model describes the most important abstractions from the problem domain (i.e., real-world objects), in nontechnical language that the project users can understand.
2. Shows the is-a (generalization) and has-a (aggregation) relationships between the domain objects.
3. Make sure use cases describe both basic and alternate courses of action, in active voice.
4. Allocating functional requirements to use case. Do not include passive voice.
5. Write use cases based on both the object model and the user interface. Name the screens that will participate in the use case, and use them in the use case text
6. Supplement use case descriptions with storyboards, screen mock-ups, or GUI prototypes.

7. Review the use cases, domain model, and screen mock-ups/GUI prototypes with end users, stakeholders, and marketing folks, in addition to more technical members.
8. Trace each requirement to its use case.

ASSIGNMENT

After each lecture on the three types of requirements definition, an assignment is posted in Blackboard. Totally there are three assignments in the requirement definition.

Assignment 1 asks students to read the provided documents carefully and try to fully understand the problem. The documents include case description, instruments used, and rubrics. Then students are asked to define the scope of the project and list all functional requirements.

Assignment 2 requires students to identify domain objects and create a domain model by following the 5-step procedure depicted in the lecture note. Students are also required to create a glossary or dictionary of the domain objects and give definition for each of them. In this way, the problem is understood unambiguous and terms are used consistently.

Assignment 3 asks students to define the way that the user and the system will interact. It is recommended to define this interaction starting with a GUI prototype. Specifically, students are expected to: a) identify a preliminary list of use cases, and create GUI prototypes or screen mock-ups for each use case; b) re-write use case in two-paragraph style based on functional requirements from assignment 1, domain model from assignment 2, and GUI prototypes; c) create a use case diagram.

Although no specific assignment is given for requirements review, students are encouraged to review their earlier submissions by following the review process and cross-checking the requirements. After modifications, students are allowed to resubmit to improve their grade performance.

In addition, the proposed solution for each assignment is available in Blackboard after the due date of the first submission. These solutions[2] will give students more specific expectations and references or examples on how to follow the principles and guidelines.

EVALUATION

Since the requirements identification process is iterative and incremental, multiple submissions for each assignment are expected from students. To enhance their learning process, it is reasonable and desirable that the evaluation of students' work should also be iterative. The assignment grading process in this study is a sample based, multi-staged assessment cycle. The detailed process is presented in Fig. 2 in Appendix.

The first grading is looking for whether students follow the approaches, guideline, or notations covered in the lecture note. Thus, the assessing rubric may be composed of approaches (process or steps learned in the class), required notations/symbols, and the completeness of major components expected. For example, when grading the use case diagram, a 3-criteria with -4 level matrix was used (Table 1 in Appendix). The benefits of using rubrics include consistent, fair, and efficient in grading, help understanding and reducing arguments from students [10]. To make it

[2] The solutions are not attached in the Appendix because of length, but they may be available upon request.

operational with multiple grading for each assignment, the first grading is sample-based. Only a few submissions were explained and commented in the class.

The second grading is to check if students can have an accurate understanding of the problems and whether the requirements identified are complete. Whether it is necessary for students to have further submission is dependent on how well they complete the assignments. This process can continue till very few students make mistakes.

The last grading is not performed until the end of semester after they submit the whole project.

DISCUSSION OF LESSONS LEARNED

Even though some demonstration and examples are presented in class showing how to following guidelines and procedures, students still make some mistakes in their practices. Following are the typical mistakes students made and discussions for each assignment.

Lessons from Assignment 1

1. Project scope is creeped. The project is only limited to team members' performance evaluation. However, some students creeped the scope by including project management function such as elect team-leader and project planning in the requirements.
2. The statement is too long and not clear. For instance, a student stated one of the functional requirement as "the same course is handled by the same faculty or a different faculty and the student will enroll in the same course work of the same faculty or will enroll for some other faculty for a different course work".
3. Functional requirements are redundant or partially overlapped. For example, "students can make comments" is part of assessment. Some students listed it as a separate functional requirement.

To avoid such mistakes, it is suggested that students to read the documents provided one more time to fully understand the problem and the scope of the project. Students should read the whole case, including the rubrics. Students should pay close attention to every word and understand the deep meaning of every term.

Lessons from Assignment 2

1. Students should pay attention to the synonyms or similar expressions such as evaluation, assessment, feedback, comment, self- and peer-assessment.
2. Some students showed wrong aggregation and generalization relationships in the domain model.
3. Some students did not following the 5-step procedure to create domain model.
4. Some students are confused between domain model and UML class diagram.

It is suggested that students should be guided in reading the documents and help them to correctly understand the term semantics in the specific context. It is important to require students to show the development process when creating the domain model, not just the result. It will be helpful to give more examples of aggregation and generalization relationships. It is also important to clarify the difference between domain model and UML class diagram, especially

design class diagram. There should be some resemblance between them, but do not expect the final class diagrams to precisely match the domain model.

Lessons from Assignment 3

1. Some students with UML background used detailed use case template. They focused on format instead of the content of the interaction and event flow.
2. Some students listed uses cases at different abstract level together. For example, "Admin can manage classes"; and "Admin can update a class".
3. Some students write use cases not in the expected style and format. The use case descriptions are not in 'Subject-Verb-Object' format or in active voice.
4. Some students did not use the appropriate notations in their use case diagrams. They used nouns to label use cases; they had arrow lines between actor and use cases; or they put all use cases (16) in one diagram.
5. Some students have inconsistency between use case event flows and screen mock-ups.
6. Some students continue to have project scope problem. They should focus on assessment only, not group project management. Some students mistakenly include functional requirements such as "Faculty assigns projects to enrolled students; Faculty can add project to the class they teach; and Faculty can view project report".

Ask students to review the lecture note on use case. Give more examples on two-paragraph use case description and use case diagrams. To have an accurate and complete use case description, it is important to encourage students to read the documents again and cross-check the requirements (requirements review).

CONCLUSION

This study is attempt to teach software engineering students the requirement identification skills by using ICONIX process, which claims using the minimum of UML diagrams. It is easy to understand and easy to follow. As part of on-going project, this study only discuss the requirement analysis, which is the first of the four phases in ICONIX process. Future study will continue to follow the ICONIX Process to discuss robustness analysis, system design, and development.

REFERENCES

[1] Dick, Jeremy, Elizabeth Hull, and Ken Jackson. Requirements engineering. Springer, 2017.
[2] Haas A.L., Haas R.W., Wotruba T.R. (1998), 'The use of self-ratings and peer ratings to evaluate performances of student group members', Journal of Marketing Education, Vol 20, No 3, pp 200–209.
[3] Jun, Huang. "Improving undergraduates' teamwork skills by adapting project-based learning methodology." In Computer Science and Education (ICCSE), 2010 5th International Conference on, pp. 652-655. IEEE, 2010.
[4] Lingard, Robert W. "Teaching and assessing teamwork skills in engineering and computer science." Journal of Systemics, Cybernetics and Informatics 18, no. 1 (2010): 34-37.
[5] Ma, Qingxiong and Jiang, Yueyang, Process-oriented information system requirements engineering - a case study, Journal of Business Cases and Applications, 10 (2014), pp. 1-16
[6] Marques, Maira, and Sergio F. Ochoa. "Improving teamwork in students software projects." In *Software Engineering Education and Training (CSEE&T), 2014 IEEE 27th Conference on*, pp. 99-108. IEEE, 2014.
[7] Roger S. Pressman & Bruce R. Maxim, *Software Engineering: A Practitioner's Approach* (8th edition), McGraw-Hill, 2015.
[8] Rosenberg, D, Stephens, M. Use Case Driven Object Modeling with UML: Theory and Practice. Berkley: Apress. ISBN-10: 1590597745; 2007.
[9] van den Berg I., Admiraal W., Pilot A. (2006), 'Designing student peer assessment in higher education: Analysis of written and oral peer feedback', Teaching in Higher Education, Vol 11, No 2, pp 135–148
[10] Walvoord, Barbara E., and Virginia Johnson Anderson. Effective grading: A tool for learning and assessment in college. John Wiley & Sons, 2011.

APPENDIX (Graphs and Tables used in the paper)

Table 1 Rubric for Assessment business process modeling with Domain Model

Level:	0	1	2	3
Follow the Modeling guidelines	Not following the guidelines taught in the class (0%)	Some of guidelines are followed (25 – 50%)	most of guidelines are used (51 – 80%)	Almost all guidelines are followed (above 80%)
Use appropriate notations	Not using the required notation (0 – 25%)	Some of required notations are used. (25-50%)	Majority of notations are correct. (51-80%)	Almost all symbols and flow flows are correct. (above 80%)
Identify the relevant components	No related components are identified (0%)	some relevant components are identified (25 – 50%)	most relevant components are identified (51 – 80%)	Almost all items and components are identified (above 80%)

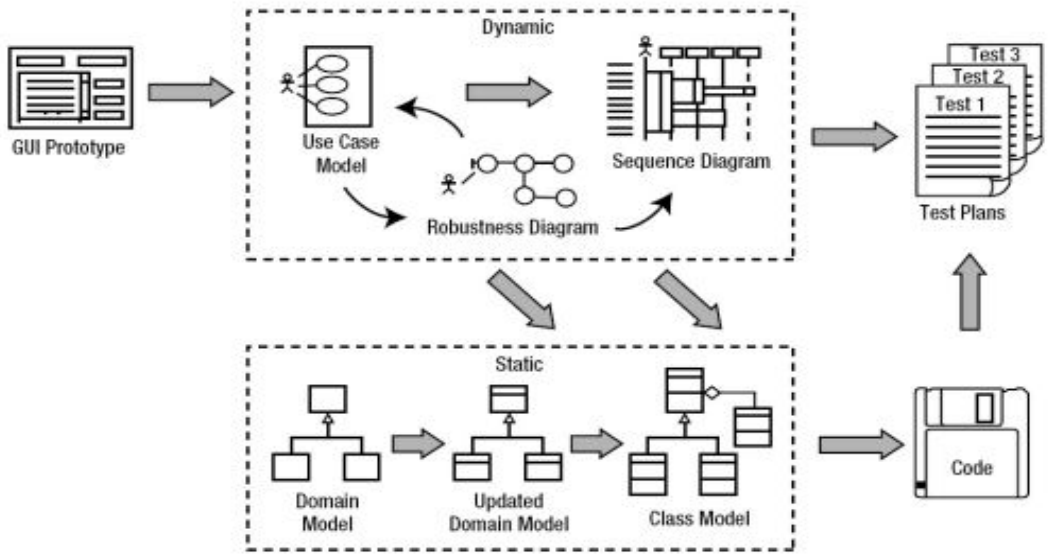

Fig. 1. ICONIX Process

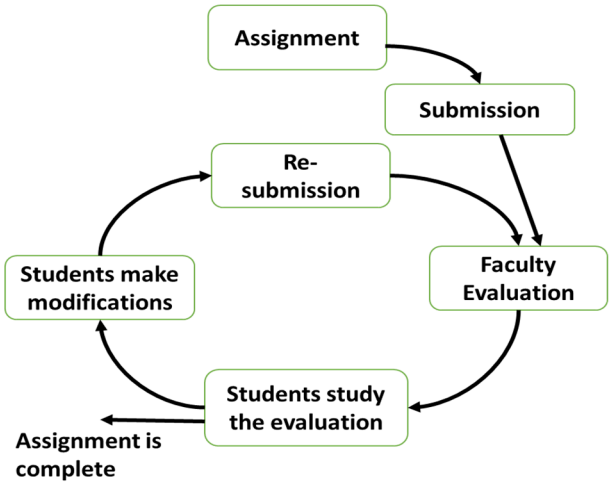

Fig. 2 Assignment Evaluation Cycle