SIMULATING REQUIREMENTS GATHERING

Martin L. Barrett
Department of Computer and Information Sciences
East Tennessee State University
P.O. Box 70711
Johnson City, TN 37614
barrettm@etsu.etsu-tn.edu

1. Abstract

One of the difficulties in teaching a project-based Software Engineering course for undergraduates is giving students experience with requirements gathering before they begin the course project. This paper describes a simulation of Joint Application Design (JAD), a technique used in industry to gather requirements from users. The simulation is a role-playing exercise in which students play the parts of both developers and customers involved in specifying a new software product. Each participant is given a script of behaviors to act out for his or her role and a set of specific requirements for the product. The participants must work out conflicts and ambiguities built into the simulation to produce a consistent product specification.

2. Introduction

A goal of an undergraduate software engineering course is to have students experience the full range of activities in software development. Project-based software engineering courses accomplish this by requiring students, usually working in groups, to specify, design, code, and test a medium-sized software project. Difficulty can arise for students in knowing how to apply newly-learned methods to the problem without having practiced those methods. This paper discusses a method for practicing one part of the development process, requirements gathering.

There are several models of software development. The iterated waterfall model breaks development into several discrete phases: requirements specification, design (both high and low level), implementation (coding), testing, delivery, and maintenance. Other models include the spiral model and the evolutionary prototyping model [3]. Regardless of which model is used as a basis for development, the customer and end users must be queried for the requirements of the system by the developers. Gathering those requirements is a non-scientific task, yet one that is crucial to the success of the software product [3].

SIGCSE '97 CA,USA

Teaching about the development phases typically involves presenting one or more methods via lectures and worked examples. In addition, students learn to use the methods both from readings and by working out small examples (necessarily small due to time constraints). This knowledge must then be used on the class project. For example, an object-oriented design method such as Object Modeling Technique [5] may be introduced, case studies examined, then practiced by designing solutions to short problems based on given requirements. In this way, students will have some design experience before tackling the course project.

Requirements specification can be more troublesome than the other phases for student projects. One reason for this is the relative lack of specification experience in other courses, due to the fact that academic projects are often fully specified. Also, gathering the requirements has more to do with interpersonal relations than it does with computer science. For these reasons, a requirements gathering simulation exercise was developed and tested to meet the need for realistic practice in this area. Its main feature is role playing, in which user and developer behaviors are scripted to provide lively interaction among participants in the course of specifying a hypothetical application. One particular method, Joint Application Design (JAD), is used for the simulation.

The rest of this paper is organized as follows. First, the requirements phase is discussed briefly. Next, an outline

of the JAD method is given. The simulation exercise is then described. Experience with the simulation is discussed next. Finally, parts of a sample script and the requirements list are given in the appendix.

3. Requirements Subphases

The requirements phase can be broken up into four subphases: requirements gathering, requirements analysis, requirements specification, and requirements verification. In other words, ask what needs to be done, make sense of it, write it down, then check that it was accomplished.

The latter three subphases are amenable to student practice. Analysis, using either a procedural method such as data flow analysis or an object-oriented method such as Shlaer-Mellor [6],can be easily practiced from a given set of requirements, and many case studies exist. Similarly, specification of requirements in technical English, while something of an art, can be practiced and existing requirements can be critiqued. Formal specification methods such as Z may require more time and practice. Requirements verification is a necessary end-of-project activity associated with acceptance testing and is a fairly straightforward process [3].

There are several techniques for the requirements gathering subphase. These include interviews, questionnaires, prototyping, market research, and Joint Application Design (JAD). Lectures and case studies can discuss these techniques but are inadequate in illustrating the nuances. The inherent fuzziness of the process makes it difficult to practice before it is needed in a project. Using real customers and users from which to elicit requirements for practice problems is unfeasible, due to time constraints. A JAD session for gathering requirements is particularly difficult to practice because of its elaborate nature.

Another set of role-playing exercises is described by Raghavan et. al. [4]. Several other techniques are used there in addition to JAD. The JAD exercise appears to be similar to the one described here, but the requirements set-up is much less extensive. Notes on the various methods are also included.

4. JAD Fundamentals

JAD uses structured group discussion among customers, users and developers, followed by consensus decision making, to develop software requirements. By getting both sides together for an extended session, JAD tries to shorten the gathering subphase and ensure that the requirements are correct, unambiguous, and consistent. For a detailed look at the JAD method, see [7].

Before a JAD session, the session chair (called the facilitator) gathers information about the proposed system. The project scope is defined and its purpose and objectives are set. The proposed work flow, data, and performance criteria are investigated, possibly by looking at an existing system. The facilitator also works out session logistics, such as group formation, scheduling, preliminary reports, and a session outline. An important part of this work is gaining high-level executive support for the entire process.

The session itself begins with a project overview by the facilitator, including a list of all assumptions currently held about the system. The facilitator leads a group discussion of all unresolved issues, attempting to forge an agreement among the participants on the system requirements. These issues include the system work flow, determination of all data items, input sources and output destinations, the user interface, and in general all functionality of the system.

The facilitator's goal as session leader is to lead the group to a consensus on all open issues. After introducing each topic, the facilitator must ensure participation from both groups. Discussion may need to be cut short in the face of unresolvable topics, which are added to the list of open issues; however, if the corporate sponsor is present, as many of these issues should be decided as possible. The facilitator then summarizes each topic before moving to the next one.

5. The Simulation Exercise

The JAD simulation exercise uses a hypothetical CAD system as its project. This system is to be used for the design, production, inventory, and sale of widgets, and it is a replacement for several stand-alone systems. The instructor has a complete set of requirements for the system; see the appendix for a partial listing.

The exercise consists of fourteen scripts, one each for the chair, a recorder, six users, and six customers. The instructor plays the role of executive sponsor. Each script contains a job description, a set of personal behaviors, and a list of system needs. It is the responsibility of the student assigned to a particular role to make sure that these needs are expressed during the session. The appendix contains the script for one of these roles.

Before the simulation, class time is spent discussing requirements gathering in general, then JAD basics are covered briefly. Scripts are handed out at least one class

session before the simulation, roles are assigned to students, and the simulation is previewed. The facilitator's pre-session tasks are mostly covered in this.

User roles are chief mechanical engineer, assistant engineer, manufacturing engineer, consumer representative, sales representative, and industrial designer. Developer roles are senior system analyst, assistant analyst, database specialist, graphics specialist, network analyst, and user interface designer. The job description for each role includes a brief statement of that role's major job responsibilities. For example, the manufacturing engineer is the liaison between the mechanical engineers and the industrial designer. This engineer handles details of machining and fabrication of widgets, and is responsible for ensuring that the new system automates the process of machine setup for manufacturing widgets based on new design drawings.

Each script describes general personality characteristics to be used in the role-playing simulation. A variety of personalities are represented in the roles. On the developer side, on personality is that of the senior systems analyst, a strong-willed person willing to argue loudly over any point in order to protect the specialists from over-committment.

In addition, the user and customer scripts contain specific system functions (for users) and technical capabilities (for developers). The system functions include specifics about the CAD drawings, input, output, database, and interface functionality of the proposed system. A sample function is that the industrial designer wants the new system to work "just like the MacIntosh interface." The technical capabilities include what each developer's strong and weak points are in design and coding. One instance is that the database specialist is worried about the potential size of the database required for widget drawings, but is willing to promise high data reliability.

The scripts have conflicting, ambiguous, and incomplete requirements. The facilitator must lead the group into making decisions to resolve these problems to produce reasonable requirements for the system. One of the main conflicts is over time. The user group demands delivery of the new system in six months, but the developers want at least a year. The facilitator may suggest a compromise of partial delivery in nine months.

The scribe records the session's main points, either on a flipchart or with a notebook computer. After the session, the facilitator and scribe produce a report outlining the final agreement, including those requirements for which agreement has not been reached. The instructor has a list of all the requirements. This list is cross-referenced against the participants that had responsibility for each requirement and tells the nature of the conflict, if any (see the appendix). The list can be used during the session to monitor participation and afterwards to check the final report.

6. Discussion

The simulation has been used several times in an undergraduate software engineering class with success, both as an educational activity and an enjoyable diversion from normal classroom routine. There is some reluctance on the part of students to act out the roles. However, several strategies can better the chance for a successful simulation.

The first strategy is to put the right students into the key roles: facilitator, chief mechanical engineer, and senior system analyst. The facilitator has the most challenging jobs - to get everyone to participate and to arbitrate disputes. This role requires a certain "stage presence" in order to command attention when needed. The lead user and developer are the bosses of each group. They must advocate for their side while protecting the interests of their people. All three roles work better in the hands of outgoing, demonstrative students. Other students typically will react to these three, especially to provocation.

A second strategy is to have the instructor provide encouragement and hints during the simulation. This includes reminding participants to make their points and helping the facilitator with arbitration.

A third strategy is to give a pep talk before the session. Each student is told to make sure to bring out the key point from his or her script during the session. Not all students like to speak up in class, let alone act out a role. However, this encouragement beforehand can make a big difference. Even reticent students normally are able to make and discuss their key points.

Experience with the simulation has been very positive. Students seem to have genuinely enjoyed the activity, judging by their comments and other reactions. In fact, the time immediately after the session is filled with much discussion and energy. In short, it's fun.

The simulation gives (almost) real experience in using JAD. Pedagogically, the experience of the session could not be easily conveyed by other means, especially lecture. For students, it is also a nice break from regular classroom activities.

The group dynamics experience of the session is also good. Quiet students, as noted earlier, have a chance to contribute to the session. Participants learned to wait for the right moment to add their comments, since the discussions were not always as structured as they should have been.

Does the simulation help students become better requirements gatherers? Not necessarily: several groups of students running JAD sessions for their semester project have had mixed success in getting the requirements completely and correctly. These sessions used faculty as users, and generally had less contention, structure, and overall activity than the simulation did. Part of the problem, however, was due to lack of preparation by the students.

Lack of preparation also applies to groups that have used non-JAD methods for their projects, however. Students felt, though, that the JAD session did accomplish its main goal, to teach them about the difficulties of getting requirements from users.

7. Acknowledgments

Thanks go to the participants of the National Science Foundation Summer Institute [1] held in Indianapolis, 1992; the simulation activity was developed for that institute. Thanks also are due to Bob Riser for using the simulation and giving valuable feedback and encouragement on the activity and this paper.

8. References

[1] Barrett, Martin, "JAD Simulation", in **Bringing the Industry View of Software Engineering to the Classroom**, Report of the 1992 NSF Summer Institute

[2] Davis, Alan, **Software Requirements**, Prentice-Hall, 1993.

[3] Pressman, Roger, **Software Engineering, A Practitioner's Approach**, 3rd Ed., McGraw-Hill, 1992.

[4] Raghavan, Sridhar, Zelesnik, Gregory, and Ford, Gary, "Lecture Notes on Requirements Elicitation", CMU/SEI-94-EM-10, Software Engineering Institute, 1994.

[5] Rumbaugh, J., Blaha M., Premerlani W., Eddy F., and Lorensen W., **Object-Oriented Modeling and Design**, Prentice-Hall, 1991.

[6] Shlaer, Sally, and Mellor, Stephen, **Object-Oriented Systems Analysis**, Yourdon Press, 1988.

[7] Wood, Jane and Silver, Denise, **Joint Application Design**, John Wiley and Sons, 1995.

9. Appendix

Partial Script for Chief Mechanical Engineer

Code in master list: C1

Job Description: You are in charge of widget design. The other mechanical engineer in the group works for you. You are also the main liaison between your group and manufacturing, and between your group and consumer support. You have many years of experience in designing widgets, although in your present job you will not use the design system on an every day basis.

Behaviors: Assertive - you got this job in part because you could make your thoughts known to others and follow through on what you felt needed to be done. You have trouble taking advice from peers and subordinates. Your main goal is to get the new design system up and running as soon as possible to increase the productivity of your group to make yourself look good.

Requirements:

a. You want the new system to run on high resolution monitors, but you do not know or care what that resolution is.

b. One of your goals is to increase productivity. You've heard that object oriented technology can do that, so for this project, you demand objects. You want each graphics piece to be an object, and for pieces to be grouped into subassemblies, and subassemblies into widgets.

c. The objects in the system should be stored in a database of reusable objects.

d. Many designers will need information from the database, possibly concurrently. File sharing must be allowed.

e. Archiving of finished designs will create a widget history for your design team in the future. Insist that an archive be part of the new system.

f. The new system better have a graphical user interface.

g. There are many people using a variety of machines in your department. This system had better run on all of them - you cannot afford to buy new equipment.

h. Reliability is a key requirement. The system has to be up and running all the time for maximum efficiency.

i. Your people will need training on the new system...

j. ... and there should be a gradual transition from the old system to the new one. Your people should get plenty of time to adjust, and the old system should be around just in case.

k. The new system is needed within six months. You're bidding on a government widget contract, and you need the system to guarantee the performance clause in that contract.

Part of the Requirements Master List

Drawings Requirements

1. The video display will be high resolution. (Cross reference: User code C1, requirement a; Developer code D1, requirement a) (Problem: The specification is incomplete.)

2. The video display will be capable of display 256 colors at any one time. (C2a, C5a, D6a, D4a) (Conflict between client and developer.)

3. Two new types of graphics primitives will be included in the graphics primitives set:

a. Cubic splines. (C6a, D4b) (Information is not volunteered.)

b. Three dimensional primitives: cubes, cylinders, and spheres. (C2b, D4b) (Inconsistent information.)

4. Graphics objects can be grouped together into subassemblies. (C1b, D1b) (Too difficult.)

5. Text annotations for graphics primitives and subassemblies can be included in drawings. (C3a, C4a, D2a) (Given at the wrong time.)

6. New operations on graphics objects will be supported:
7.
a. Cut and paste. (C2c, D4c) (Leads to a sidebar conversation.)

b. Scaling, rotation, and reflection. (C2c, D4c) (Improves existing capability.)

7. The system will produce three dimensional renderings of models. (C6b, D4d) (The specification is incomplete.)

8. Manufacturing specifications will automatically be produced from finished drawings. (C3b, D2b, D3a) (Overlapping requirements.)

Database Requirements

9. A database of reusable objects will be maintained. (C1c, D1c, D2c) (Imprecise definition of terms leads to conflict.)

10. Multiple versions of drawings will be managed through version control. (C2d, D3b) (Inconsistent requirements.)

11. Files in the database can be shared by several users concurrently. (C1d, D3c, D5a) (Technical problems need to be overcome.)

12. A parts inventory will automatically be produced from a finished drawing. (C3c, D3d) (Difficulty in previous specification is carried over.)

13. A price list will automatically be produced from a finished drawing. (C4b, C5b, D3e) (Difficulty in previous specification is carried over.)

14. An archive of finished drawings will be maintained over a long period of time. (C1e, D3f, D5b) (Technical problems need to be overcome.)