CS122 Lecture: Requirements Engineering

last revised January 11, 2016

Objectives:

1. To understand the importance of Requirements Analysis in the overall software development process
2. To understand the distinction between functional and non-functional requirements

Materials:

1. Projectable of book figures 2-4, 2-5
2. Projectable of book figure 2-6.
3. Quick Check Questions from book and answers
4. Projectable of Categories of Requirements
5. Requirements activity
6. Projectable of Tree Swing

I. Introduction

A. As we pointed out a while ago, there are many different processes that can be followed in software development (e.g. waterfall life cycle, RUP, etc).

B. Regardless of what process is followed, however, certain tasks will need to be done as part of the development process *per se* - whether all at once, iteratively, or incrementally. In fact, activities like these will be part of any situation in which one uses his/her professional skills to help solve someone else's problem - not just when creating software or even in a computer field.

1. Establishing Requirements: The goal of this is to spell out what constitutes a satisfactory solution to the problem.

2. Analysis. The goal of this is to *understand* the problem. The key question is "What?".

3. Design. The goal of this is to develop the *overall structure* of a solution to the problem in terms of individual, buildable components and their relationships to one another. The key question is "How?".

4. Implementation. The goal of this task is to actually *build* the system as designed.

5. Installation / Maintenance / Retirement

   All of these must be done in a context of commitment to Quality Assurance - *ensuring* that the individual components and the system as a whole do what they are supposed to do (which may involve identifying their shortcomings and fixing them.)

C. Today's focus is the first of these: establishing requirements - sometimes called requirements engineering. The question that is answered is "how will I know that I have found a satisfactory solution to the problem - i.e. what must characterize an acceptable solution? Though we will discuss the specific context of software development, establishing requirements is part of any design process.

   Example: Gordon recently built this building we are in. The design of the building was based on input garnered from science division faculty and others affected by the building (e.g. the registrar had some input regarding classroom space). A suitable building would have to incorporate space for various purposes.

D. What were the specific stages of requirements engineering the book discussed? (This is Quick Check Question a) What is the purpose of each stage?

   ASK

   1. Requirements elicitation - gathering information

   2. Requirements specification - putting the information into an ordered form

   3. Requirements validation - checking to be sure the requirements are consistent and complete

II. **Requirements Elicitation**

A. Since the book discussed this extensively, we will spend only limited time on it. What were some of the requirements elicitation techniques the book discussed?

ASK

1. Interviews with key people. Who might this include? (Think back to the notion of a stakeholder)
   a) Potential users of the system
   b) The client
2. Questionnaires
3. Studying documents
4. Observing the existing system (if there is one)

B. Quick-Check Questions / Exercises

1. What documents should be produced by the developer before and after an interview with a client or user of the system? (QC b)

   ASK

   a) Interview plan
   b) An interview summary (reviewed with interviewee)

2. When are questionnaires useful? (QC c)

   ASK

   When a small amount of information is needed from a large number of people whom it is not practical to interview individually.

3. What is a scenario? (QC d)

   ASK

   A narrative of a typical instance of fulfilling a given requirement

   a) Examples in the book - figure 2-4, 2-5 (pages 30, 31)

      PROJECT

   b) Scenarios serve as a bridge to the development of use cases - our next major topic

4. [As time permits] Exercise 2.5 p. 38 - do in pairs

III. **Requirements Specification**

A. Having gathered general information about a particular project, it is important to document the information.

B. Sometimes, this is done fairly informally.

1. This may take the form of a simple statement of the problem that is to be solved - a problem definition. (This is true for any problem solving process - not just with regard to software).

   Example: For the "Wheels" case study in the book, a problem definition appears as figure 2.6 (page 32) in the book.

   PROJECT Figure 2.6

   Note: often, it is helpful to see if one can capture the gist of such a statement in 1-2 sentences.

   Exercise: do this for the "Wheels" system.

2. Quick check question e: What are the typical sections of a problem definition?

   ASK

   cf figure 2.6 on page 32 - PROJECT

C. Sometimes, a specification is more formal.

1. In contract projects, the requirements specification document often becomes explicitly or implicitly a part of the formal contract between the developer and the client - i.e. the developer gets paid for developing a system that fulfills the specifications.

2. As part of a formal specification of requirements, it is common to give each requirement a number. This is done to facilitate traceability: i.e. checking that no requirements are inadvertently lost sight of during development, and that capabilities which are not required do not inadvertently creep into the system.

D. Whether specifications are informal or formal, one issue that needs to be made very clear at this point is the scope of the system - i.e. what is part of the system and what is not? (Again, this is part of identifying requirements in general, not just with regard to software).

   Example: the Science Building includes some classroom space in addition to office and laboratory space for the sciences; but it does not include administration office space (though such space is needed). That is, science faculty office space, laboratories, and classroom space are part of the scope of the project; administration office space is not.

1. Often, identifying the scope for a system can be done by listing all the sorts of things the system might do, then narrowing the list - recognizing that some areas rightly belong to a different project.

2. Often, the scope of a system is incorporated into the problem definition statement we just discussed.

Example: Problem definition for "Wheels" system - Note how section on scope explicitly states what is included and what is excluded.. (This is also implicit in the rest of the statement).

3. Exercise: consider the development of a software system for use by a car-rental agency. (Do on board)

   a) What are some things that might be part of such a system?

      ASK

   b) Now let's narrow the scope

      ASK

E. At this point, it is also useful to recognize that requirements fall into two broad categories:

   PROJECT

   1. Functional requirements are concerned with what functionality that system must provide - i.e. what a user of the software must be able to do.

   2. Non-functional requirements are other requirements that a satisfactory solution must satisfy. These can be just as important as functional requirements; for this reason, sometimes this category is fleshed out as

      a) Quality requirements.

         (1) Speed: For software, two measures of speed can be important:

            (a) Response time - the time needed to respond to an individual request.

(b) Throughput - the volume of requests that can be handled in a unit of time.

Example: for a web system, response time is measured in terms of the time between a user clicking on a link and the user seeing the actual page; throughput is measured in terms of how many users can be accessing the site at the same time. (Note that inadequate throughput capacity can show up as a response time problem under heavy load).

(2) Security - often a major issue with software systems:

(a) Transmission of information over a network (e.g. the use of encryption; https vs http)

(b) Break-ins to databases (e,g, theft of credit card numbers stored in databases is a recurring problem)

(3) Reliability - often measured in terms of mean time between failures (MTBF) - where "failure" is not limited to a system crash, but includes anything which interferes with a user getting work done.

(4) Recovery time after a failure - e.g. if a failure typically necessitates at system reboot that takes 10 minutes, then the recovery time is 10 minutes.

(5) Availability - a related issue. It is measured in terms of the percentage of the time a system is available for use, after recovering from failures and anything else that makes the system unavailable.

Example: suppose a certain system has a MTBF of 100 hours; that recovery from a failure (involving a reboot) takes on the average ten minutes, and that one hour every week the system must be taken off line for backup. Then availability is

5990 minutes / 6000 (ten minutes lost to reboot every 100 hours) - one hour / 168 hours = 99.83 % - .6% = 99.2%

(6) Resource utilization -

(a) Historically, storage requirements (RAM and/or disk) have been important considerations. As prices for memory and mass storage have plummeted, storage utilization has become a minor issue for many systems; however, it can still be a major issue for embedded systems.

(b) For embedded systems and devices like PDA's and Cell phones, power consumption is often the key issue. While one tends to thing of power consumption as a hardware issue, software has an impact as well - e.g. systems consume more power when they are actively computing or accessing disk; interestingly, even GUI features such as scrollbars can impact power consumption!

(7) Provision for maintenance and/or re-use

b) Platform requirements - having to do with the environment in which the software can operate (computing platform, peripheral devices, etc.)

c) Process requirements - including cost and delivery date as well as development process.

3. In addition to thinking about functional requirements and various kinds of non-functional requirements, one must also be aware of the possibility of spurious requirements - e.g. things that shouldn't be considered a requirement at all, since they do not help to determine what constitutes an acceptable solution to a problem.

   Often, spurious requirements deal with matters of *how* the problem is solved, rather than *what* a solution must be.

   Example: in most cases, a statement like "the software must be written in Java" would be a spurious requirement (unless there were some compelling reason why this requirement is necessary).

4. Do requirements classification activity. Have them work on in pairs, then work through answers and rationale. Project classification projectable while they are doing this.

F. Finally, it is important to recognize that cost considerations and/or conflicts between requirements frequently result in a need to prioritize requirements.

1. In general, it is often the case that not all of the requirements within acceptable costs for the project.

   a) Software projects always have a cost associated with them. Though cost is not stated as a requirement, it is a very real factor nonetheless.

      (1) For custom software, this may be the actual amount that the contractor bids for the project. [And if there are multiple bidders and the bid is too high, someone else will win the bid.]

      (2) For generic software, development cost impacts the final cost of the product. [If a reasonable forecast of sales is N units, then the development cost per unit is total cost / N, which must be an acceptable fraction of anticipated revenue per unit]

      (3) For software developed internally, there is an associated cost in terms of personnel and other resources. Though this may not be stated explicitly in dollar terms, it is still very real.

   b) Of course, this is not unique to software - it is true of any project.

      For example, in the case of this building, desirable features had to be adjusted due to cost considerations.

2. Moreover, it is often the case that various requirements conflict with one another, so it is not possible to have all of them. (We call these conflicting requirements).

   a) Occasionally, functional requirements conflict.

   b) More often, there can be a conflict between certain functional requirements and non-functional requirements.

   c) Quite often, there are conflicts between non-functional requirements that must be settled by deciding which are most important

      Examples of (b) and/or (c)?

      ASK

      Provision for certain functionality versus speed, security and/or size.

      Speed versus size

      Speed versus security

      Speed versus broad platform availability
      ...

IV. **Requirements Validation**

A. The final stage of requirements engineering - and an absolutely crucial one - is requirements validation. Here we are concerned with three major issues:

   1. Completeness
   2. Correctness
   3. Consistency

      (Note: some of the conflicts discussed above may actually be caught at this point).

B. QC question g in book - then collect QC's

C. In reality, it is easy to miss requirements. Experience has shown, however, that correcting missing or incorrect requirements becomes increasingly expensive as development proceeds. So it is very important to ensure that requirements are as complete, correct, and consistent as possible from the outset.

PROJECT Tree Swing Again