ESD264/1.264
Lecture 5 case studies
Fall, 2013

1. Chemical inventory system

Chris was in charge of version 1 of MIT's chemical inventory system, MITCIS. With higher security required these days, MIT realized that its decentralized system of ordering, delivering and disposing of chemicals had to change. Chris had a general idea of the capabilities necessary from his past experience and from the first meeting of the MIT steering committee for the project. Dave was the head of the committee. "Chris, how long is MITCIS going to take, he asked?"

"I think it will take about 9 months, but that's just a rough estimate at this point," Chris said.

"That's not going to work," Dave said. "I was really hoping you'd say 3 or 4 months. We absolutely must have MITCIS within 6 months; we can't tolerate our security exposure any longer than that, at the absolute worst. You know how serious this problem is, don't you? Can you do it in 6 months?"

"I'm not sure," Chris said honestly. "I'd have to look at the project more carefully, but I can try to find a way to get it done in 6."

"Treat 6 months as a goal, then," Dave said. "That's what it's got to be, anyway." The rest of the committee agreed.

By week 5, additional work on the system requirements had convinced Chris that the project would take closer to the initial 9-month guess than to 6 months, but he thought that with some luck he might still be able to complete it in 7 or 8 months. He didn't want to be labeled a troublemaker or uncooperative, and he wanted to stay in the good graces of the committee, since it would heavily influence his promotions and future assignments.

Chris' team made steady progress but requirements analysis took longer than they had hoped. Every lab and department had a different way of doing things, and it took a long time to reach agreement on a common approach. They were now almost 3 months into what was supposed to be a 6-month project. "There's no way we can do the rest of the work we have to do in 3 months," he told Dave. He told Dave he needed a 2-month extension and rescheduled the project to take 8 months. He felt it could be done in 9 or 10 months, realistically, but he didn't feel he could present those times to the committee.

A few weeks later, Chris realized that the designs for the database, Web pages and program weren't proceeding as quickly as he had hoped either. "Implement the parts you can do easily," he told the team. "We'll worry about the rest of the parts when we get to them."

Chris met with the MIT steering committee. "We're now 7 months into our 8-month project. Detailed design is almost complete, and we're making good progress but we can't complete the project in 8 months." Chris announced his second schedule slip to 10 months. Dave grumbled and asked Chris to find a way to bring the schedule back to about 8 months.

At the 9-month mark, the team had completed detailed design, but program development had still not begun on some parts of the system. It was clear that the team couldn't make the 10-month schedule either. Chris announced the third schedule slip, to 12 months. Dave's face turned red when Chris announced the slip, and the pressure from the MIT committee became more intense. Chris began to feel his job was on the line.

Coding proceeded fairly well, but a few areas needed redesign and recoding. The team hadn't coordinated design details among the Web page, database and Visual Basic sub-teams well, and some of the implementations conflicted. At the 11-month steering committee meeting, Chris announced the fourth schedule slip, to 14 months. Dave became livid. "Do you have any idea of what you're doing?" he yelled. "You obviously don't have any idea when the project is going to be done! I'll tell you when the project will be done! It's going to be done by the 12-month mark, or you're going to be out of a job! You and your team are going to work 80 hours per week until you deliver!" Chris felt his blood pressure rise, especially since Dave had backed him into an unrealistic schedule in the first place. But he knew that with four schedule slips, he had no credibility left.

Chris told his team about the meeting. They worked hard and managed to deliver the software in just over 13 months.

Answers:

a. **List at least 5 errors made by this development team in their execution of this project.**

1. No resource estimates (function points, lines of code, schedule months, person months) were made at the start of the project. Chris gave an off-the-cuff estimate when asked at the first meeting.
2. No uncertainty was communicated to the steering committee. Chris gave a point estimate, and he didn't state any assumptions on the size of the project, complexity of requirements, etc.
3. No discussions of tradeoffs in functions or team size to meet the 6-month date were held. Chris should have done this as requirements became complex, and even at the start of the project.
4. Chris did not communicate the true schedule as the project progressed. The real estimates were not given, and no proactive information was given before there was a problem. If requirements took twice as long as expected, so would every other part of the project, and this was not addressed.
5. The team did not use the spiral model (or other appropriate model such as evolutionary prototyping, etc.). They used a waterfall, which is inappropriate for rapid development, which this project clearly required.
6. The design was not coordinated properly since the pieces did not fit together. This is poor management and possibly poor use of tools.
7. Integration was not begun until too late in design and coding. Leaving the hard parts until the end is a mistake; they won't integrate properly.

b. Outline the key steps in the resource estimation that the team should have used to avoid these errors/problems. List at least 5 steps; describe each in 1-2 sentences or phrases.

1. Quickly estimate the number of Web pages (or VB screens) and database tables and external interfaces, if any, to be built. Estimate the complexity of each Web page, table, interface.
2. Estimate the function points for each.
3. Based on the type of software to be used (database, VB, etc.) for each part of the system, estimate the lines of code.
4. Look up the schedule and person months for the appropriate project type (business) and software maturity (nominal, at best).
5. Apply a range of uncertainty based on the stage of the project.

c. Outline the key steps in the software development process that the team should have used to avoid these errors/problems. List at least 5 steps; describe each in 1-2 sentences or phrases.

1. Use the spiral model (or other evolutionary model)
2. Use 3 month spirals to ensure that a system can be shipped at the 3 month point (a very simple one, but this ensures integration) and at the 6 month point (also likely to be very simple)
3. Do requirements in the first 3 weeks of spiral 1, instead of letting them go for 3 months. You can start coding with a rough set of requirements for the first spiral, since you'll be finding all sorts of other problems with the data, pages, interfaces, etc. Requirements should be roughly ¼ of the effort in one spiral
4. Do design in the next 3 weeks. It should also be ¼ of each spiral. Design went on for 6 months (!!) in this project, as long as the total project was supposed to take.
5. Build an initial prototype, similar to what you did in 1.264, in about 4-5 weeks. Make sure everything integrates. You can do better than in 1.264 because you have more time and more knowledge.
6. Test, review and debug for 2 weeks or so in spiral 1 and show an initial product to the steering committee at the 3-month mark. If it takes until the 4-month mark, that's not a major problem, though it will tell you that the product won't be complete until 7 months, since 3 month spirals are the shortest possible.
7. The committee, if it sees a working prototype at 3-4 months, will be more supportive, can help limit requirements changes, etc.

2. Warehouse management system

Your management has asked you to extend its current warehouse management system to receive input from RFID tags and readers for high value electronic equipment. This equipment, such as automated fareboxes, passenger counters, and train control systems, is sold by manufacturers to public transportation operators. There are many models and variations of each item; large numbers of spare items are carried in inventory in your warehouse and shipped overnight to public transit operators when a unit fails. Some inventory owned by the manufacturers is also kept at public transportation operator facilities for rapid replacement of failed units. It is very difficult to keep track of this expensive inventory. While all inventory is supposed to be barcode scanned monthly, this is a low priority task for the staff, and many expensive units are lost or otherwise written off. Also, with poor data on what the inventory levels are, much more inventory is held than needed to support the service levels and repair times promised by the manufacturers.

RFID tags are electronic transponders that can be read automatically by an RFID reader, without human intervention. The RFID-enabled warehouse will read all tags every few minutes, 24 hours per day, to provide continuous data on inventory levels. The warehouse is run by your organization, who acts as a third party logistics provider for 20 manufacturers of these devices. Each public transit operator who is a customer of yours will be required to place all manufacturer-owned inventory in a room also equipped with an RFID reader that, again, will automatically read inventory every few minutes.

RFID technology is somewhat untested in this environment. The functional needs of the manufacturers, your company and the public transportation operators are not well understood: the inventory levels, ownership, business terms, data flows and overall use of the new system are not well understood, since it will create substantial changes from current practice.

Answers:

1. You have been asked by your management to provide a short discussion of the system development/acquisition/customization approach you would take, as follows:

a. What lifecycle model would you use, and why? Also name at least two lifecycle models you would <u>not</u> use, and say why.

Spiral: Requirements, design, user interface hard to define at start, must be discovered. Time and resource estimates highly uncertain. New technology, which is risky. Likely changes in business process, from barcoding to RFID. Likely new interfaces with different identifiers (primary keys) between manufacturers and you. And others.

Evolutionary prototyping: If risks are in performance of RFID tags, readers and whether the system works, starting a prototype and getting it to work at all first, and then refining it, is appropriate. The software risks may be less than the hardware/new technology risks here.

Pure waterfall is not appropriate: requirements not knowable at start, locking in design early will be wrong, changes in business process will occur, many risks exist. Perception of little progress until end would not be acceptable to management in this situation.

Modified waterfall: not ideal for the same reasons as waterfall.

Staged delivery: again, not ideal for same reasons as waterfall. Requirements and design are unlikely to be fixed at the start, or to be modified only slightly. A full spiral or evolutionary prototype are much more appropriate.

Code and fix: unacceptable in any project: No control, poor quality, no progress estimates

b. Your management asked how long it would take to prepare a first release of the system. Briefly describe how you would estimate this time.

1. Write an initial requirements document, probably as use cases, sequence diagrams, state models and other UML documents to define the system.
2. Count function points; plan significant extra function points in software related to the RFID hardware, since it will be revised many times as the system proceeds. This is close to ‘software research’, which is hard to estimate. Treat RFID-related functions as complex. If off-the-shelf software is used for components, estimate the modifications needed to get the function points (table changes, user interface changes, interface changes).
3. Treat all RFID-related software as systems software; treat your team as nominal even if highly skilled, since no one has much RFID experience.
4. Estimate lines of code, schedule time, person-months and team size from tables.
5. Apply a very wide range of convergence in the first spirals or prototypes, since the product definition will not be as stable as in better-understood technologies.
6. Communicate ranges to management, not point estimates.

c. List at least four major risks in this project, and how you lessen them.

1. Hardware not working. Prototype early to understand how well it works.
2. Changing requirements. With new process and new information previously unavailable, there will be many changes to requirements, desired reports and outputs, frequency of data transmission, etc. Spiral model to plan for evolving requirements.
3. Changing business process due to project. New actors, new reports, disputes among parties whose inventory control or costs will change. Spiral model to plan for requirements changes driven by these issues.
4. Software churn due to changing requirements. Throwaway prototype might be best approach. Throw it away once all is understood and rewrite requirements, redo the design and write/configure/modify the software and hardware from scratch.
5. Schedule visibility and control. It will be difficult to keep on schedule. Principled negotiation based on estimate convergence graph can be used to communicate and control.
6. People problems. This will be an exciting but difficult project. Insist on best staff.
7. Process problems. Many pressures will exist to derail your software process. Stay with it carefully. Don’t shortchange requirements, design, QA steps.