CSE435 Software Engineering
Requirements Elicitation and Specification

Acknowledgements: M. Langford, Pfleeger/Atlee, McUmber, Easterbrook

Review

- **What is a software process?**
  - A set of activities required to develop software
- **What are some examples of formal process models?**
  - Waterfall, V-Model, Iterative Model, Spiral Model
- **What activity does every one of these process models begin with?**
  - Communication
  - Gathering Requirements
Motivation

- **In order to solve a problem, we must understand the problem**
  - Before we can make software, **we need to question what is needed**
- **Communication is fundamental but complex**
  - Cultural differences, assumptions, **confused messages**
- **Shannon-Weaver Model**

![Shannon-Weaver Model Diagram]

How do we bridge the gap between customers and developers?

Claude Shannon: Michigander (inventor of "bit"; early work on cryptography, chess-playing computers; https://www.britannica.com/biography/Claude-Shannon)

System Requirements

- System **requirements** state **what is expected**
  - A condition/capability needed to solve a problem
  - A condition/capability that must be possessed by a solution

Requirements tell us **what the problem is and what a solution looks like**.

- All formal software processes begin with **capturing requirements**
  - **Requirements elicitation** – collecting requirements from customer/stakeholders
  - **Requirements analysis** – modeling the desired behavior for better understanding
  - **Requirements specification** – defining requirements in clear language
  - **Requirements validation/verification** – verifying and validating specified requirements
The Requirements Process: Process for Capturing Requirements

- Performed by the req. analyst or system analyst
- The final outcome is a Software Requirements Specification (SRS) document

Cost of Fixing Errors By Phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Median Cost to Fix an Error by Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1x</td>
</tr>
<tr>
<td>Design</td>
<td>5x</td>
</tr>
<tr>
<td>Code</td>
<td>10x</td>
</tr>
<tr>
<td>Test</td>
<td>50x</td>
</tr>
</tbody>
</table>

Errors significantly easier to address during requirement phase.
Requirements Elicitation:
Stakeholders

- **Clients**: pay for the software to be developed
- **Customers**: buy the software after it is developed
- **Users**: use the system
- **Domain experts**: familiar with the problem that the software must automate
- **Market Researchers**: conduct surveys to determine future trends and potential customers
- **Lawyers or auditors**: familiar with government, safety, or legal requirements
- **Software engineers** or other technology experts

Requirements Elicitation
Approaches to Eliciting Requirements

- Interviewing stakeholders
- Reviewing available documentations
- Observing the current system (if one exists)
- Apprenticing with users to learn about user's task in more details
- Interviewing user or stakeholders in groups
- Brainstorming with current and potential users
Requirements Elicitation

- **Communicate with stakeholders**
  - Identify a list of stakeholders to question.
  - Question customers/end-users to discover and prioritize needs.
  - Develop scenarios to describe user interactions.
  - Reconcile/negotiate any conflicting requirements.

- **Customer Interviews**
  - Begin with a general questions to ease into the interview.
  - Delve into questions about the problem to be addressed.
  - End by asking for suggestions for others to talk to.

Interview Questions

- **Ask open-ended questions**
  - Avoid yes/no questions.
  - Force the customer to elaborate.
  - **What** is the problem?
  - **Why** does the problem exist?
  - **When** does the problem occur?
  - **Where** does the problem occur?
  - **How** is the problem handled now?

- **Don’t directly ask what the customer wants**
  - Customers do not know the solution.
  - Focus on what the problem is.

  - Would XYZ be a useful feature?  
    - **Close-Ended Question**
  - How would XYZ be useful for you?  
    - **Open-Ended Question**
  - What would you like us to do?  
    - **Solution-Focused Question**
  - What are you trying to do?  
    - **Problem-Focused Question**
Sample System: Smart/Adaptive Cruise Requirements

Credit: W. McUmber, W. Milam

Interview Questions

- **Example**
  - **Q:** When a vehicle cuts in front of the car, you have to slow down quickly and not hit it, right?
  - **A:** Yes

We learned nothing.
Open-ended questions

Q: What happens when a car cuts in front of you?
A: Well, if the lead car is too close, the driver has to intervene or else a crash results. I guess we need a warning light in this case. If the car is moving faster, you don’t have to do anything. He’s pulling away. I guess the only time brakes are used is when the closing speed is too high for the distance and yet within the capabilities of the system to slow down. But I guess if a collision is imminent, we should max out the braking.

Now, we learned something...

Dialogue with different responses

Q: Tell me what should happen if a car cuts in front of our car too close to avoid a collision?

Q: What? Are you nuts? We should at least try to stop. Shouldn’t we?

A: Perhaps...

Q: We have quite a bit of braking power in the system. What would happen if we used it here?

A: Well, I guess it could avoid a collision and at least get the car slowed down but the attorneys tell me we don’t want the system active when a collision occurs.

Ah ha! Non-technical constraint

You are done at this point, and still unresolved.
From elicitation to analysis...

Your interview should result in a large volume of facts that must be analyzed to derive requirements

- Here “analysis” involves both analysis and synthesis
- Synthesis: attempt to compose a coherent “model” of the problem requirements

A model can be analyzed to:

- identify potentially inconsistent facts, and
- infer facts that should be true

Both of these issues must be clarified, often via a second client interview

Putative questions

Asks about a situation in a way that tests your model of the domain

SWE:
“If a lead vehicle turns, or otherwise is not in front of the car anymore, the car can resume the previous speed, correct?”

CLIENT:
“Yes, exactly.”

Very specific question that tests the idea of cruise plus collision avoidance
Sample Interview I

SWE: Could you tell me about the cruise control system?

CLIENT Yes, normal cruise control holds a fixed speed. What we want is to make the car “smart” so that it slows down when there is a vehicle in front of it.

SWE: What does a driver currently do in this situation?

CLIENT Currently, the driver can step on the brakes to disengage the cruise, or turn the cruise off completely. Or, not use the cruise.

SWE: Why is turning off the cruise this way a problem?

i.e., Why do you need “smart” cruise?

Try to get at the motivation for the problem

Sample Interview II

CLIENT In an urban environment, say I-75 in Detroit, using the cruise becomes irritating, but really we are more interested in avoiding collisions.

SWE: Tell me more about the collision avoidance aspect, please.

CLIENT If we limit how close a lead vehicle can get, and control the speed while the car is in trail, the chances of a collision can be greatly reduced.

SWE: How would a system avoid a collision in a typical scenario?

CLIENT Suppose the driver is following a truck, but at a higher speed than the truck. As the car closes, the system could alter the speed to match the speed of the truck.

SWE: What does the slowdown profile look like?
Sample Interview III

CLIENT
Well, we have discovered that slowing down linearly over a long distance can lead to other cars cutting in front of you. This is also not what a human driver does. Instead, we continue at our current speed and start a coast when we compute that we will get too close.

SWE: What is "too close"
CLIENT Oh, within 2 seconds of trail distance
SWE: Does that mean at 60 mph, 88 ft/sec, too close is 176 ft?
CLIENT Yes, closer than 176 ft is too close.

Sample Interview IV

SWE: What if a car cuts in front of you within the "safe" 2 second distance?

CLIENT I guess since there is nothing the system can do. Turn off the controller and hope the driver brakes in time.

SWE: The specs indicate we have a fair amount of braking power available. What would be the problem with using it here?

CLIENT The system does have access to the brakes, which are anti-lock. Technically, we could apply the brakes, but at the moment, our attorneys tell us we’d rather not have the system active if a collision is imminent.
Recap: Types of Questions as Tools

- **Why...** Usually leads to deeper motivations, information on structure.
- **What...** Usually leads to facts
- **How...** Usually leads to a discussion of process, not structure
- **Could...** Maximally open, might lead to no data
  “could you explain why the safety zone is 2 seconds?”

Elicitation/analysis structure

Elicitation/analysis may require multiple interviews

- ask questions to obtain information
- build information into your "model"
- answer matches
- pose putative questions
- wrong answer
- figure out where the ambiguity or problem is
Summary

Elicitation is critical to:

- address the requirements-completeness problem
- support analysis, which aims to address the requirements-consistency problem

Client interviews are a useful tool, but:

- Must be carefully planned and orchestrated
  - Meetings should focus on a primary goal (e.g., information extraction vs. clarification)
  - Big mistake to fail to plan for some iteration here

Interview Questions

- **Types of questions**
  - *What* – leads to facts about the problem
  - *Why* – leads to deeper motivation about the problem structure
  - *How, Where, When* – leads to details about process

- **Don’t ask leading questions**
  - *Do not provide example solutions*
  - Try to get *unbiased perspective*
Interview Questions

- **Other tips**
  - Don’t be afraid to ask the “obvious;” *don’t make assumptions*
  - Ask questions from *multiple angles* (spread apart)
  - Ask *follow-up questions*
  - Allow for *periods of silence*
  - Take *detailed notes*
  - *Space interviews* apart

After Interviews

- **Summarize and consolidate key insights**
  - Identify major *goals to be achieved*
  - *Rank goals* by importance
  - Identify any *conflicting goals*
  - Identify any gaps in understanding or *missing information*
  - *Develop goal models* to describe how goals are related
  - *Develop use cases* to describe user interactions with the system
  - Determine how to *validate requirements* (acceptance testing)

It may take *multiple iterations* of interviews to refine requirements specification.
Functional Requirements

- **Functional requirements** state what the system should do
- What services should the system provide?
- How should the system react to specific inputs?
- How should the system behave in particular situations?

**Examples**
- “The system shall display a list of product codes and descriptions to the user.”
- “The system shall enable a user to search for keywords in product descriptions.”
- “The system shall display matching product codes/descriptions based on search.”

Functional requirements **describe system behavior.**

Invariant Requirement

- **Invariant:**
  - Something that is always true
- **Invariant requirement:**
  - A requirement that specifies a condition that always holds
  - An event that shall always produce a specific behavior
    - Example: Braking always disengages cruise control
- **Safety-critical systems typically have one or more invariant requirements:**
  - Consider invariant requirement for the Therac-25
  - Potential impact of invariant (or lack of invariant)?
Non-Functional Requirements

- **Non-functional requirements** clarify what a solution should look like
  - **Quality constraints** on the system
  - Deal with issues like performance, security, portability, scalability, etc.

- **Examples**
  - “All user search queries shall be encrypted with a PEKS scheme.”
  - “Reports generated by the system shall not exceed 100 kilobytes.”
  - “Reports generated by the system shall be generated in less than 1 minute.”

Non-functional requirements **describe constraints on system.**

Example Requirement Specification

- **Mental Health Care, Patient Management System (MHS-PMS)**

  1. The MHC-PMS **shall** generate monthly reports showing the cost of drugs prescribed, their cost, and the prescribing clinics.

     1.1. On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics **shall** be generated.

     1.2. The MHC-PMS **shall** automatically generate the report for printing after 17:30 on the last working day of the month.

     1.3. A report **shall** be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

Requirements are specified **hierarchically, as enumerated lists.**
Writing “Good” Requirements

- Requirements are specified using **natural language**
  - **Expressive** (good) but also **vague** (bad)

- **Characteristics of a “good” requirement**
  - **Cohesive** – addresses one and only one system property/expectation
  - **Complete** – fully states in a single place with no missing information
  - **Consistent** – does not contradict other requirements
  - **Unambiguous** – avoids use of uncommon technical jargon, acronyms, verbiage
  - **Verifiable** – can be demonstrated, measured, tested

Any two people reading a requirement should have **the same understanding**.

“Bad” Requirement Example

“To assist in the position of entities on a diagram, the user may turn on a grid in either centimeters or inches, via an option on the control panel.”

- **Issues**
  - Mixes functional and non-functional requirements into a single statement
  - Leaves outs details for the reader to infer (where/how should the grid appear?)
“Better” Requirement Example

1. “The editor shall provide a grid panel comprising horizontal and vertical lines to provide a background to the editor window.”
   1.1. “The user shall be responsible for the appearance of grid lines.”
   1.2. “The control panel shall provide functionality to turn on and off grid lines.”
   1.3. “The grid panel shall space horizontal and vertical lines in units measured by centimeters or inches.”
   1.4. “The grid panel shall initially space horizontal and vertical lines in units measured by inches.”

- Requirement is more explicit and decomposed into atomic elements

Common Problems

- **Implementation details**
  - Functional requirements should only state what is needed
  - Non-functional requirements state constraints on solution (generic, high-level)
  - There could be multiple solutions (leave it to the design phase)

- **Standard usage of terms**
  - All “shall” statements must be verifiable
  - Terms such as “are, is, was, will, and must” do not belong in a requirement
  - Avoid using “and/or” and “etc.”
More Considerations

- **“XYZ must have good usability”**
  - *What does this mean?*
  - Describe the users and what is expected from them.
  - Provide some *quantifiable* way to measure usability.

- **“XYZ should respond in less than X seconds”**
  - *Under what conditions?*
  - *Provide a context* and description of an ideal environment.
  - Provide acceptable *error bounds*

Formal Documentation

- **Software Requirements Specification (SRS)** *(IEEE std 830-1998)*
  - Establishes a *basis of agreement* between stakeholders
  - *Reduces the development effort*
  - Provides a basis for *estimating costs and schedules*
  - Provides a *baseline for verification and validation*
  - *Facilitates transfer* of software products to new customers/developers
  - Serves as a *basis for enhancement*

Our group projects will be developing SRS for our industrial customers.
Summary

- **Requirements Elicitation** is the act of communicating with stakeholders to discover their needs
  - Ask open-ended questions (What are you trying to do?)
  - Avoid assumptions

- **Requirements Specification** is the act of formally defining stakeholder needs in a complete and unambiguous format
  - Requirements can be *functional* or *non-functional*
  - Should be cohesive, complete, consistent, unambiguous, and verifiable

Arguably the hardest part of the software process.

---

When a user takes a photo, the app should check whether they’re in a national park…

Sure, easy GIS lookup, gimme a few hours.

…and check whether the photo is of a bird.

I’ll need a research team and five years.

In CS, it can be hard to explain the difference between the easy and the virtually impossible.

xkcd [1425]