Pragmatic Programmer Tip: Don’t Gather Requirements – Dig for them

Requirements rarely lie on the surface. They’re buried deep beneath layers of assumptions, misconceptions, and politics.
Readings

- Requirements and Use cases:
  - Writing Effective Use Cases, Alistair Cockburr
  - Pragmatic Programmer, p. 202-208

- Paper prototypes
  - http://www.snyderconsulting.net/article_paperprototyping.htm
  - http://www.snyderconsulting.net/article_paperprototyping.htm
Outline

- What are requirements?
- Some interesting requirements facts
- How can we gather requirements?
- How can we specify requirements?
Software requirements

- requirements: specify what to build
  - tell "what" and not "how"
  - tell the problem, not the solution (in detail)

Some goals of doing requirements:
- understand precisely what is required of the software
- communicate this understanding precisely to all development parties
- control production to ensure that system meets specs (including changes)
General classes of requirements

- features – core features, additional, stretch
- quality
- performance
- schedule
- user interface
- system – browsers, os
- interoperability
- accessibility/ease of use
General classes of requirements

Examples requirements types:
• Feature set
• GUI
• Performance
• Reliability
• Expansibility (ie. support plug ins)
• Environment operates in (ie. HW, OS, browsers)
• Schedule
How do we gather requirements?

Let’s start with two facts:

- Standish group survey of over 8000 projects, the number one reason that projects succeed is user involvement.
- Easy access to end users is one of three critical success factors in rapid-development projects (McConnell).
How do we gather requirements?

Benefits of working with customers:
- Good relations improve development speed
- Improves perceived development speed
- They don’t always know what they want
- They do know what they want, and it changes over time
The most difficult part of requirements gathering is not the act of recording what the users want; it is the exploratory, development activity of helping users figure out what they want.

McConnell, SG
Words of Wisdom 2

Work with a User to Think Like a User – it’s the best way to get insight on how the system is easily used

Pragmatic Programmer Tip
How can we work with our customers?

What can we do during the lifecycle stages of:

- **Planning**
  - select lifecycle
  - identify real customer
  - establish interaction method

- **Requirements Analysis**
  - help customer determine what they want (ie. prototypes)
  - videotape customers operating, use cases
  - surveys, meetings, focus groups, discussions

- **Design**
  - design for change

- **Construction**
  - implement to allow change
  - show customer tangible signs of progress, phased delivery allowing feedback
Throughout your travels with the customer, be sure to set reasonable customer expectations.
How can we specify requirements?

So… we’re working with the customer to understand their needs, how do we capture these requirements?

Possibilities include:
  • Prototype
  • **System Requirements Specification Document**
    • Use Cases
    • Feature List
    • Paper GUI prototype
Use cases

**use cases**: written descriptions of user's interaction with the software product to accomplish a goal

"A behaviorally related sequence of transactions performed by an actor in a dialogue with the system to provide some measurable value to the actor" (Jacobson 1995)

- Use cases describe the ways in which a system can be used
- Use cases help us discover/document requirements
Benefits of doing use cases

- The list of goal names provides executives:
  - Shortest summary of what system will contribute
  - Project planning skeleton (priorities & timing)

- The main success scenario provides all:
  - Agreement as to the system’s responsibilities

- The extension conditions provide programmers:
  - List of things programmers have to watch for
  - List of things analysts have to investigate
**Actors and stakeholders**

- **What is an actor? A primary actor?**
  - **actor**: anything with behavior that acts on the system
  - **primary actor**: initiates interaction to achieve goal
    (when system is a software product, primary actor is often the computer user)
  - **supporting actor**: performs sub-goals to help use case

- **What is the difference between an actor and a stakeholder?**
  - **stakeholder**: anyone interested in the system
    - examples: supplier, stock agency, vendor
    - the difference: stakeholder might not "act" in any scenario
Use case goals and levels

- goal: action that actor wants to accomplish
- level: type / scope of a goal
  - summary goals: multi-sitting
  - user goals: one sitting
  - subfunctions: partial
Use Cases

A use case characterizes a way of using a system.
It represents a dialog between a user and the system, from the user’s point of view.
Benefits of doing use cases

- Establish an understanding between the customer and the system developers of the requirements (success scenarios)

- Alert developers of problematic situations (extension scenarios)

- Capture a level of functionality to plan around (list of goals)
Terminology

**Actor**: agent that acts on the system

**Primary actor**: initiates interaction to achieve the goal

**Stakeholder**: agent with a vested interest in the behavior of the system

**Goal**: action that actor wants to accomplish

**Level**: type / scope of a goal
- summary goals: multi-sitting
- user goals: one sitting
- subfunctions: partial

CSE 403, Spring 2007, Alverson
Goals and levels

What level are these goals at?

- Checkout a video from the library
- Purchase a book from the online store, and have it shipped to the user; can be cancelled while in transit
- Update user's balance after a deposit
- Add a class to a course schedule

Answers:

- user goal, summary goal, subfunction, user
Do use cases capture these?

Which of these requirements should be represented directly in a use case?

- Order cost = order item costs * 1.06 tax.
- Promotions may not run longer than 6 months.
- Customers only become Preferred after 1 year
- A customer has one and only one sales contact
- Response time is less than 2 seconds
- Uptime requirement is 99.8%
- Number of simultaneous users will be 200 max

Answer: *NONE*. Most of these requirements are non-functional, so the use cases wouldn't explicitly mention them. The user doesn't see them directly in the success scenario.
Styles of use cases

1. Use case diagram (UML/Visio/Violet)
   - shows all use cases in system
2. Informal use case
3. Formal use case

Let's examine each of these in detail...
1. Use case summary diagrams

The overall list of your system's use cases can be drawn as high-level diagrams, with:

- actors as stick-men, with their names (nouns)
- use cases as ellipses with their names (verbs)
- line associations, connecting an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use / rely on

Library patron

Check out book
# Use case summary diagrams

It can be useful to create a list or table of primary actors and their "goals" (use cases they start). The diagram will then capture this material.

<table>
<thead>
<tr>
<th>Actor</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Patron</td>
<td>Check out book</td>
</tr>
<tr>
<td></td>
<td>Search for book</td>
</tr>
<tr>
<td></td>
<td>Reserve book</td>
</tr>
<tr>
<td>Librarian</td>
<td>Record new book</td>
</tr>
<tr>
<td></td>
<td>Search for book</td>
</tr>
<tr>
<td></td>
<td>Generate catalog</td>
</tr>
</tbody>
</table>
Use case summary diagram 1

Library System

- Check out
- Search
- Reserve
- Record new
- Gen catalog

Library Patron

Librarian
Use case summary diagram 2

- **Trading Manager**
- **Trader**
- **Accounting System**
- **Set Limits**
- **Update Accounts**
- **Analyze Risk**
- **Valuation**
- **Price Deal**
- **Capture Deal**
- **Limits Exceeded**
- **Use Case**

- «extends»
- «uses»
2. Informal use case

informal use case: written as a paragraph describing the scenario/interaction

• Example:
  
  - Patron Loses a Book
    The library patron reports to the librarian that she has lost a book. The librarian prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The system will be updated to reflect lost book, and patron's record is updated as well. The head librarian may authorize purchase of a replacement tape.
### 3. Formal use case

<table>
<thead>
<tr>
<th>Goal</th>
<th>Patron wishes to reserve a book using the online catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary actor</td>
<td>Patron</td>
</tr>
<tr>
<td>Scope</td>
<td>Library system</td>
</tr>
<tr>
<td>Level</td>
<td>User</td>
</tr>
<tr>
<td>Precondition</td>
<td>Patron is at the login screen</td>
</tr>
<tr>
<td>Success end condition</td>
<td>Book is reserved</td>
</tr>
<tr>
<td>Failed end condition</td>
<td>Book is not reserved</td>
</tr>
<tr>
<td>Trigger</td>
<td>Patron logs into system</td>
</tr>
</tbody>
</table>
| Main Success Scenario | 1. Patron enters account and password  
2. System verifies and logs patron in  
3. System presents catalog with search screen  
4. Patron enters book title  
5. System finds match and presents location choices to patron  
6. Patron selects location and reserves book  
7. System confirms reservation and re-presents catalog |
|----------------------|--------------------------------------------------|
| Extensions (error scenarios) | 2a. Password is incorrect  
2a.1 System returns patron to login screen  
2a.2 Patron backs out or tries again  
5a. System cannot find book  
5a.1 … |
| Variations (alternative scenarios) | 4. Patron enters author or subject |
Steps in creating a use case

1. Identify actors and their goals
2. Write the main success scenario
3. Identify and list possible failure extensions and how they’re handled
4. Identify and list possible variations
1. Identify actors and goals

What computers, subsystems and people will drive our system? (actors)

What does each actor need our system to do? (goals)

Result: a list of use cases, a sketch of the system
  - can now draw [UML] use case diagram for each reference
Identify actors/goals exercise

- Let’s identify some major actors and their goals for the 4 projects
  - UTalk
  - SuiteRates
  - CourseRegistration
  - Sustainability
2. Write the success scenario

- Main success scenario is the preferred "happy" case
  - easiest to read and understand
  - everything else is a complication on this

- Capture each actor's intent and responsibility, from trigger to goal delivery
  - say what information passes between them
  - number each line
3. List the failure extensions

- Usually, almost every step can fail
- Note the failure condition separately, after the main success scenario
- Label with step number and letter:
  - 5a failure condition
    - 5a.1 use case continued with failure scenario
    - 5a.2 continued

Exercise: What are some failure conditions for posting a spreadsheet to the sustainability db?
4. List the variations

- Many steps can have alternative behaviors or scenarios
- Label with step number and alternative
  - 5’. Alternative 1 for step 5
  - 5”’. Alternative 2 for step 5

Exercise: What are some alternatives that arise around looking up a course in the school catalog?
Qualities of a good use case

- A good use case:
  - starts with a request from an actor to the system
  - ends with the production of all the answers to the request
  - defines the interactions (between system and actors) related to the function
  - takes into account the actor's point of view, not the system's
  - focuses on interaction, not internal system activities
  - doesn't describe the GUI in detail
  - has 3-9 steps in the main success scenario
  - is easy to read

- A good use case summary fits on a page
Exercise: Project use case

- Each project team break into 2 groups
- Meet up with one of your customer groups
- Together write a use case for each of your products
- Choose one per project team to share with the class

You will be able to use these in your SRS document!
Pulling it all together

How much is enough?

You have to find a balance

• comprehensible vs. detailed correctness
• graphics vs. explicit wording and tables
• short and timely vs. complete and late

Your balance may differ with each customer depending on your relationship and flexibility

CSE 403, Spring 2007, Alverson
Words of Wisdom 5

After you create a specification, go over it to:
- Eliminate all requirements not absolutely necessary
- Simplify those that are more complicated than necessary
- Substitute cheaper options when available
- Move non essentials to future releases
I'll need to know your requirements before I start to design the software.

First of all, what are you trying to accomplish?

I'm trying to make you design my software.

I mean what are you trying to accomplish with the software?

I won't know what I can accomplish until you tell me what the software can do.

Try to get this concept through your thick skull: the software can do whatever I design it to do!

Can you design it to tell you my requirements?