Lecture 08: Requirements Gathering Techniques (Part II)

"The goal of requirements engineering is to develop high quality – <u>not perfect</u> – requirements that allow you to proceed with construction at an <u>acceptable level of risk</u>."
-- from "Software Requirements", Karl Wiegers

Outline

- Techniques:
  - Use Cases / Usage Scenarios (covered)
  - Commonality and Variability Analysis (covered)
  - Frequent Customer Feedback
  - (Throwaway) Prototyping
- Risks from Inadequate Requirements Processes
- Discussion Questions

Resources

- *"Software Requirements"*, by Karl Wiegers
- *"Rapid Development"*, by Steve McConnell
  - Ch. 10, 14.1
- *"The Pragmatic Programmer"*, by Andrew Hunt and David Thomas
  - Ch. 7 – all of it is relevant
- Standish report
  - http://www.standishgroup.com/

Frequent Customer Feedback

- Why work with customers?
  - Good relations improve development speed.
  - Improves perceived development speed.
  - Customers don't always know what they want.
    - Are requirements ever exact and clear?
  - Customers do know what they want, but it changes over time.
    - So when are the requirements final?
  - No need to make dangerous assumptions about what customers want, or whether it is final and complete.
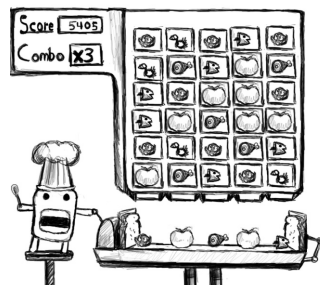 - <u>Bottom line</u>: improved efficiency, less rework, reduced risks, less friction

---

Throwaway Prototyping

- Using a rough sketch / diagram to show your understanding and to evoke customer response
- Example:
  © Busta' Sandwich Co.

- <u>Caution</u>: Do not overdo it! It must look and remain throwaway.

Risks from Inadequate Requirements Processes

- Insufficient user involvement => ?
- Creeping user requirements => ?
- Ambiguous requirements => ?
- Gold-plating by developers and users => ?
- Minimal specifications => ?
- Overlooking the needs of certain user classes => ?
- Incompletely defined requirements => ?

Risks from Inadequate Requirements Processes

- Insufficient user involvement => unacceptable products
- Creeping user requirements => overruns and degraded product quality
- Ambiguous requirements => ill-spent time and rework
- Gold-plating by developers and users => unnecessary features
- Minimal specifications => missing key requirements
- Overlooking the needs of certain user classes => dissatisfied customers
- Incompletely defined requirements => accurate project planning and tracking impossible

A Word of Advice on Managing Requirements

- After you put together a requirements specification, go over it to:
  - Eliminate all requirements not absolutely necessary
  - Simplify those requirements that are more complicated than necessary
  - Substitute cheaper options when available
  - Move non-essential requirements to future releases

© Scott Adams, Inc./Dist. by UFS, Inc.
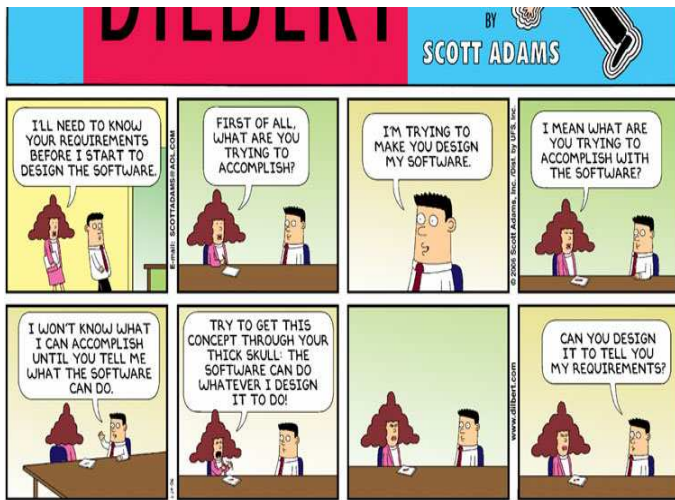
Feature / Scope Creep

"The software was late and far over budget; in fact, it almost didn't make it out the door. And it bore little resemblance to their original plans…"
-- from "The Wall Street Journal"

"Our analysis found that the average requirements overrun on our projects is about 40%."
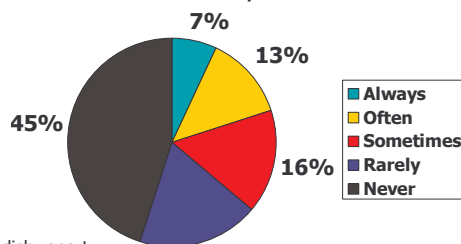-- from Construx analyses

Feature Creep Estimated

Software Features
Actually Used

7%
13%
45%
16%
19%

- Always
- Often
- Sometimes
- Rarely
- Never

Reference: Standish report, http://www.standishgroup.com/

Strategies to Manage Feature / Scope Creep

- Scope change document
  - May feel bureaucratic, but prevents frivolous changes to product scope and feature set
  - Need to first analyze cost & impact, then decide on tradeoffs
- Change control board