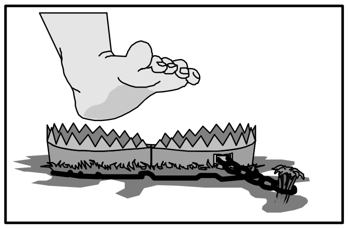
Software Requirements: 10 Traps to Avoid



Karl E. Wiegers
Process Impact
www.processimpact.com

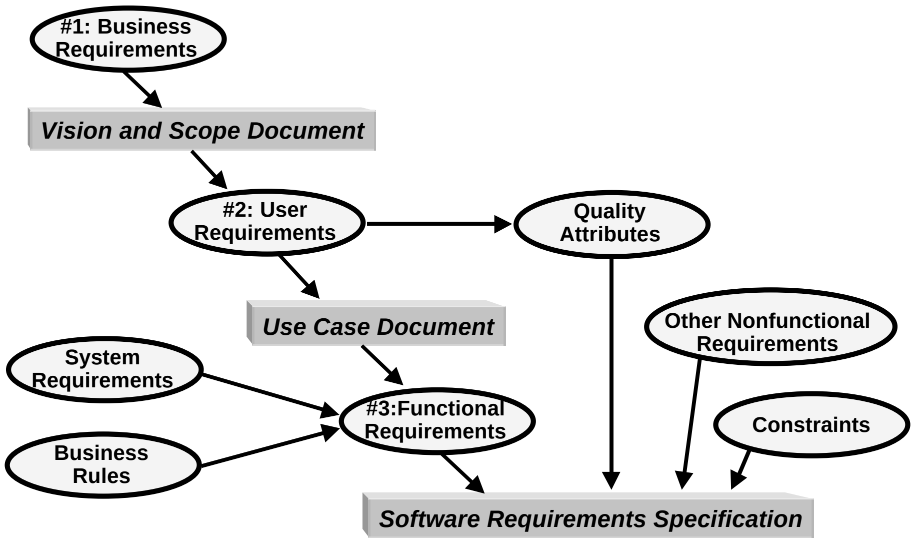
version 2

Trap #1: Confusion Over "Requirements"

Symptoms

- Stakeholders discuss "requirements" with no adjectives in front.
- Project sponsor presents a high-level concept as "the requirements".
- User interface screens are viewed as "the requirements".
- User provides "requirements," but developers still don't know what to build.
- Requirements focus just on functionality.

Three Levels of Software Requirements


#1: Business Requirements
Vision and Scope Document
#2: User Requirements
Quality Attributes
Use Case Document
Other Nonfunctional Requirements
System Requirements
#3:Functional Requirements
Constraints
Business Rules
Software Requirements Specification

Trap #1: Confusion Over “Requirements”

Solutions

- Adopt templates for three levels of requirements.
  - ✓ business requirements (Vision & Scope Document)
  - ✓ user requirements (Use Case Document)
  - ✓ functional requirements (Software Requirements Specification)
- Distinguish functional from nonfunctional requirements.
  - ✓ quality attributes, constraints, external interface requirements, business rules
- Classify customer input into the different categories.
- Distinguish solution ideas from requirements.

Trap #2: Inadequate Customer Involvement

Symptoms

- Some user classes are overlooked.
- Some user classes don't have a voice.
- User surrogates attempt to speak for users.
  - ✓ user managers
  - ✓ marketing
  - ✓ developers
- Developers have to make many requirements decisions.
- Customers reject the product when they first see it.

Trap #2: Inadequate Customer Involvement

Solutions

- Identify your various user classes.
- Identify product champions as user representatives.
- Convene focus groups.
- Identify decision-makers.
- Have users evaluate prototypes.
- Have user representatives review the SRS.

Trap #3: Vague & Ambiguous Requirements

Symptoms

- Readers interpret a requirement in several different ways.
- Requirements are missing information the developer needs.
- Requirements are not verifiable.
- Developer has to ask many questions.
- Developer has to guess a lot.

Software Requirements: 10 Traps to Avoid 7 *Copyright © 2001 by Karl E. Wiegers*

Trap #3: Vague & Ambiguous Requirements

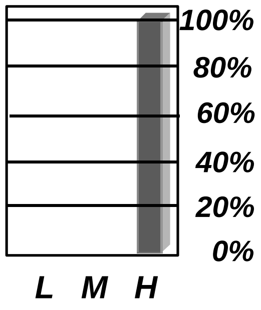
Solutions

- Formally inspect requirement documents.
- Write conceptual test cases against requirements.
- Model requirements to find knowledge gaps.
- Use prototypes to make requirements more tangible.
- Define terms in a glossary.
- Avoid ambiguous words:
  - ✓ minimize, maximize, optimize, rapid, user-friendly, simple, intuitive, robust, state-of-the-art, improved, efficient, flexible, several, and/or, etc., include, support

Software Requirements: 10 Traps to Avoid 8 *Copyright © 2001 by Karl E. Wiegers*

Trap #4: Unprioritized Requirements

Symptoms



- All requirements are critical!
- Different stakeholders interpret "high" priority differently.
- After prioritization, 95% are still high.
- Developers don't want to admit they can't do it all.
- It's not clear which requirements to defer during the "rapid descoping phase."

Software Requirements: 10 Traps to Avoid 9 *Copyright © 2001 by Karl E. Wiegers*

Trap #4: Unprioritized Requirements

Solutions

- Align functional requirements with business requirements.
- Align functional requirements with high-priority use cases.
  - ✓ frequency of use
  - ✓ favored user classes
  - ✓ core business processes
  - ✓ demanded for regulatory compliance
- Define priority categories unambiguously.
- Allocate requirements or features to releases.
- Analytically prioritize discretionary requirements.

Software Requirements: 10 Traps to Avoid 10 *Copyright © 2001 by Karl E. Wiegers*

Trap #5: Building Functionality No One Uses

Symptoms

- Users demand certain features, then no one uses them.
- Proposed functionality isn't related to business tasks.
- Developers add functions because "the users will love this".
- Customers don't distinguish "chrome" from "steel".

Trap #5: Building Functionality No One Uses

Solutions

- Derive functional requirements from use cases.
- Trace every functional requirement back to its origin.
- Identify user classes who will benefit from each feature.
- Analytically prioritize requirements, use cases, or features.
  - ✓ have customers rate value (benefit and penalty)
  - ✓ have developers estimate cost and risk
  - ✓ avoid requirements with high cost and low value

Trap #6: Analysis Paralysis

Symptoms

- Requirements development seems to go on forever.
- New versions of the SRS are continually released.
- Requirements are never baselined.
- All requirements are modeled six ways from Sunday.
- Design and coding can’t start until the SRS is perfect.

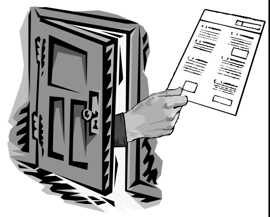
Trap #6: Analysis Paralysis

Solutions

- Remember: the product is software, not an SRS.
- Select an appropriate development life cycle.
  - ✓ staged release, evolutionary prototyping, time-boxing
- Decide when requirements are good enough.
  - ✓ acceptable risk of proceeding with construction
  - ✓ reviewed by analyst, developers, testers, and customers
- Model just the complex or uncertain parts of the system.
- Don’t include final user interface designs in SRS.

Trap #7: Scope Creep

Symptoms



- New requirements are continually added.
  - ✓ schedule doesn't change
  - ✓ no more resources provided
- Product scope is never clearly defined.
- Requirement changes sneak in through the back door.
- Proposed requirements come, and go, and come back.
- Scope issues are debated during SRS reviews.
- Sign-off is just a game.

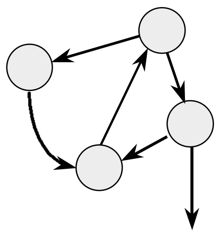
Trap #7: Scope Creep

Solutions

- Determine root causes of the scope creep.
- Document the product's vision and scope.
- Define system boundaries and interfaces.
- Follow the change control process for ***all*** changes.
- Improve requirements elicitation methods.
- Follow a meaningful baselining process.
- Renegotiate commitments when requirements change.

Trap #8: Inadequate Change Process

Symptoms



- No change process is defined.
- Some people bypass the change process.
  - ✓ talk to buddies on the inside
  - ✓ implement rejected changes
  - ✓ work is done on proposed changes before they're approved
- New functionality becomes evident during testing.
- Unclear change request status.
- Changes aren't communicated to all those affected.
- It's not clear who makes change decisions.

Software Requirements: 10 Traps to Avoid 17 *Copyright © 2001 by Karl E. Wiegers*

Trap #8: Inadequate Change Process

Solutions

- Define a practical change control process.
- Set up a Change Control Board.
  - ✓ diverse group
  - ✓ makes binding change decisions
- Use a tool to collect, track, and communicate changes.
  - ✓ problem or issue tracking tools work well
  - ✓ a tool is not a process!
- Establish and enforce change control policies.
- Compare priorities against remaining requirements.

Software Requirements: 10 Traps to Avoid 18 *Copyright © 2001 by Karl E. Wiegers*

Trap #9: Insufficient Change Impact Analysis

Symptoms

- People agree to make changes hastily.
- Change is more complex than anticipated.
- Change takes longer than promised.
- Change isn't technically feasible.
- Change causes project to slip.
- Developers keep finding more system components affected by the change.

Trap #9: Insufficient Change Impact Analysis

Solutions

- Systematically analyze the impact of each proposed change.
  - ✓ identify all possible tasks
  - ✓ consider other implications of accepting the change
  - ✓ estimate effort and schedule impact
- Use requirements traceability information.
  - ✓ identify all affected system components
- Estimate costs and benefits before making commitments.

Trap #10: Inadequate Version Control

Symptoms

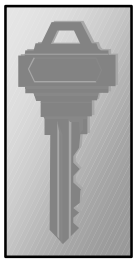
- Accepted changes aren't incorporated into SRS.
- You can't distinguish different SRS versions.
  - ✓ different versions have the same date
  - ✓ identical documents have different dates
- People work from different SRS versions.
  - ✓ implement canceled features
  - ✓ test against the wrong requirements
- Change history and earlier document versions are lost.

Trap #10: Inadequate Version Control

Solutions

- Merge changes into the SRS.
- Adopt a versioning scheme for documents.
- Place requirements documents under version control.
  - ✓ restrict read/write access
  - ✓ make current versions available read-only to all
- Communicate revisions to all who are affected.
- Use a requirements management tool.
  - ✓ record complete history of every requirement change.
  - ✓ SRS becomes a report from the database

Keys to Excellent Software Requirements



- Educated developers, managers, and customers
- A collaborative customer-developer partnership
- Understanding different kinds of requirements
- Iterative, incremental requirements development
- Standard requirements document templates
- Formal and informal requirements reviews
- Writing test cases against requirements
- Analytical requirements prioritization
- Practical, effective change management

Software Requirements: 10 Traps to Avoid 23 *Copyright © 2001 by Karl E. Wiegers*

Requirements References

Carnegie Mellon University/Software Engineering Institute. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, Mass.: Addison-Wesley, 1995.

Davis, Alan M. *Software Requirements: Objects, Functions, and States*. Englewood Cliffs, N.J.: PTR Prentice-Hall, 1993.

Gause, Donald C., and Gerald M. Weinberg. *Exploring Requirements: Quality Before Design*. New York: Dorset House Publishing, 1989.

IEEE Std. 830-1998, "Recommended Practice for Software Requirements Specifications." Los Alamitos, Ca.: IEEE Computer Society Press, 1998.

Leffingwell, Dean, and Don Widrig. *Managing Software Requirements*. Reading, Mass.: Addison Wesley Longman, 2000.

Sommerville, Ian, and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. New York: John Wiley & Sons, 1997.

Wiegers, Karl E. *Creating a Software Engineering Culture*. New York: Dorset House Publishing, 1996.

Wiegers, Karl E. *Software Requirements*. Redmond, Wash.: Microsoft Press, 1999.

Software Requirements: 10 Traps to Avoid 24 *Copyright © 2001 by Karl E. Wiegers*