Software Requirements

Andy Podgurski
EECS Department
Case Western Reserve University

Software Requirements

- *Properties* a software product must exhibit to meet the needs of *users*
- Described in a *software requirements specification* (SRS) document
- Derived and specified *collaboratively* by “customer” and developers
- The process of developing requirements is called *requirements engineering*

Consumers of Requirements

- Project sponsors
- Managers
- Users
- Designers
- Programmers
- Testers
- Maintainers
- Trainees
- Documentation writers
- Lawyers

Functional Requirements

- Describe the *functionality* that a product must provide to users
- Written primarily in *natural language*
- Refer to concepts from the *problem domain*
  - Must be precisely defined
- Written requirements are typically *open to interpretation*
- *Ambiguity* and *incompleteness* must be *resolved* prior to design
- A complex system may have hundreds or thousands of requirements

Example: Functional Requirements for a Word Processor

- *The user shall be able to select a region of text and either cut (delete) it from the document, in which case it is copied to the clipboard, or simply copy it to the clipboard without affecting the selected region.*
- *The user shall be able to locate the insertion point at any location in the document and insert (paste) the contents of the clipboard at that location.*
- *The contents of the clipboard shall be retained until the next cut or copy operation.*

Example: Functional Requirements for a Computer Physician Order Entry System

- *The CPOES shall permit a patient's physician to order the administration of any drug available from the hospital pharmacy and to specify the dosage, frequency, and duration.*
- *The CPOES shall provide support for checking prescribed drug dosages for safety and efficacy, based on the patient's diagnosis, age, weight, and other relevant clinical factors.*
- *The CPOES shall provide support for standard drug interaction checking.*

Non-Functional Requirements

- *All other criteria* a product must satisfy to meet the needs of users, including *quality attributes*
- Types of non-functional requirements:
  - Usability
  - Reliability
  - Availability for use
  - Efficiency/performance
  - Security
  - External interface requirements
  - Constraints on the hardware or software platform
  - Portability to different platforms
  - Compliance with regulations and standards
  - Certification for use

Example: Non-functional Requirements for a Word Processor

- *After completing the product tutorial, 80% of trial users will successfully complete the benchmark editing tasks within 1 hour.*
- *Latency of editor response to keyboard or GUI inputs will not exceed 100 milliseconds.*
- *The contents of the edited document shall be corrupted in no more than 1 in 10,000 user editing sessions on average.*

Example: Some non-functional requirements for a CPOES

- *The CPOES shall be unavailable for use no more than 10 minutes per week on average.*
- *The CPOES shall permit only authorized physicians and other authorized users with prescriptive privileges to enter orders.*
- *The CPOES shall satisfy the ONC HIT Certification Criteria**

*www.healthit.gov/policy-researchers-implementers/standards-and-certification-regulations

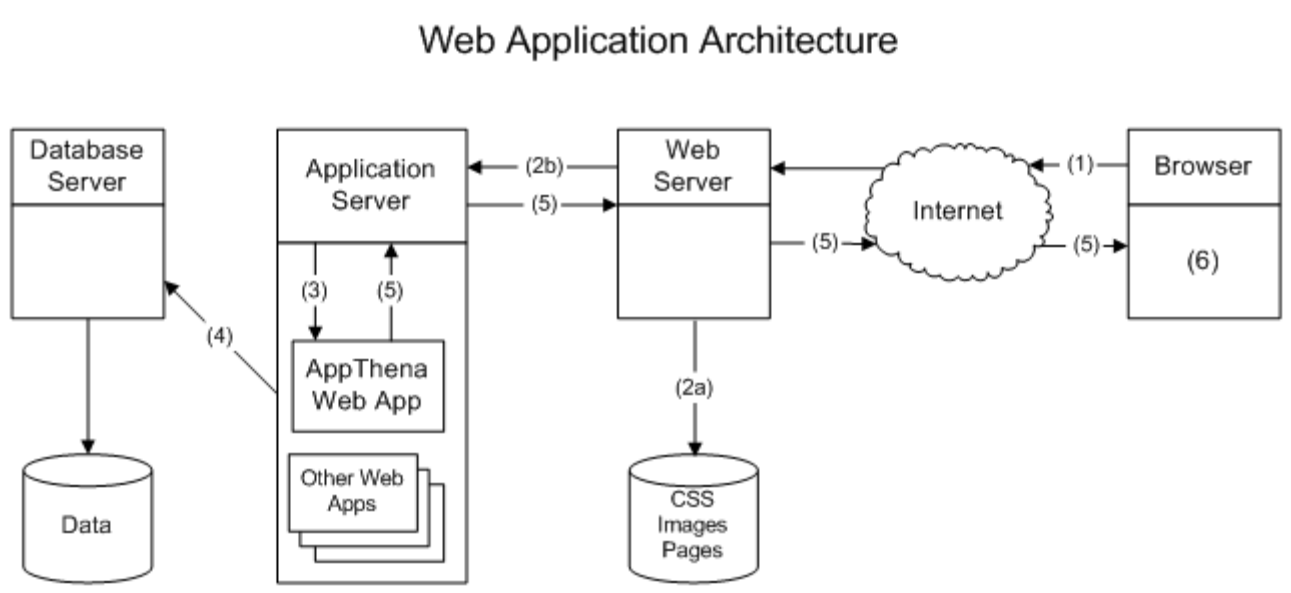
Requirements vs. Design

- It is considered good practice to *exclude* software design and implementation details from an SRS
  - "Requirements should state *what* software is supposed to do, but not *how* it is supposed to do it."
  - This avoids imposing *unecessary constraints* on designers and implementers
- However, if requirements are *too abstract* they may be difficult to understand

Example: Requirements for a Music Store Web Application

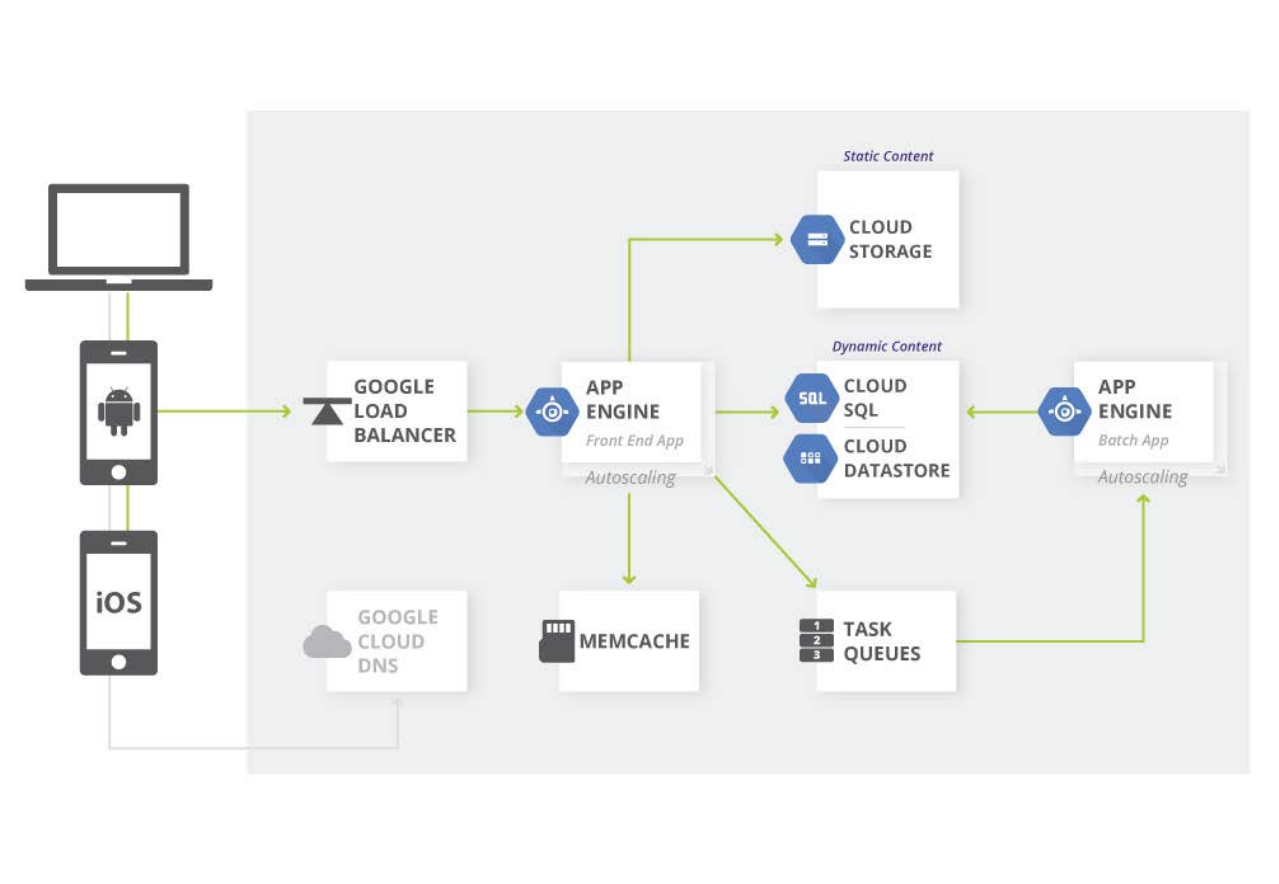
- It's possible to state these very abstractly
- However, assuming a web-app *architecture* permits more *concrete* requirements to be stated
  - Helps to make requirements more *understandable*

“Typical” Web-Application Architecure



codecondo.com/web-application-architecture

Web Application on Google App Engine



cloud.google.com/solutions/architecture/webapp

Requirements and Design Patterns

- A *design pattern* is a reusable solution to a recurring problem in software design
- Kovitz: requirements should be based on an *architectural design pattern*
  - A choice of design pattern *raises specific questions* whose answers clarify many of the product's requirements

Example: Some Questions Raised by Use of Typical Web Application Architecture

- What services should the application provide to users?
- What should the interface to those services look like?
- How should services be mapped to web pages?
- What should the page(s) corresponding to each service look like?
- How many service requests must be handled normally and during peak usage?
- What response times are acceptable?
- What kinds of application failures and security violations are possible?
- What are their severity levels?
- What frequency of occurrence is tolerable for each kind?
- What forms of remediation are called for?

Levels of Detail in Requirements Specification

- Different levels of detail are appropriate to different audiences
  - *Sponsors* and *managers* want to know how a product will help their organization achieve its objectives
  - *Users* care about product features
  - *Designers* require much more detail

Business Requirements

- *High-level* requirements
- Directly related to achieving *business objectives*
- Do *not* describe specific software functionality
- Constrained by *business rules*

Example: Business Requirements for Insurance Agency Software

- *Reduce the average time required to process a policy application by 30%.*
- *Reduce the average time required to underwrite a policy by 25%.*
- *Reduce the average time required to process a claim by 35%.*
- *Increase customer retention by 20%.*

User Requirements

- *Describe or constrain* tasks that users need to accomplish
- *Multiple* user requirements may relate to *one* business requirement

Example: User Requirements for Insurance Agency Software

- *Evaluate the risk to the insurer.*
- *Determine the premium rate.*
- *Write a policy covering the applicant.*

Functional and Non-Functional Requirements

- Most *detailed* kinds of requirements
- Describe and constrain functionality a product must provide to *satisfy user requirements*
- *One* user requirement may generate *multiple* functional and nonfunctional requirements

Example: User and Functional Requirements for Insurance Software

- User requirement: *Evaluate the risk to the insurer*
- Corresponding functional requirements:
  - *The system shall input or retrieve the following data for each customer:*
    - *The vehicle make, model, year, and mileage*
    - *The requested coverage level*
    - *The customer's driving record*
    - *The customer's claim history*
    - *The customer's credit history*
  - *The system shall use the ACME scoring algorithm to rate the policy application.*

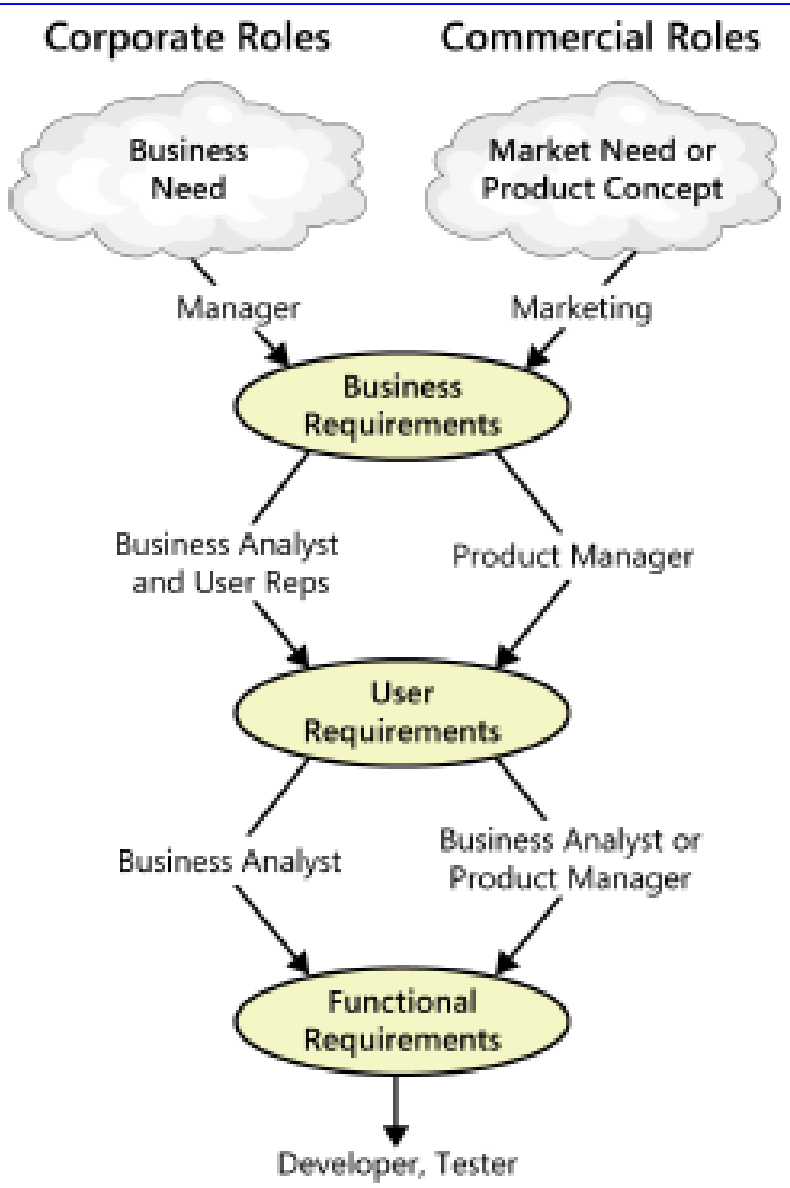


FIGURE 1-3 An example of how different stakeholders participate in requirements development.

From Weigers & Beatty, *Software Requirements*, 2013

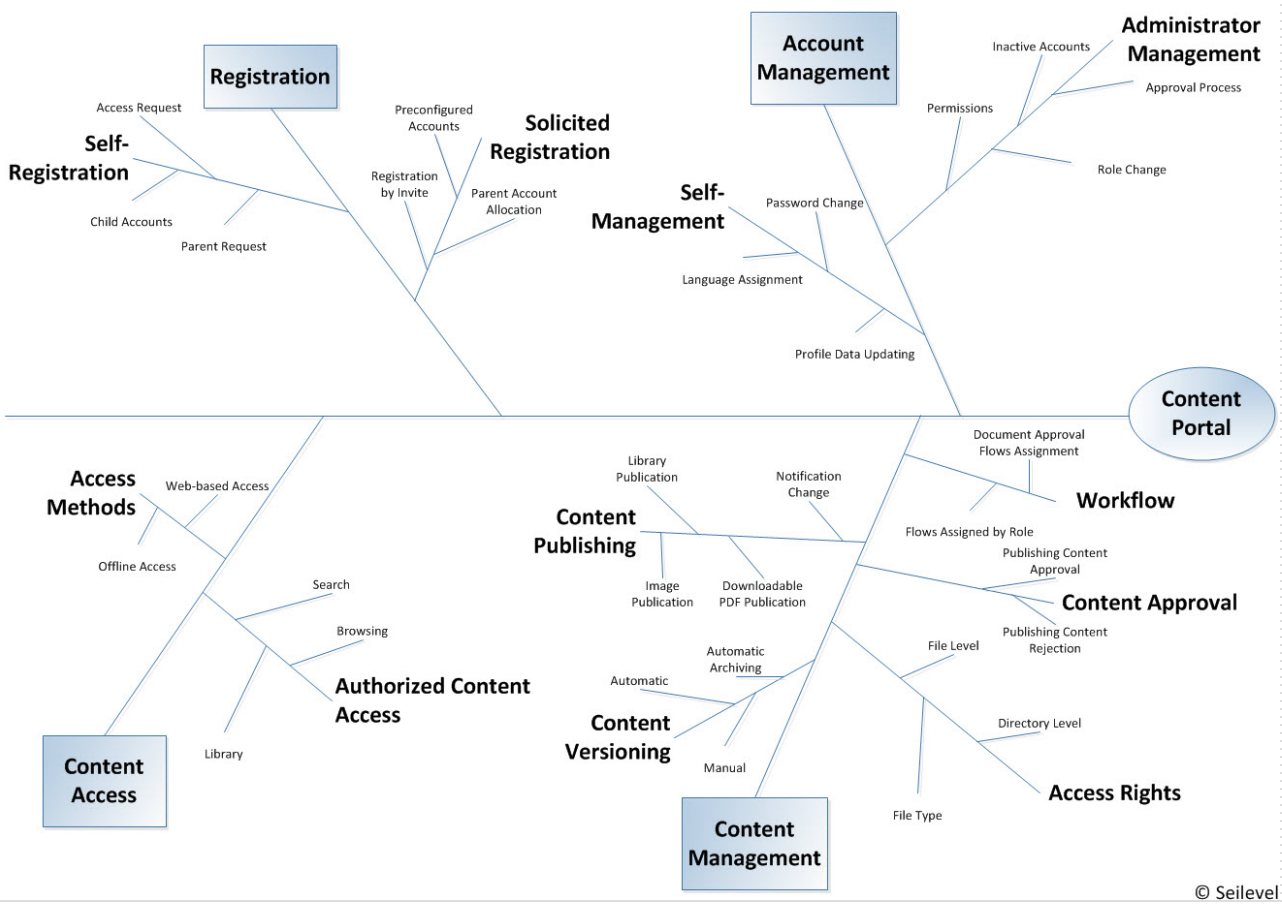
Software Features

- *Groups of logically related functional requirements*
- May influence a customer to *choose* product over others
- May correspond to user requirements
- Emphasized in *marketing*
- Relationships may be depicted graphically in *feature tree*

Example: Features of Insurance Agency Software

- *Tracks Your Clients and Prospects*
- *Generates Policy Renewal Reports*
- *Calculates and Reports Agent Commissions*
- *Synchronizes with ACME Accounting Package*

Feature Tree



www.seilevel.com/ba-resources/rml-requirements-visual-models/feature-tree/

Properties of Good Requirements Specification

- *Complete* – no requirements should be missing
- *Correct* – each requirement must accurately describe a needed unit of functionality
- *Feasible* – it must be possible to implement each requirement given available resources and time

Adapted from *Requirements Engineering* by K. Wiegers

Properties of Good SRS (2)

- *Unambiguous* – each requirement should have one clear and reasonable interpretation
- *Consistent* – requirements should not conflict with each other
- *Validateable* – it must be clear how to verify satisfaction of each requirement

Properties of Good SRS (3)

- *Modifiable* – the SRS should facilitate revision through its structure, labels, table of contents, index, etc.
- *Traceable* – each requirement should be *linked* both *backward* to its origin and *forward* to the design elements and source code that implement it and to the test cases that exercise it

Example SRSs

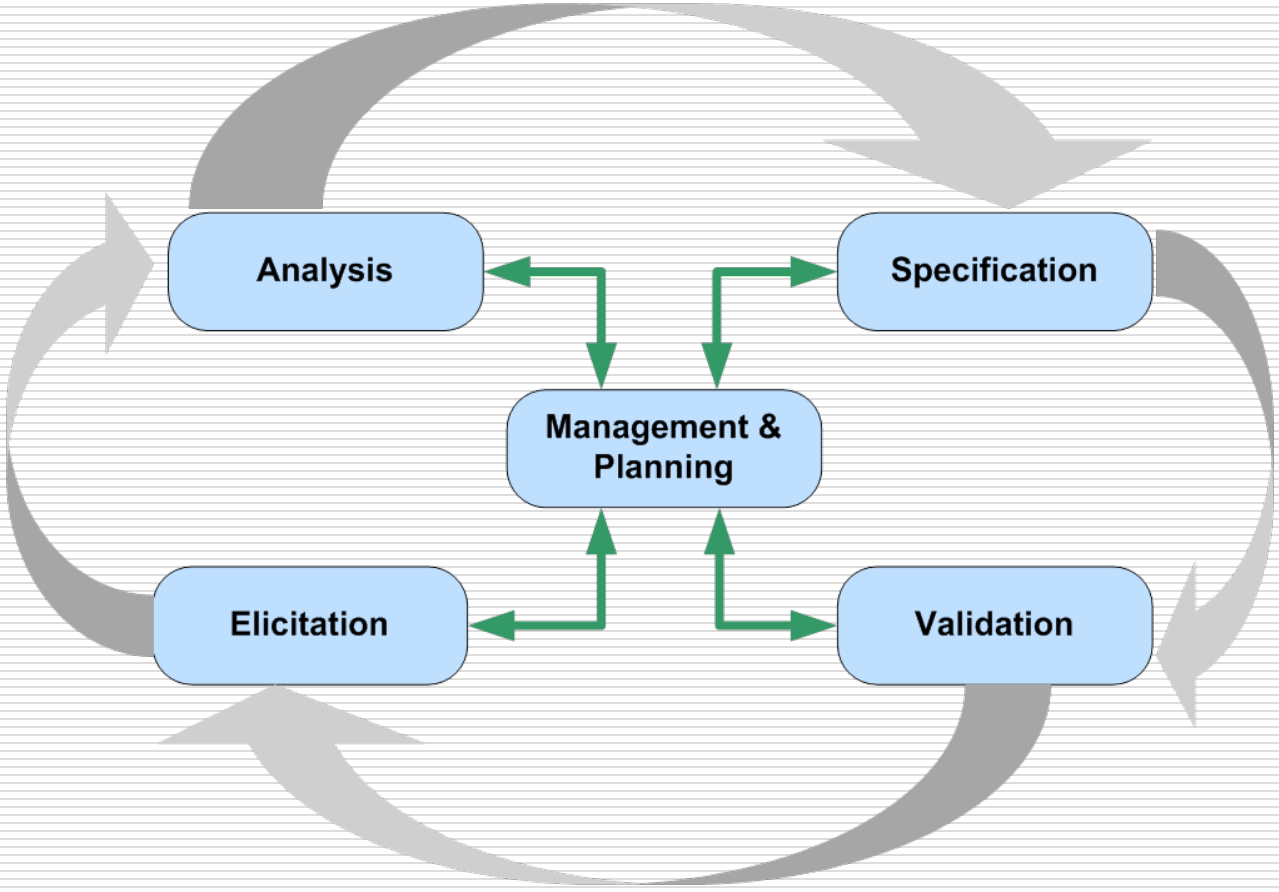
- See *Wiegers' COS SRS* (best general model) and *Vision & Scope* document
- See *Requirements Examples* folder in Canvas

The Requirements Engineering Process

- Identifying the product's expected *user classes* and their *viewpoints* on the system
  - *Eliciting needs* from individuals representing each user class
  - *Understanding* user tasks/goals and the business objectives they align with
  - Helps to achieve *completeness* of SRS
- *Analyzing* information received from users to identify functional and nonfunctional requirements, business rules, etc.
- *Allocating* requirements to top-level components
- *Negotiating* implementation priorities
- *Documenting* requirements (creating SRS and models)
- *Reviewing* documented requirements to ensure common understanding and correct problems

Adapted from *Requirements Engineering* by K. Wiegers

Requirements Engineering Process



Requirements Engineering Activities

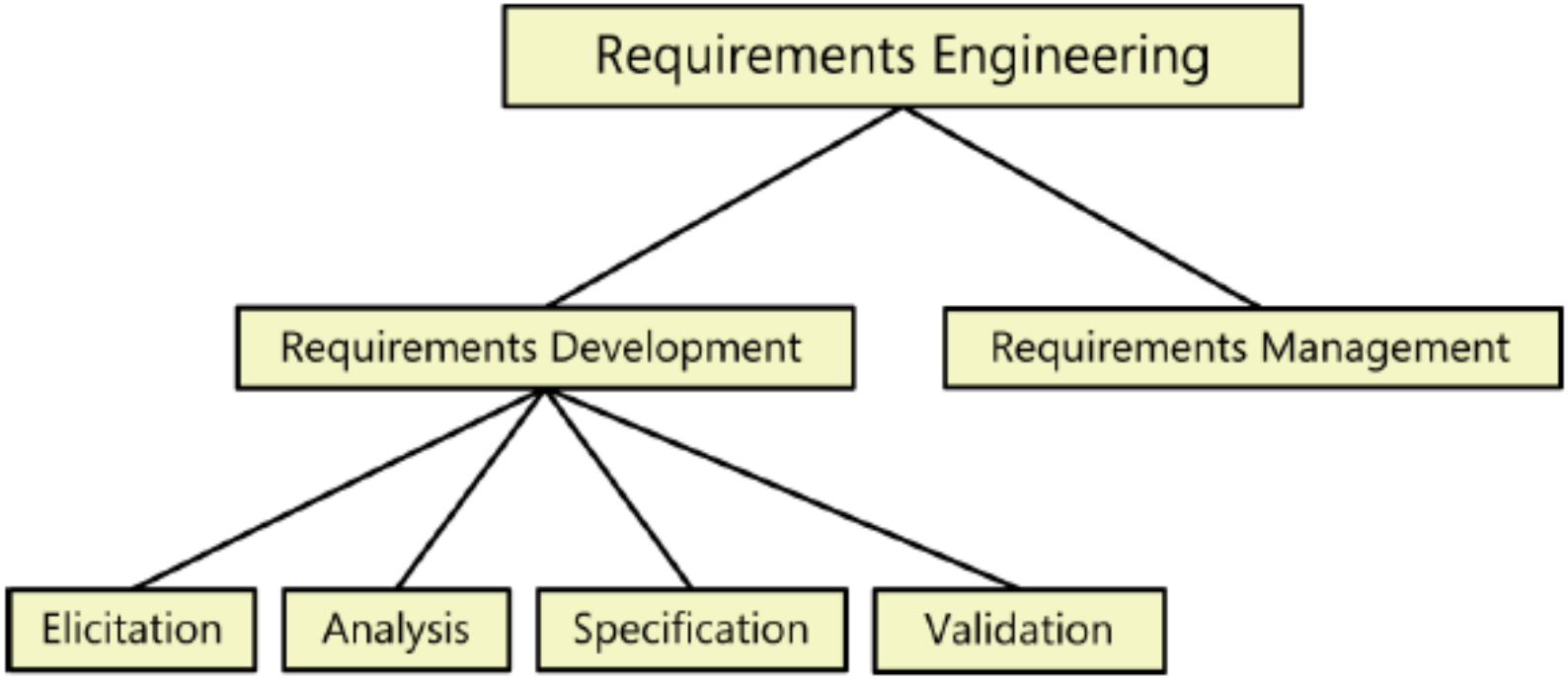


FIGURE 1-4 Subdisciplines of software requirements engineering.

From Weigers & Beatty, *Software Requirements*, 2013

Example: Auto-teller Machine Viewpoints (User Classes)

- Bank customers
- Representatives of other banks
- Hardware and software maintenance engineers
- Marketing department
- Bank managers and counter staff
- Database administrators and security staff
- Communications engineers
- Personnel department

Information Gathering Techniques

- Interviews
- Document analysis
- Surveys
- Customer site visits
- Business process analysis
- Problem lists
- Competitive product analysis
- Reverse engineering of existing systems
- Retrospectives performed on the previous product
- Facilitated requirements workshops

Adapted from *Requirements Engineering* by K. Wiegers

Facilitated Requirements Workshops

- Structured meetings
- Group of *stakeholders* and *content experts* carefully selected
  - They *collaboratively* define, create, and refine deliverables
- Neutral, trained *facilitator*
  - Facilitates communication, decision-making, mutual understanding

From www.construx.com; see also Ellen Gottesdiener, *Requirements by Collaboration*, Addison Wesley, 2002.

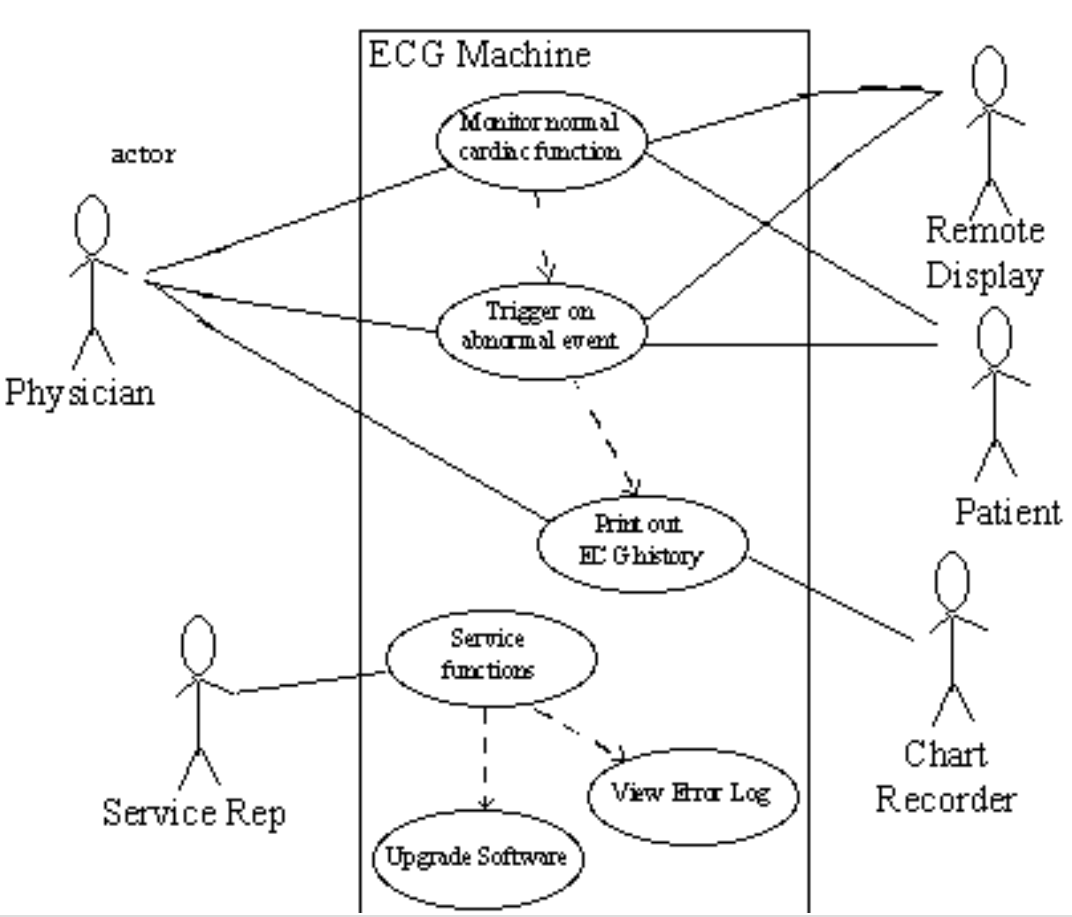
Use Case

- Describes a set of *related usage scenarios* for a software system
- Takes the form of a *sequence of interactions* between a system and an external actor
- An *actor* may be a person, another software system, or a device
- Represented graphically with a *use-case diagram*

Use Case Description

- A unique identifier
- Descriptive name (verb + object)
- A short textual description
- Preconditions
- Postconditions
- Steps
- *Normal course* scenario
- *Alternative courses*

Example: Use-Case Diagram for ECG System



From http://isds.bus.lsu.edu/cvoc/learn/bpr/cprojects/Spring1998/modeling/usecase.html

Use Cases and Functional Requirements

- Use cases *don't* contain all necessary information about requirements
- Functional requirements can be *added* to use cases
- Alternatively, functional requirements *derived* from each use case can be *detailed* in the SRS
- Not all SRSs explicitly describe use cases

User Stories

- Used in *agile* development projects
- *Short* description of feature told from the perspective of user or customer
- Template:
  - *As a <type of user>, I want <some goal> so that <some reason>.*
- They are typically *not expanded* into functional requirements
- Instead, *customers* must be *available* during iterations to provide clarification

Examples of Use Cases & User Stories

TABLE 8-2 Some sample use cases and corresponding user stories

Application	Sample use case	Corresponding user story
Chemical tracking system	Request a Chemical	As a chemist, I want to request a chemical so that I can perform experiments.
Airport check-in kiosk	Check in for a Flight	As a traveler, I want to check in for a flight so that I can fly to my destination.
Accounting system	Create an Invoice	As a small business owner, I want to create an invoice so that I can bill a customer.
Online bookstore	Update Customer Profile	As a customer, I want to update my customer profile so that future purchases are billed to a new credit card number.

From Weigers & Beatty, *Software Requirements*, 2013

Use Cases vs. User Stories

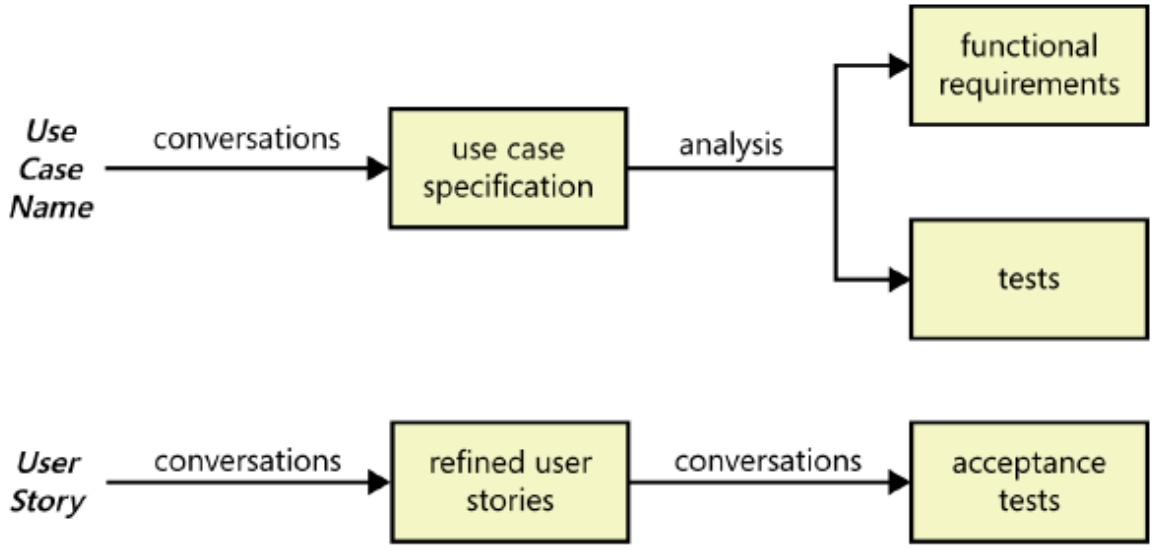


FIGURE 8-1 How user requirements lead to functional requirements and tests with the use case approach and the user story approach.

From Weigers & Beatty, *Software Requirements*, 2013

Graphical Models for Requirements

- Used in requirements analysis & specification to augment text, e.g.,
  - Data flow diagrams
  - Process flow diagrams
  - Object diagrams
  - State transition diagrams
  - Decision tables and trees
  - Feature trees
  - Use-case diagrams
  - Activity diagrams
  - Entity-relationship diagrams

Graphical Models (2)

- Uses of graphical models:
  - Describing *existing* business, process, system, or interface
  - Describing *new* business, process, or system, or interface
- Should *not* be used in SRS to describe *detailed design*

Example: Data Flow Diagram

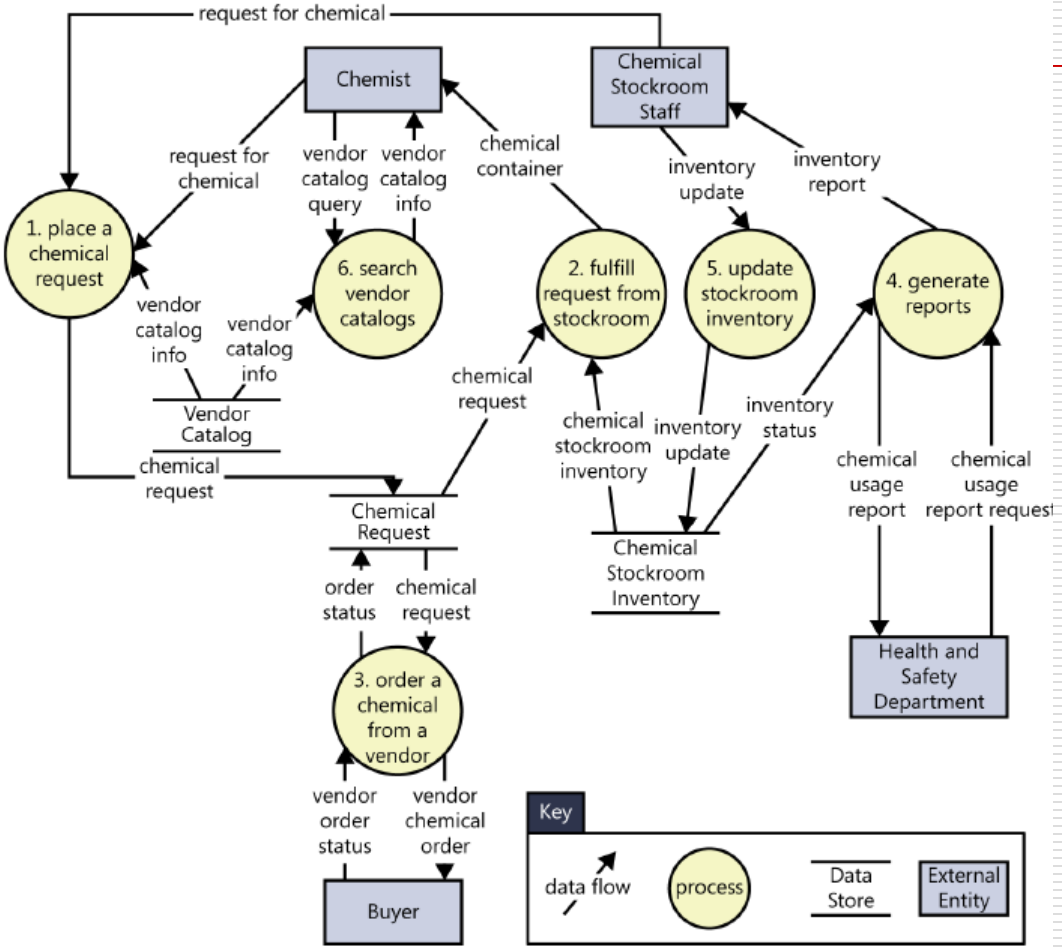


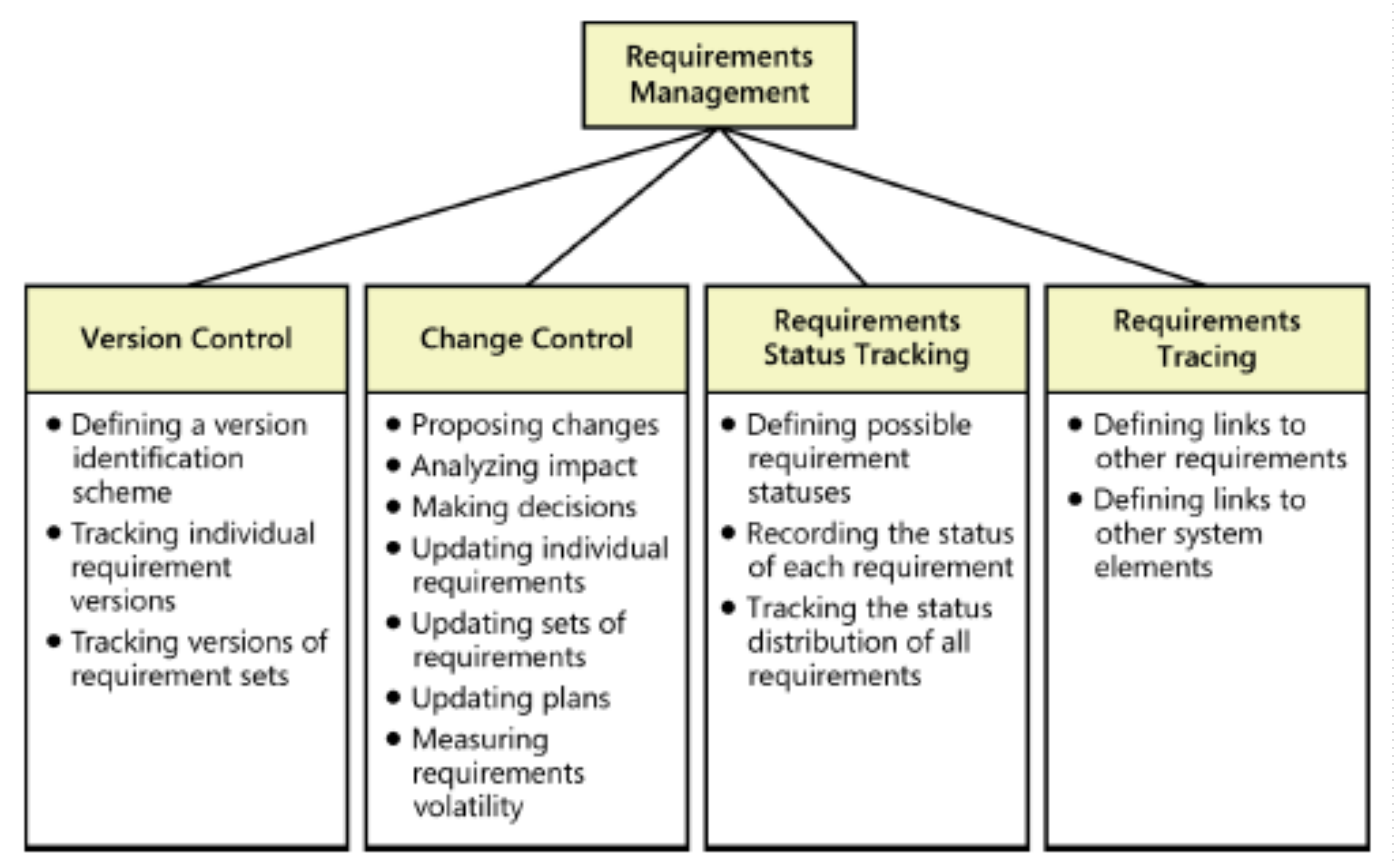
FIGURE 12-1 Partial level 0 data flow diagram for the Chemical Tracking System.

From Weigers & Beatty, *Software Requirements*, 2013

Requirements Management

- Includes all activities that maintain the *integrity*, *accuracy*, and *currency* of requirements throughout project
- The SRS should be updated *whenever* requirements change
- This is facilitated by *traceability* between the SRS, design, and code

Requirements Management Process



From *Software Requirements*, 3rd Edition, by Karl Wiegers and Joy Beatty

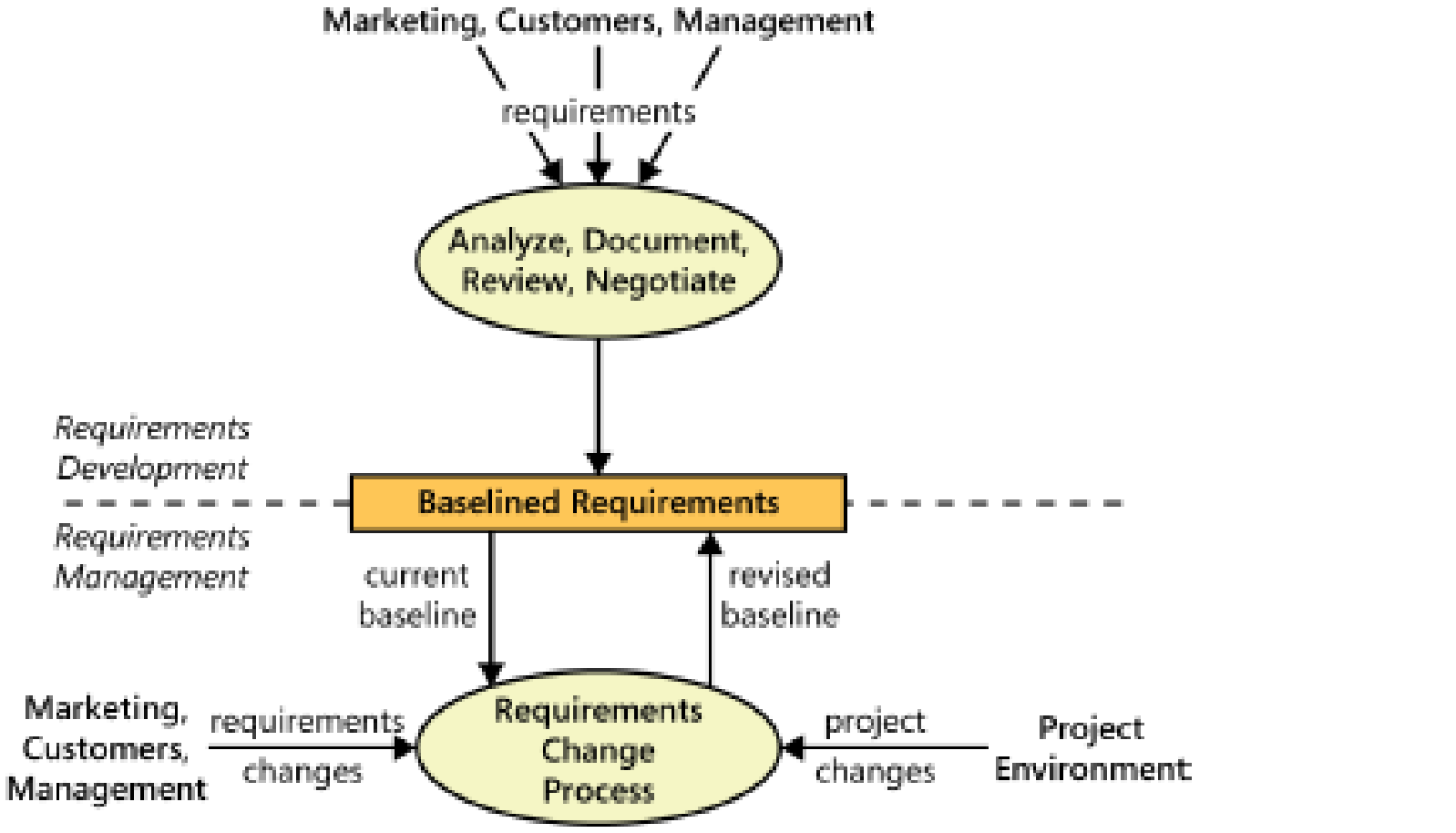


FIGURE 1-5 The boundary between requirements development and requirements management.

From Weigers & Beatty, *Software Requirements*, 2013

Requirements Management Activities

- Define a requirements *change-control process*
- Establish a *change control board*
- Perform requirements *change impact analysis*
- Establish a *baseline and control the versions* of requirements documents
- Maintain a *history* of requirements changes
- Track the *status* of each requirement
- Measure requirements *volatility*
- Use a *requirements management tool*
- Create a requirements *traceability matrix*

From *Requirements Engineering* by K. Wiegers

Requirements Status Tracking

TABLE 27-1 Suggested requirement statuses

Status	Definition
Proposed	The requirement has been requested by an authorized source.
In Progress	A business analyst is actively working on crafting the requirement.
Drafted	The initial version of the requirement has been written.
Approved	The requirement has been analyzed, its impact on the project has been estimated, and it has been allocated to the baseline for a specific release. The key stakeholders have agreed to incorporate the requirement, and the software development group has committed to implement it.
Implemented	The code that implements the requirement has been designed, written, and unit tested. The requirement has been traced to the pertinent design and code elements. The software that implemented the requirement is now ready for testing, review, or other verification.
Verified	The requirement has satisfied its acceptance criteria, meaning that the correct functioning of the implemented requirement has been confirmed. The requirement has been traced to pertinent tests. It is now considered complete.
Deferred	An approved requirement is now planned for implementation in a later release.
Deleted	An approved requirement has been removed from the baseline. Include an explanation of why and by whom the decision was made to delete it.
Rejected	The requirement was proposed but was never approved and is not planned for implementation in any upcoming release. Include an explanation of why and by whom the decision was made to reject it.

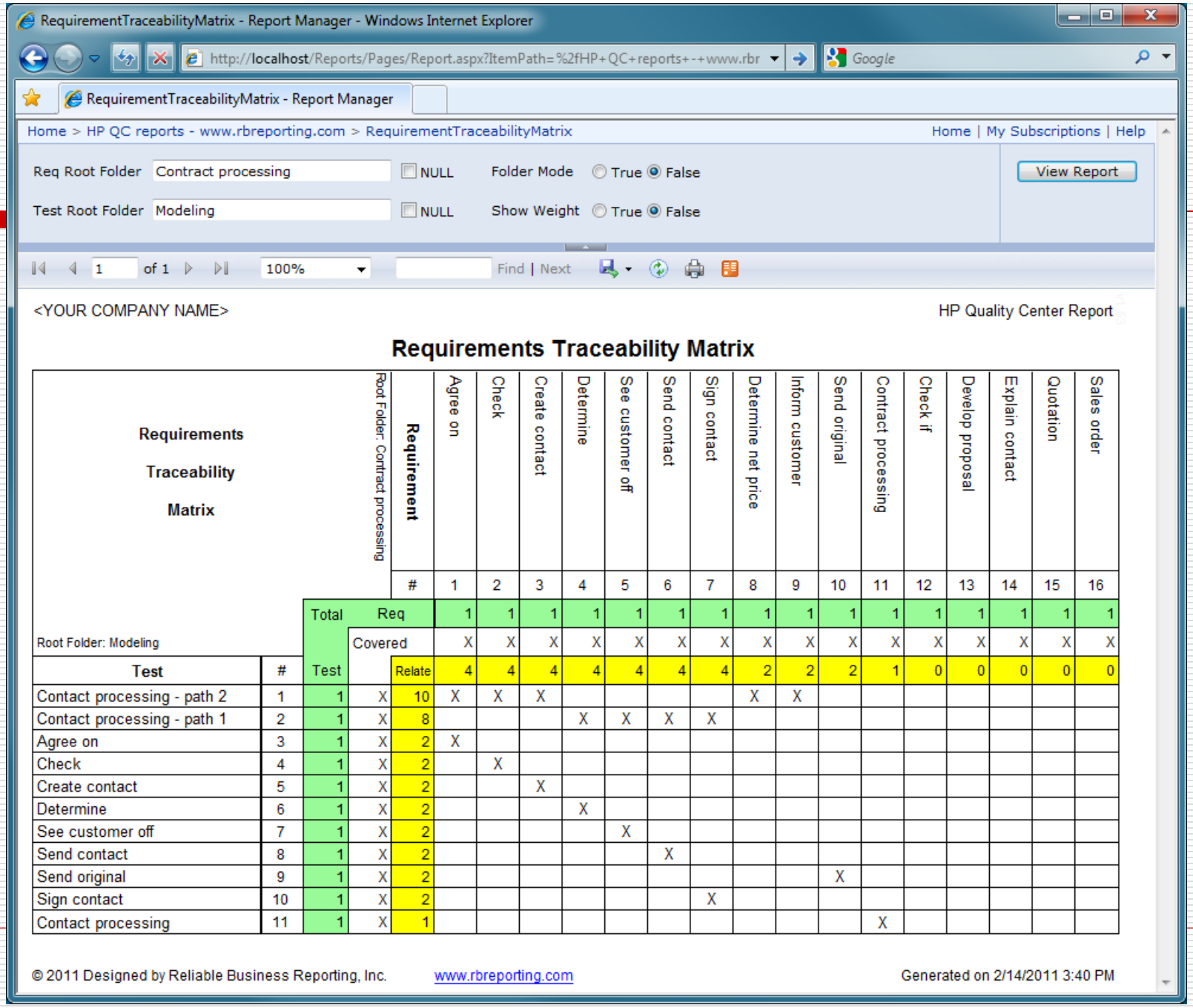
From *Software Requirements*, 3rd Edition, by Karl Wiegers and Joy Beatty

Example Traceability Matrix

Functional Requirement	Implemented in	Tested in
CAT.FR1	../src/cat.py:23	test_plan.sgm:429
CAT.FR2	../src/cat.py:33	test_plan.sgm:438
CAT.FR3	../src/cat.py:53	test_plan.sgm:448
CGI.CHANGE-HTML-TEMPLATES.FR1	../src/html_substitutor.py:20	test_plan.sgm:753
CGI.CHANGE-HTML-TEMPLATES.FR2	../src/html_substitutor.py:45	test_plan.sgm:753
CGI.CHANGE-HTML-TEMPLATES.FR3	../src/html_substitutor.py:48	test_plan.sgm:754
CGI.CHANGE-HTML-TEMPLATES.FR4	../src/html_substitutor.py:54	test_plan.sgm:754
CGI.EDIT.FR1	../src/cgi_switchboard.py:208	test_plan.sgm:998

Note: this example does *not* exhibit backward traceability.

From http://yaktrack.sourceforge.net/yaktrack_docs/a2332.html


RequirementTraceabilityMatrix - Report Manager - Windows Internet Explorer
http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fHP+QC+reports+-+www.rbr
Google
RequirementTraceabilityMatrix - Report Manager
Home > HP QC reports - www.rbreporting.com > RequirementTraceabilityMatrix
Home | My Subscriptions | Help
Req Root Folder
Contract processing
NULL
Folder Mode
True
False
View Report
Test Root Folder
Modeling
NULL
Show Weight
True
False
1
of 1
100%
Find | Next
<YOUR COMPANY NAME>
HP Quality Center Report
Requirements Traceability Matrix
Requirements Traceability Matrix
Root Folder: Contract processing
Requirement
Agree on
Check
Create contact
Determine
See customer off
Send contact
Sign contact
Determine net price
Inform customer
Send original
Contract processing
Check if
Develop proposal
Explain contact
Quotation
Sales order
#
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Total Req
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Root Folder: Modeling
Covered
X X X X X X X X X X X X X X X X
Test
#
Test
Relate
4 4 4 4 4 4 4 2 2 2 1 0 0 0 0 0
Contact processing - path 2 1 1 X 10 X X X X X
Contact processing - path 1 2 1 X 8 X X X X
Agree on 3 1 X 2 X
Check 4 1 X 2 X
Create contact 5 1 X 2 X
Determine 6 1 X 2 X
See customer off 7 1 X 2 X
Send contact 8 1 X 2 X
Send original 9 1 X 2 X
Sign contact 10 1 X 2 X
Contact processing 11 1 X 1 X
© 2011 Designed by Reliable Business Reporting, Inc.
www.rbreporting.com
Generated on 2/14/2011 3:40 PM

REQUIREMENTS TRACEABILITY MATRIX

Project Name:	<optional>
National Center:	<required>
Project Manager Na	<required>
Project Description:	<required>

ID	Assoc ID	Technical Assumption(s) and/or Customer Need(s)	Functional Requirement	Status	Architectural/Design Document	Technical Specification	System Component(s)
001	1.1.1						
002	2.2.2						
003	3.3.3						
004	4.4.4						
005	5.5.5						
006							
007							
008							
009							
010							
011							
012							
013							
014							
015							
016							
017							
018							
019							
020							
021							
022							
023							
024							
025							
026							
027							
028							
029							
030							
031							
032							
033							
034							

www2.cdc.gov/cdcup/library/templates/CDC_UP_Requirements_Traceability_Matrix_Template.xls

Requirements “Bill of Rights” for Customers

1. Expect analysts to speak your language.
2. Expect analysts to learn about your business and your objectives for the system.
3. Expect analysts to structure the information you provide during requirements elicitation into a written SRS.
4. Have analysts explain all work products of the requirements process.
5. Expect analysts and developers to treat you with respect and to maintain a collaborative and professional attitude throughout your interaction.

From *Requirements Engineering* by K. Wiegers

Customer Rights (2)

6. Have analysts and developers provide ideas and alternatives for your requirements
7. Describe characteristics of the product that will make it easy and enjoyable to use.
8. Be given opportunities to adjust your requirements to permit reuse of existing software components.
9. Receive good faith estimates of the costs, impacts, and trade-offs when you request a change in the requirements.
10. Receive a system that meets your functional and quality needs.

Customer "Bill of Responsibilities"

1. Educate analysts and developers about your business and define business jargon.
2. Spend the time needed to provide requirements, clarify them, and iteratively flesh them out.
3. Be specific and precise when providing input about the system's requirements.
4. Make timely decisions about requirements when requested to do so.
5. Respect a developer's assessment of the cost and feasibility of requirements.

Adapted from *Requirements Engineering* by K. Wiegers

Customer Responsibilities (2)

6. In collaboration with the developers, set priorities for functional requirements.
7. Carefully review requirements documents and evaluate prototypes.
8. Communicate changes to the requirements as soon as you know about them.
9. Follow the development organization's process for requesting requirements changes.
10. Respect the process the analysts use for requirements engineering.