Software Engineering

Session 3 – Main Theme
Planning and Managing Requirements
Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

1 Session Overview
2 Planning and Managing Requirements
3 Summary and Conclusion
What is the class about?

- Course description and syllabus:
  - [http://www.nyu.edu/classes/jcf/g22.2440-001/](http://www.nyu.edu/classes/jcf/g22.2440-001/)

- Textbooks:
  - *Software Engineering: A Practitioner’s Approach*
    - Roger S. Pressman
    - McGraw-Hill Higher International

Agenda

- Understanding Requirements
- Requirements Engineering
- Requirements Gathering Guidelines
- Managing the Requirements Engineering Gap
- Distributed Requirements Engineering
- Summary and Conclusion
  - Readings
  - Team Assignment #1
  - Individual Assignment #1
  - Course Project
Agenda – Software Engineering Fundamentals

2 Planning and Managing Requirements
- Understanding Requirements
- Requirements Engineering
- Requirements Gathering Guidelines
- Managing the Requirements Engineering Gap
- Distributed Requirements Engineering

New Features and Functions Start With Requirements Management

I’LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING!
What is a Requirement?

- A “requirement” is a set of measurable user needs and wants negotiated for incorporation into a project or application
  - It is an essential condition; something needed or necessary
  - A requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formal agreement
- As a system is being developed, it is evaluated according to how well it meets its requirements
- Requirement types
  - Business or functional
  - User interface and usability (functional)
  - Hardware (functional)
  - Software (functional)
  - Performance (non-functional)
  - Security (non-functional)
  - Etc.
Why is Developing Software Hard?

“The hardest single part of building a software system is deciding what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.”


Obligatory SE Horror Statistics

From the Standish Group Report (1998):

- 26% of IT projects are successful
- 46% are challenged
- 28% fail
- average cost overrun is 89%
- average schedule overrun is 122%
- 45% of the functions provided in newly developed systems are never used.
Some Typical Problems

- Specifications
  - Requirements not always identified
  - Requirements not always verifiable
  - Allocation to components incomplete
  - Requirements traceability informal

- Poor integration of disciplines

- Lessons are not learned from the past
  - The systems engineers are permanently reinventing the wheel

Sources of Failure

Five of the top reasons:

- Incomplete requirements
- Lack of user involvement
- Unrealistic customer expectations
- Changing requirements and specifications
- Capabilities provided no longer needed.
## Requirement Pitfalls

- Assume requirements are *stated* by the user.
- Customers are confused with users, and visa-versa.
- Dialog (communication) between users and developers is weak.
- Process lacking for managing changes to requirements.
- Developers allowed to fill in the gaps.
- Rush to get through requirements – “Let’s get to the good stuff…”

## Why can’t we get requirements right?

- Mistaking “stated requirements” with the “real requirements.”
- Ambiguous requirements which cannot be validated in the final product.
- Inability to transfer domain expertise.
- Getting requirements from the wrong source(s).
Why can’t we get requirements right?

- The users do not always know what they want!
- Not change, but the inability to effectively manage change.
- Commitment from all parties to be responsible for the success of a project.
- Politics
- Summary: It involves people.

Obstacle to Successful Business Solution Delivery: Business-IT Gap

**Sobering Statistics:**
- 7 out of 10 projects fail
- Of those that fail, 7 out of 10 failures are due to problems in the early stages of deciding on the business requirements, project scope and planning.

**How to ‘Bridge the Gap?’**

**Mind the Gap!**

Successful Solutions depend upon a precise translation from Customer Needs to Business Requirements to Software Engineering Capabilities.
What causes 72% of all major IT projects to fail?

The industry has found three major obstacles:

1. Translation of Business Strategy and Vision into Solution Requirements
2. Coping with Product Choices, Marketing Hype and ever changing Technologies
3. Increasing Complexity of Enterprise Environments

Overcoming Communication Breakdowns During Solution Design

Voice of the Customer
- Why doesn’t it do...
- How do they expect us to use this?
- Look what is possible!

Voice of the Business
- We need a solution to a business problem. How can technology give us a competitive advantage?
- Where’s our requirement? We’re designing in a vacuum!

Voice of the Technologist
- We're designing in a vacuum!
... What happens without translation?

- Premature commitments to solutions are often made
- Business objectives are not communicated well
- Vendors make unsupportable claims and projections regarding their capabilities
- Solution planning takes too long - people get stuck
- Non-optimal solution decisions are made
- Often developers are left to decide for themselves
- Business results are not realized

Envisioning has always been a necessary and difficult job

“Customers never know what they want...”

“... until they see what they get”
Agenda – Software Engineering Fundamentals

2 Planning and Managing Requirements

- Understanding Requirements
- Requirements Engineering
- Requirements Gathering Guidelines
- Managing the Requirements Engineering Gap
- Distributed Requirements Engineering

Requirements Gathering (e.g., Borland CaliberRM Definition Tool)

Business Analyst
- Requirement Engineering
- Requirements management
  - Versioning
  - Tracing
  - Impact analysis
- Automatic Notification
After Requirements, You Design

Effectively generating High Quality Requirements:
- correct
- consistent
- complete
- modifiable
- traceable
- verifiable
- non-ambiguous
- understandable
- annotated

Requirements Engineering

Requirements Development
- Elicitation
- Analysis
- Modeling & Specification
- Verification & Validation

Requirements Management
- Change Control
- Version Control
- Tracing & Impact Analysis
- Status Tracking

Best Practices, Methods, Tools and Processes for Requirements Engineering
### Definitions of Requirements Engineering

- “A systematic process of developing requirements through an iterative, co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained.” [Loucopoulos, 95]

- “Requirements engineering is the branch of software engineering that is concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.” [Zave, 97]

### Traditional RE

- 44%-80% of errors are due to incorrect requirements
- Cost of fixing an error increases exponentially with time
- Phase during which the most critical decisions are made
- Daniel Berry: “Devoting 25% of the system development budget on requirements reduces the cost overrun from ~ 80% to ~ 5%.”
### Project Success Factors (Chaos report)

- User Involvement: 15.9%
- Executive Management Support: 13.9%
- Clear Statement of Requirements: 13.0%
- Proper Planning: 9.6%
- Realistic Expectations: 8.2%
- Smaller Project Milestones: 7.7%
- Competent Staff: 7.2%
- Ownership: 5.3%
- Clear Vision & Objectives: 2.9%
- Hard-Working, Focused Staff: 2.4%
- Other: 13.9%

### Project Challenged Factors (Chaos report)

- Lack of User Input: 12.8%
- Incomplete Requirements & Specifications: 12.3%
- Lack of Executive Support: 7.5%
- Changing Requirements & Specifications: 11.8%
- Technology Incompetence: 7.0%
- Lack of Resources: 6.4%
- Unrealistic Expectations: 5.9%
- Unclear Objectives: 5.3%
- Unrealistic Time Frames: 4.3%
- New Technology: 3.7%
- Other: 23.0%
Project Impaired Factors (Chaos report)

- Incomplete Requirements 13.1%
- Lack of User Involvement 12.4%
- Lack of Resources 10.6%
- Unrealistic Expectations 9.9%
- Lack of Executive Support 9.3%
- Changing Requirements & Specifications 8.7%
- Lack of Planning 8.1%
- Didn't Need It Any Longer 7.5%
- Lack of IT Management 6.2%
- Technology Illiteracy 4.3%
- Other 9.9%

Types of Requirements

```
Requirements
  | Functional requirements |
  | Non-functional requirements |
  | Product requirements |
  | Process requirements |
  | External requirements |
  | Efficiency requirements |
  | Reliability requirements |
  | Portability requirements |
  | Interoperability requirements |
  | Ethical requirements |
  | Usability requirements |
  | Delivery requirements |
  | Implementation requirements |
  | Standards requirements |
  | Legislative requirements |
  | Performance requirements |
  | Space requirements |
  | Privacy requirements |
  | Safety requirements |
```
Use Cases and Requirements

- The use cases are not all of the requirements – because some things don’t fit into scenarios
  - Business Rules (structural facts, computation rules, action restricting rules, action triggering rules)
  - Non-functional Requirements: security, reliability, performance, usability, flexibility (all of the “-ilities”)

- Links to use cases
  - The Business Rules and the Non-functional Requirements are written in text, and they have links to the use cases where they are pertinent
  - Some folks store these items in a spreadsheet or a repository tool (such as Telelogic DOORS or Rational RequisitePro)

Measurable metrics

<table>
<thead>
<tr>
<th>Property</th>
<th>Metric</th>
</tr>
</thead>
</table>
| Performance | 1. Processed transactions per second  
              2. Response time to user input  |
| Reliability | 1. Rate of occurrence of failure  
              2. Mean time to failure       |
| Availability | Probability of failure on demand          |
| Size       | Kbytes                                     |
| Usability  | 1. Time taken to learn 80% of the facilities 
              2. Number of errors made by users in a given time period |
| Robustness | Time to restart after system failure       |
| Portability | Number of target systems                 |
Requirements Engineering Process Activities (1/2)

- **Inception** - ask a set of questions that establish...
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- **Elicitation & Discovery** - elicit requirements from all stakeholders
- **Specification** – capture the requirements in written form

Requirements Engineering Process Activities (2/2)

- **Validation**—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- **Requirements management**
Spiral Model of the RE Process

Inception

- Identify stakeholders
  - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- The first questions
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution
  - Is there another source for the solution that you need?
Eliciting & Discovering Requirements (1/2)

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- an agenda is suggested
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements

Requirements Analysis

- Define **what** the business needs to accomplish
- Define Constraints on **how** a solution is manifested but not on **how** system it is designed
  - *What is accomplished conceptually*
  - *What is required to interface to the system*
  - *What is required to operate it*
Discovering Requirements

“The plan is nothing; the planning is everything”
- Dwight Eisenhower
- Allied supreme commander during World War II
- 34th President of United States (1953-61)

“The discovery is nothing: the discovering (the exploring) is everything.”
“The most difficult part of requirements gathering is not the act of recording what the users want; it is the exploratory, developmental activity of helping users figure out what they want.”

(Steve McConnell)
Sources of Requirements

- Environment for the Software Requirements Definition Process
- Constrained

% of Requirements Gathered from Stakeholders

Unconstrained

On-Line Ordering System
New Corporate Accounting System
Enhancements to Corporate Accounting
Airliner Control System
Missile Guidance System

Domain Expertise

- Domains are subjects that you will need to know something about if you are to build a product.
- Subject domain expertise helps you understand how and why your customers use the kinds of systems that you're developing.
  » (“Work with a User to Think Like a User.”)
- Warning – Domain knowledge acquisition is a nonlinear process!
Domain Expertise

- Ignoring implementation details, what’s your confidence level for developing a system to:
  - Play solitaire
  - Play poker
  - Play bridge

- How about:
  - Paging device
  - Pulse monitor
  - Pacemaker

Eliciting Requirements

- Conscious Requirements: uppermost in the user’s mind – symptomatic of something the user wants to improve.
- Unconscious Requirements: not mentioned since the user assumes everyone else has the same knowledge.
- Undreamed of Requirements: things the user will ask for once they realize what is possible
Quality Function Deployment

- **Function deployment** determines the “value” (as perceived by the customer) of each function required of the system
- **Information deployment** identifies data objects and events
- **Task deployment** examines the behavior of the system
- **Value analysis** determines the relative priority of requirements

Interviews

- Interviewing is more than asking questions. It is a structured technique that helps the requirements engineer or analyst to discuss the system with different stakeholders and build up an understanding of their requirements.
- **Types of interview:**
  - Open/unstructured interviews: There is no predefined agenda and the requirements engineer discusses, in an open-ended way, what stakeholders want from the system.
  - Closed/structured interviews: The requirements engineer looks for answers to a pre-defined set of questions.
Preparing the questions

- Context-free questions:
  - Process
  - Product
  - Meta-questions
- Open- / closed-ended questions

Context-free process questions (1/2)

- Relate to the process, rather than the product
- Examples:
  - Who is the client? Who is the user?
  - What is a highly successful solution really worth?
  - What is the real reason for wanting to solve this problem?
  - How much time do we have for this project?
  - What is the trade-off between time and value?
  - Where else can the solution to this problem be obtained?
Context-free product questions (2/2)

- Relate to the product, rather than the process
- Examples:
  - What problems will the system solve?
  - What problems could this system create?
  - What environment is this system likely to encounter?
  - What kind of precision is required or desired in the product?

Meta-questions

- These are questions about the questions asked
- Examples:
  - Am I asking you too many questions?
  - Do my questions seem relevant?
  - Are you the right person to answer these questions? Are your answers official?
  - Is there anyone else who I should be talking to?
  - Is there anything else I should be asking you?
  - Is there anything you want to ask me?
  - Can I return or call you with more questions later?
Open-ended questions

- Encourages unconstrained answers that provide a large amount of information
- Can be used when you don’t have a good understanding of the problem or domain
- Examples:
  - What problems do you currently experience in your work?
  - What kind of things would you like to change in the way you work?
  - What type of reports would you like to see?

Closed-ended questions

- Used when you want a precise answer to a question
- Examples:
  - How often should the XYZ report be generated?
  - What fields do you think should be protected?
  - What kind of error should be generated if the user types in bad data?
### Scenarios / Use Cases

- Scenarios are stories which explain how a system might be used. They should include:
  - a description of the **system state before** entering the scenario
  - **normal flow** of events in the scenario
  - **alternative flows** of events in the scenario
  - **exceptions** to the normal flow of events
  - information about **concurrent activities**
  - a description of the **system state at the end** of the scenario

- Scenarios are examples of interaction sessions which describe how a user or other system interacts with a system.

- Discovering scenarios exposes possible system interactions and reveals system facilities which may be required.

### Types of prototyping

- **Throw-away prototyping**
  - prototype only serves the elicitation of the user reaction
  - creation of prototype must be rapid, otherwise too expensive

- **Evolutionary prototyping**
  - prototype altered to incorporate design changes
  - eventually becomes the final product

- **Incremental prototyping**
  - product built as separate components / modules
  - each component prototyped and tested, then added to the final system
Requirements analysis and negotiation

Requirements analysis

- Necessity checking
- Consistency and completeness checking
- Feasibility checking

  - Unnecessary requirements
  - Conflicting and incomplete requirements
  - Infeasible requirements

Requirements negotiation

Elicitation Work Products

- A statement of need and feasibility.
- A bounded statement of scope for the system or product.
- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system’s technical environment.
- A list of requirements (preferably organized by function) and the domain constraints that apply to each.
- A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- Any prototypes developed to better define requirements.
The Specification Trap

The Landing Pilot is the Non-Landing Pilot until the ‘decision altitude’ call, when the Handling Non-Landing Pilot hands the handling to the Non-Handling Landing Pilot, unless the latter calls ‘go-around,’ in which case the Handling Non-Landing Pilot continues handling and the Non-Handling Landing Pilot continues non-handling until the next call of ‘land,’ or ‘go-around’ as appropriate. In view of recent confusions over these rules, it was deemed necessary to restate them clearly.

- British Airways memorandum, quoted in Pilot Magazine, December 1996

The Specification Trap

“Some Things Are Better Done than Described”
- Pragmatic Programmer Tip #57

“In the Development of Every Product There Comes a Time When it is Necessary to Shoot the Engineers and Start Production.”
- Anonymous (Taylor Instrument - 1982)
### Quality Requirements

- **Correct** – only user representative can determine
- **Feasible** – get reality check on what can or cannot be done technically or within given cost constraints.
- **Necessary** – trace each requirement back to its origin
- **Unambiguous** – one interpretation
- **Verifiable** – how do you know if the requirement was implemented properly?
- **Prioritized** – function of value provided to the customer

(From “Writing Quality Requirements” – Karl Wiegers)

### Writing Example #1

“The product shall provide status messages at regular intervals not less than every 60 seconds.”
Writing Example #1

“The product shall provide status messages at regular intervals not less than every 60 seconds.”

- Incomplete – What are the status messages and how are they supposed to be displayed?
- Ambiguous – What part of the product? Regular interval?
- Not verifiable

Alternative #1

   1.1. The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds.
   1.2. If background task processing is progressing normally, the percentage of the background task processing that has been completed shall be displayed.
   1.3. A message shall be displayed when the background task is completed.
   1.4. An error message shall be displayed if the background task has stalled.
“The product shall switch between displaying and hiding non-printing characters instantaneously.”

- Not Feasible – computers cannot do anything instantaneously.
- Incomplete – conditions which trigger state switch
- Ambiguous – “non-printing character”
Alternative #2

“The user shall be able to toggle between displaying and hiding all HTML markup tags in the document being edited with the activation of a specific triggering condition.”

- Note that “triggering condition” is left for design

Validating Requirements (1/2)

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?
Validating Requirements (2/2)

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

Verification vs. validation

- Requirements verification works with raw requirements as elicited from the system stakeholders
  - “Have we got the requirements right?” is the key question to be answered at this stage
- Requirements validation works with a final draft of the requirements document
  - “Have we got the right requirements?” is the key question to be answered at this stage
What to look out for (1/3)

- Incomplete lists, typically ending with "etc.", "and/or", and "TBD".
- Vague words and phrases, such as "generally", "normally", "to the greatest extent", and "where practicable".
- Imprecise verbs, such as "supported", "handled", "processed", or "rejected".
- Implied certainty, such as "always", "never", "all", or "every".
- Passive voice, such as "the counter is set." (by whom?)
- Every pronoun, particularly "it" or "its" should have an explicit and unmistakable reference.

What to look out for (2/3)

- Comparatives/superlatives, such as "earliest", "latest", "highest". Words ending in "est" or "er" should be suspect.
- Words whose meanings are subject to different interpretations between the customer and contractor such as:
  - Instantaneous
  - Simultaneous
  - Achievable
  - Complete
  - Finish
  - Degraded
  - A minimum number of
  - Nominal/normal/average
  - Peak/minimum/steady state
  - As required/specified/indicated
  - Coincident/adjacent/synchronous with
What to look out for (3/3)

- Non-quantifiable measures, such as
  - Flexible
  - Modular
  - Efficient
  - Adequate
  - Accomplish
- Some/worst
- Usually/often
- To the extent specified
- To the extent required
- To be compatible/associated with
- Possible (possibly/correct(ly))
- Minimum required/acceptable/reasonable
- Better/higher/faster/less/slower/infrequent

Requirements Management

- The principal concerns of requirements management are:
  - Managing changes to agreed requirements
  - Managing the relationships between requirements
  - Managing the dependencies between the requirements document and other documents produced in the systems engineering process
- Requirements cannot be managed effectively without requirements traceability.
  - A requirement is traceable if you can discover who suggested the requirement, why the requirement exists, what requirements are related to it and how that requirement relates to other information such as systems designs, implementations and user documentation.
### Types of traceability (1/2)

- **Requirements-sources traceability**
  - Links the requirement and the people or documents which specified the requirement

- **Requirements-rationale traceability**
  - Links the requirement with a description of why that requirement has been specified.

- **Requirements-requirements traceability**
  - Links requirements with other requirements which are, in some way, dependent on them. This should be a two-way link (dependants and is-dependent on).

### Types of traceability (2/2)

- **Requirements-architecture traceability**
  - Links requirements with the sub-systems where these requirements are implemented. This is particularly important where sub-systems are being developed by different sub-contractors.

- **Requirements-design traceability**
  - Links requirements with specific hardware or software components in the system which are used to implement the requirement.

- **Requirements-interface traceability**
  - Links requirements with the interfaces of external systems which are used in the provision of the requirements.
A requirements management system

Requirements database

- Req. browser
- Req. query system
- Traceability support system
- Traceability report
- Report generator
- Change control system
- Requirements report
- WP linker
- Req. converter
- NL requirements document

Agenda – Software Engineering Fundamentals

2  Planning and Managing Requirements

- Understanding Requirements
- Requirements Engineering
- Requirements Gathering Guidelines
- Managing the Requirements Engineering Gap
- Distributed Requirements Engineering
"If you don't know how well you are doing, then you know you are not doing very well"

This is not a discussion on how to create software requirements, but how to know you have created good requirements.

The goal of this section is to give you some tips to help you be more successful.

The problem with doing Requirements is that it is difficult to measure their quality, so you can't get feedback.

As a result:

You don't know how well you are doing
You don't know when to stop
Steps to Create Requirements

1. You must be motivated, i.e. want to do it.
2. You must know how to do it.
3. You must create CORRECT requirements.
4. You must create COMPATIBLE requirements.

We will discuss 3 and 4.

1: Correctness

Definition: Software Requirements tell what the system is to do, not how to do it. ‘Correct’ is too broad a term to be useful.

Instead we will look at:

Agreement with the system requirements
Complete
Clear
Feasible
Agree with System Requirements

There is really no such thing as software requirements. There are only system requirements, some of which are implemented with software in a computer.

TIP: Is there a non-trivial system, above the software? If so, Review the Software specification with the System Engineers.

What if No System?

TIP: If there is no larger system, create “Use Cases”, or CONOPS to capture functionality.

TIP: Make sure you know who all the stakeholders are. There are more than you think there are.

TIP: Remember that there are Requirements that the customer doesn’t tell you.
Complete

Compleness means that nothing is missing.

TIP: Use a Requirements Development tool or the suggested format in IEEE 830, to remind you of what should be in the specification.

TIP: Wrong information is better than no information because it motivates people to correct it. So be complete by entering wrong information.

Clear

Not ambiguous and Inter-consistent

TIP: You are the worst person to determine if your specification is ambiguous. Get some one else to read and explain it.

TIP: Never put the same information is two places. Instead use references, or better yet, hypertext format.
Feasible

Vendors love a non-feasible project because they can make more money working on it.

TIP: Create a working model or simulation, if only on paper.

TIP: It never hurts to call the first version, a Feasibility model. There is no upside risk and less downside risk.

2. Compatibility

Definition: Software Requirements must be compatible, that is interrelate within the other parts of the software development project.

Compatibility implies:
- Verifiable
- Traceable
- Modifiable
- Ranked for importance
Verifiable

A Requirement that cannot be tested, is not a Requirement.

TIP: Take a tip from the Agile/XP/Scrum people: create requirements that are in fact the tests that an acceptable system should be able to pass.

Traceable

Traceability is required for Change Management

TIP: Use a software tool or at least number each paragraph in the requirements specification. Then carry that number forward through the remainder of the development. Never change these numbers.
“Software projects change as rapidly as any project ever conceived by the human mind.”
“Requirements change 2% per month”
T. Capers Jones

TIP: Remember software has diseconomy of scale. Design and build the system in parts (GUI, DBM, etc.)

TIP: Because the requirements will change, design and build the system, a portion at a time: Incremental or Spiral development.
TIP: Guarantee there is some success within every budget year.
TIP: This is a big help when requirements change; It is always easier to say ‘later’ than to say ‘no’.
TIP: Decide what is the most important aspect of the system:

- Get to market fast
- Ease of use
- Run in real time
- Be error-free to .999?

TIP: Attach an importance number to every requirement, if only 1...5, or H-M-L. Build just the most important subset of requirements first.

TIP: Remember that risky requirements are, by definition, important.
Ranked for stability

TIP: Attach a stability number to every requirement, if only 1...5, or H-M-L. Isolate the volatile parts in the system and don't spend too much time polishing them.

Other Things to Remember

TIP: Do not put too much into the Requirements specification, i.e. Project information or Design information.
TIP: Remember that the Requirements specification will be used to size the project.
TIP: Alternatives that are almost equally good, will generate the most argument.
Summary

If you do not **know** that you have good requirements, you will not have good requirements.

Make sure they are correct by following the steps here.

Make sure they are compatible by following the steps here.

Resources


Standards Resources

IEEE/EIA 12207.0 “Industry Implementation of International Standard ISO/IEC 12207”
IEEE Std 1220 “Application and Management of the Systems Engineering Process”
IEEE Std 830 “Recommended Practice for Software Requirements Specifications”
IEEE Std 1362 “Concept of Operations Document”
IEEE Std 1540 “Risk Management”
IEEE Std 1012 “Software Verification and Validation”
IEEE Std 730 “Software Quality Assurance Plans”
IEEE Std 828 “Software Configuration Management Plans”

Agenda – Software Engineering Fundamentals

Planning and Managing Requirements
- Understanding Requirements
- Requirements Engineering
- Requirements Gathering Guidelines
- Managing the Requirements Engineering Gap
- Distributed Requirements Engineering
### Need for More Reliable, Concept-driven, Early Requirements Processes

- SDLC Process model frameworks have matured and converged significantly on effective practices in system development phases, however, they are still **weakest on the front-end**.

<table>
<thead>
<tr>
<th>RUP Phases:</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSF Phases:</td>
<td>Envisioning</td>
<td>Planning</td>
<td>Developing</td>
<td>Stabilizing / Deploying</td>
</tr>
</tbody>
</table>

| Less mature methods / support | More mature methods / support |

### Requirements Categories in UP

- **Requirements** are capabilities and conditions to which the system—and more broadly, the project—must conform.
- The UP promotes a set of best practices, one of which is *manage requirements*.
  - In the context of inevitably changing and unclear stakeholder's wishes—"a systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system"
Types of Requirements

- In the UP, requirements are categorized according to the FURPS+ model:
  - **Functional**—features, capabilities, security.
  - **Usability**—human factors, help, documentation.
  - **Reliability**—frequency of failure, recoverability, predictability.
  - **Performance**—response times, throughput, accuracy, availability, resource usage.
  - **Supportability**—adaptability, maintainability, internationalization, configurability.

- The "+" in FURPS+ indicates ancillary and sub-factors, such as:
  - **Implementation**—resource limitations, languages and tools, hardware, ...
  - **Interface**—constraints imposed by interfacing with external systems.
  - **Operations**—system management in its operational setting.
  - **Packaging**
  - **Legal**—licensing and so forth.
Case Study: Use Cases in a Typical Inception Phase

- Not all use cases are written in their fully dressed form during inception.
- The Use-Case Model at this phase of the case study could be detailed as follows:

<table>
<thead>
<tr>
<th>Fully Dressed</th>
<th>Casual</th>
<th>Brief</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Sale</td>
<td>Process Rental</td>
<td>Cash In</td>
</tr>
<tr>
<td>Handle Returns</td>
<td>Analyze Sales Activity</td>
<td>Cash Out</td>
</tr>
<tr>
<td></td>
<td>Manage Security</td>
<td>Manage Users</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start Up</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Shut Down</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage System Tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Other Requirements

- **Supplementary Specification**
  - Captures other kinds of requirements, such as documentation, packaging, supportability, licensing, and so forth.

- **Glossary**
  - Captures terms and definitions; it can also play the role of a data dictionary.

- **Vision**
  - Defines the stakeholders' view of the product to be developed, specified in terms of the stakeholders' key needs and features. Containing an outline of the envisioned core requirements, it provides the contractual basis for the more detailed technical requirements.
### Agile RE

- We cannot determine in advance what the final product looks like.
- The requirements evolve over time.
- Minimum of documentation
- Minimum of RE process
- Traditional RE will produce documents that are out of date while they are still being developed.

### SW developer vs. user

“... Software developers by themselves aren’t very good at designing software that users like. But software developers can be very good at helping users discover what they like, because users by themselves aren’t very good at designing software they like either.”

(Steve McConnell)
Revisited: Definition of RE

- Requirements Engineering is concerned about finding out what the system is all about.
- Depending on the project, it also includes documentation and traceability.

Do Agile Methods and Approaches Fill the Gap?

- Agile approaches recognize that continuous, effective communication among all stakeholders is absolutely central
  - they stress developing an attitude of mutual collaboration and invention
  - represent a significant step forward in the maturity and “realism” of software development methods
- But, Agile approaches also don't offer mature, reliable techniques or assets with which to ensure that this type communication and joint decision making is fostered.
- Bottom line: NO
  - approaches of all kinds recognize the need to address the “envisioning” phase
  - there is definitely room for improvement for reliably facilitating envisioning
### Agile Requirements

- Agile Methodologies (XP, Scrum, FDD,..) are designed to address many of the requirement challenges we have discussed:
  - Access to domain expertise
  - Short, frequent release cycles - reviews
  - Quick prototype solutions (spikes)
  - Test driven design – encourages requirement discovery.
  - Embrace change

### Agile Requirement Tips

1. Expect to gather requirements throughout the entire project - *Embrace Change*.
2. Explain the techniques to all stakeholders
3. Get buy-in from your management and customer.
4. Use the terminology of your users, avoid geek-speak – “Keep the Customer Satisfied”
5. Take a breadth-first approach.
6. Existing documents are a good source of requirements – beware of N.I.H. syndrome
7. Build models and prototypes
8. Get into your application domain!
AM enhances other software processes

<table>
<thead>
<tr>
<th>Agile Modeling (AM)</th>
<th>Your Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Software Process</td>
<td>(XP, UP, DSDM, …)</td>
</tr>
</tbody>
</table>

How is the ‘intended’ system identified in the first place

Addressing the creative or inventive side of solution development requires exploration of:

- ‘What is it possible to be?’
- “What technology enablers should we assess and adopt” (in other words the ‘system how’)
- “How to encourage and support the discovery processes that occur when ‘solutions are in search of problems’?
- “How to deal with emerging technologies?” (how to realize their business value and gain competitive advantage)

“Software development is a game of invention and communication.”
Alistair Cockburn
Importance of Modeling the Problem Situation

- Why model problems?
  - Understanding of cause and effect
  - Surfacing assumptions
  - Determining scope
  - Understanding systemic nature of a world

“A problem well stated is a problem half solved”

Companies need to align IT projects to their Business Strategy

Key stakeholders include:

- End-users who will have to live with the solution
- The Enterprise that is hoping to realize a business ROI
- IT departments or other technical organizations responsible for implementing the solution
- Technology vendors whose technology provides underpinnings and components of the solution

Stakeholders often have conflicting objectives.
**Net Effect of “Emerging Technologies” and “Component Solutions”**

Emerging Technologies and Component-Based solutions are revolutionizing “Business Capabilities”

- System development is no longer “green-field” development, but is now often more about:
  - configuring and integrating components,
  - innovatively leverage pre-built solution capabilities
  - creatively using new technologies.
- Methods are needed to help companies move from:
  - strategy to capability design
  - technology assessment to selection
  - Technology selection to solution roadmaps
- Tools are important
  - for decision making with consensus building, solution envisioning & design, strategic business case to project planning and project execution

**Repeatable means to mediate strategic communication and requirements**

Candidate Approaches (illustrative, partial list):

- Business Service Patterns
- Business Centric Methodology
- Business Capabilities Design
- Capability Cases (note, members of the organizing committee will present an approach based on these)
- Agile Enterprise Architecture Models
- …
Distributed RE

- More and more distributed teams
- Cost savings
- Time savings
- Can “get in” experts quickly
- HW cannot always travel
- Possible with new communication media:
  - Video conferencing
  - Groupware tools
Ocker, Hiltz, Turoff, Fjermestad:
- 41 groups of 5 participants each
- Face-2-face vs. distributed, asynchronous
- Distributed teams produced:
  - marginally more creative designs
  - higher-quality designs

Damian:
- 15 groups of 4 people
- Face-2-face vs. 4 distributed settings
- Used facilitation
- One of the distributed settings produced the best results
Advantages of Distributed Development: Productivity

- Aoyama:
  - A distributed concurrent development process replaced a sequential centralized process for a large-scale telecoms software project.
  - Drastic reduction of the development cycle-time

Key Challenges in Distributed RE

- Communication
- Coordination
- Time shift
- Cultural issues
- Misunderstandings
- Knowing people
- Common repository
Example

An embedded system product at Lucent Technologies was developed in two locations: Germany and the UK. This project revealed many problems resulting from the distributed development process. Unit testing and development was plagued by incomplete specifications. Implementation diverged from the design document, and the design document was not updated. Bug reports were generated from out-of-date design documents that identified ill-functioning components. While the components were working as implemented, the tests generated at other sites used design documentation that was never updated after designs changed.

Agenda

1. Session Overview
2. Planning and Managing Requirements
3. Summary and Conclusion
Conclusions (1/2)

- Use well-established RE practices
- Communicate!!
- Have ONE common repository
- Distributed RE is possible!

Conclusions (2/2)

“Everything should be made as simple as possible, but not simpler.”
- Albert Einstein
Course Assignments

- Individual Assignments
  - Reports based on case studies / class presentations
- Project-Related Assignments
  - All assignments (other than the individual assessments) will correspond to milestones in the team project.
  - As the course progresses, students will be applying various methodologies to a project of their choice. The project and related software system should relate to a real-world scenario chosen by each team. The project will consist of inter-related deliverables which are due on a (bi-) weekly basis.
  - There will be only one submission per team per deliverable and all teams must demonstrate their projects to the course instructor.
  - A sample project description and additional details will be available under handouts on the course Web site

Team Project

- Project Logistics
  - Teams will pick their own projects, within certain constraints: for instance, all projects should involve multiple distributed subsystems (e.g., web-based electronic services projects including client, application server, and database tiers). Students will need to come up to speed on whatever programming languages and/or software technologies they choose for their projects - which will not necessarily be covered in class.
  - Students will be required to form themselves into "pairs" of exactly two (2) members each; if there is an odd number of students in the class, then one (1) team of three (3) members will be permitted. There may not be any "pairs" of only one member! The instructor and TA(s) will then assist the pairs in forming "teams", ideally each consisting of two (2) "pairs", possibly three (3) pairs if necessary due to enrollment, but students are encouraged to form their own 2-pair teams in advance. If some students drop the course, any remaining pair or team members may be arbitrarily reassigned to other pairs/teams at the discretion of the instructor (but are strongly encouraged to reform pairs/teams on their own). Students will develop and test their project code together with the other member of their programming pair.
Document Transformation methodology driven approach
- Strategy Alignment Elicitation
  - Equivalent to strategic planning
    - i.e., planning at the level of a project set
- Strategy Alignment Execution
  - Equivalent to project planning + SDLC
    - i.e., planning at the level of individual projects + project implementation

- Build a methodology Wiki & partially implement the enablers
- Apply transformation methodology approach to a sample problem domain for which a business solution must be found
- Final product is a wiki/report that focuses on
  - Methodology / methodology implementation / sample business-driven problem solution

Team Project Approach – Initial Step

- Document sample problem domain and business-driven problem of interest
  - Problem description
  - High-level specification details
  - High-level implementation details
  - Proposed high-level timeline
Assignments & Readings

- Readings
  - Slides and Handouts posted on the course web site
  - Textbook: Part Two-Chapter 5
  - Documentation provided with Rational RequisitePro
- Team Project
  - Team Project proposal (format TBD in class)
  - See Session 2 Handout: “Team Project Specification” (Part 1)
- Team Exercise #1 (continued)
  - Presentation topic proposal (format TBD in class)
- Individual Assignment
  - See Session 3 Handout: “Assignment #1”
- Project Frameworks Setup (ongoing)
  - As per reference provided on the course Web site

References - From Requirements to Models

- Ian Sommerville’s Slide Sets:
  - Computer-based System Engineering - ch2.ppt
  - Project Management - ch3.ppt
  - Requirements Engineering - ch4.ppt
  - Requirements Analysis - ch5.ppt
  - System Models - ch6-1.ppt, ch6-2.ppt
  - Requirements Definition and Specification - ch7-1.ppt, ch7-2.ppt
  - Software Prototyping - ch8.ppt
  - Formal Specification - ch9.ppt
  - Algebraic Specification - ch10-1.ppt, ch10-2.ppt
  - Model-based Specification - ch11.ppt
Any Questions?

Back to Development Life Cycle Model

- **Process Steps**
  - REQUIREMENTS DEFINITION
  - HIGH LEVEL DESIGN
  - DETAIL DESIGN
  - SYSTEM CONSTRUCTION
  - VERIFICATION & VALIDATION
  - SYSTEM DELIVERY

- **Process Gates**
  - REVIEW

- **Prototypes**
  - PROTOTYPE 1
  - PROTOTYPE 2
  - PROTOTYPE 3

- **Project Management Support Processes**
  - Risk Reduction
  - Training
  - Planning
  - Configuration Management
  - Estimating
  - Metrics
  - Quality Assurance
Next Session: Software Engineering Tools Primer