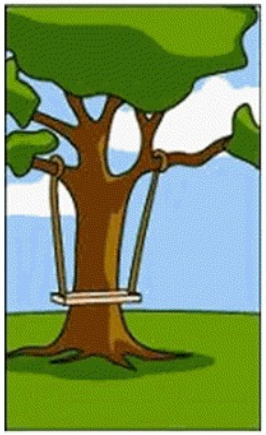
Requirements 1: Overview and Concepts

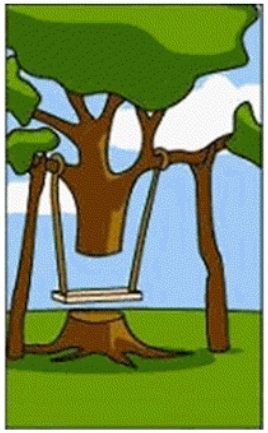
Rohan Padhye and Michael Hilton

institute for SOFTWARE RESEARCH | Carnegie Mellon University School of Computer Science

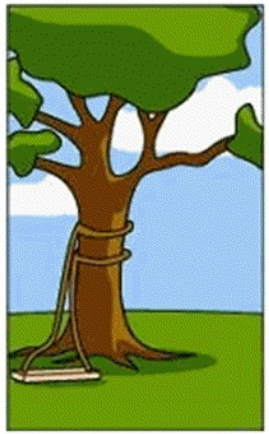
How the customer explained it



How the project leader understood it

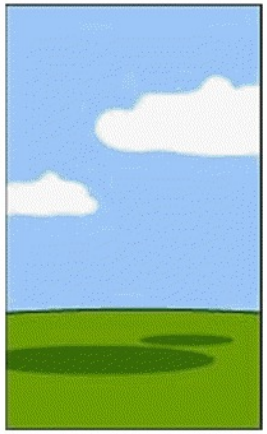


How the engineer designed it

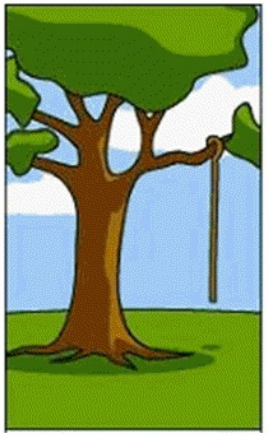


How the programmer wrote it

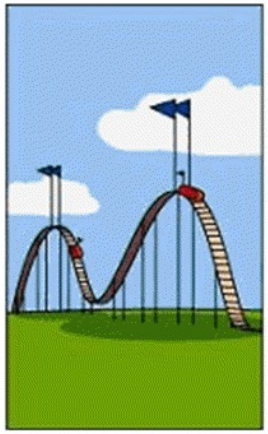
How the sales executive described it



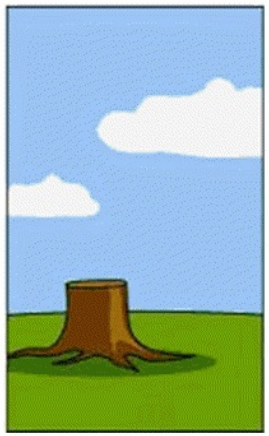
How the project was documented



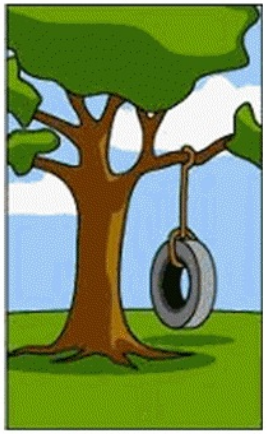
What operations installed



How the customer was billed



How the help desk supported it



What the customer really needed

Administrivia

HW1 returned soon

HW2-

- some teams have 4, we will consider when grading
- You can defer CI till later

MONDAY quiz deadline, interview questions for Josh Gardner

Examples adapted arbitrarily from prior years without identifying information!

REFLECTIONS ON REFLECTIONS

Reflection documents

Shallow reflection examples

[PROCESS]

At our first meeting, we developed an initial outline of our approach. This was followed by preparing a list of tasks which were required for implementing the X system. Next, we divided the tasks among ourselves and came up with a rough timeline of the process to be followed."

[SCHEDULE]

"Although we managed to meet all the milestones and implement all the desired features, the exact dates for the same could not be followed towards the end."

[PLANNING]

"Learning how to use API X took a little longer than expected, which caused a setback of a day; but overall we managed to complete the entire project before the deadline and adhered to the timeline."

[TEAM WORK / COMMUNICATION]

"We all agreed to use tool Y to keep in touch. We used it to announce when we started or completed individual tasks, current milestone statuses.. We also used Y to schedule a group meeting for the integration portion of our coding assignment"

Good reflection examples

[PLANNING / PROCESS]

"Since I was interested in the planning, we decided as a team I would be in charge of documenting our progress.. It worked really well to have one person managing what needed to get done or who needed to do it, and ensuring a shared single vision and set of goals as a group. However, there exist negatives approaching things this way…I found that my teammates sometimes would rely on me too heavily."

[TEAM WORK / COMMUNICATION]

"An example of something that [would] work well is…issue tracking – something I asked them to do since first meeting. It's easy to forget this information over time… If we had simply reminded ourselves on a regular basis, we would have had fewer problems forgetting these things."

[PLANNING]

"People seemed to be annoyed because X "was not doing any work". I believe X did the least amount of work, but we also assigned X the least amount of work. I wonder if this can all be traced back to the fact that X could not attend our first group meeting"

More good examples

[TEAMWORK]

"It helps to say 'thank you' before complaining about a teammate's work. Only take conflict-inducing action if you think it is extremely important and are willing to follow up. Otherwise, you are wasting everyone's time. Would we have treated each other differently if we had known we would be partnered up on more than just this assignment for the class?"

[TEAMWORK]

"two takeaways I had from this project are :

– It is best to present yourself as someone who is willing to help out, and do more than what was originally asked of you. This way, if people decline your offer to help out, they will be okay with the fact that you may not be working as hard as them at that point in time.

– Respect other people's time and work, and take that into consideration when you decide to criticize their work or bring up issues. "

Also

The homework document includes bulleted lists and prose outlining what a “good solution” looks like.

Consider checking your submission against it, at the very least before submitting, if not sooner.

Learning goals

Explain the importance and challenges of requirements in software engineering.

Explain how and why requirements articulate the relationship between a desired system and its environment. Identify assumptions.

Distinguish between and give examples of: functional and quality requirements; informal statements and verifiable requirements.

State quality requirements in measurable ways

Overly simplified definition.

Requirements say what the system will do
(and not how it will do it).

Fred Brooks, on requirements.

The hardest single part of building a software system is deciding precisely ***what to build****.*

No other part of the conceptual work is as difficult as establishing the detailed technical requirements …

No other part of the work so cripples the resulting system if done wrong.

No other part is as difficult to rectify later.

— Fred Brooks

A problem that stands the test of time…

A 1994 survey of 8000 projects at 350 companies found: 31% of projects canceled before completed; 9% of projects delivered on time, within budget in large companies, 16% in small companies.

Similar results reported since.

Causes:

- 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 and specifications (8.7%)
- Lack of planning (8.1%)
- System no longer needed (7.5%) .

WHY IS THIS HARD?

Communication problem

Overall problems

Involved subproblems?

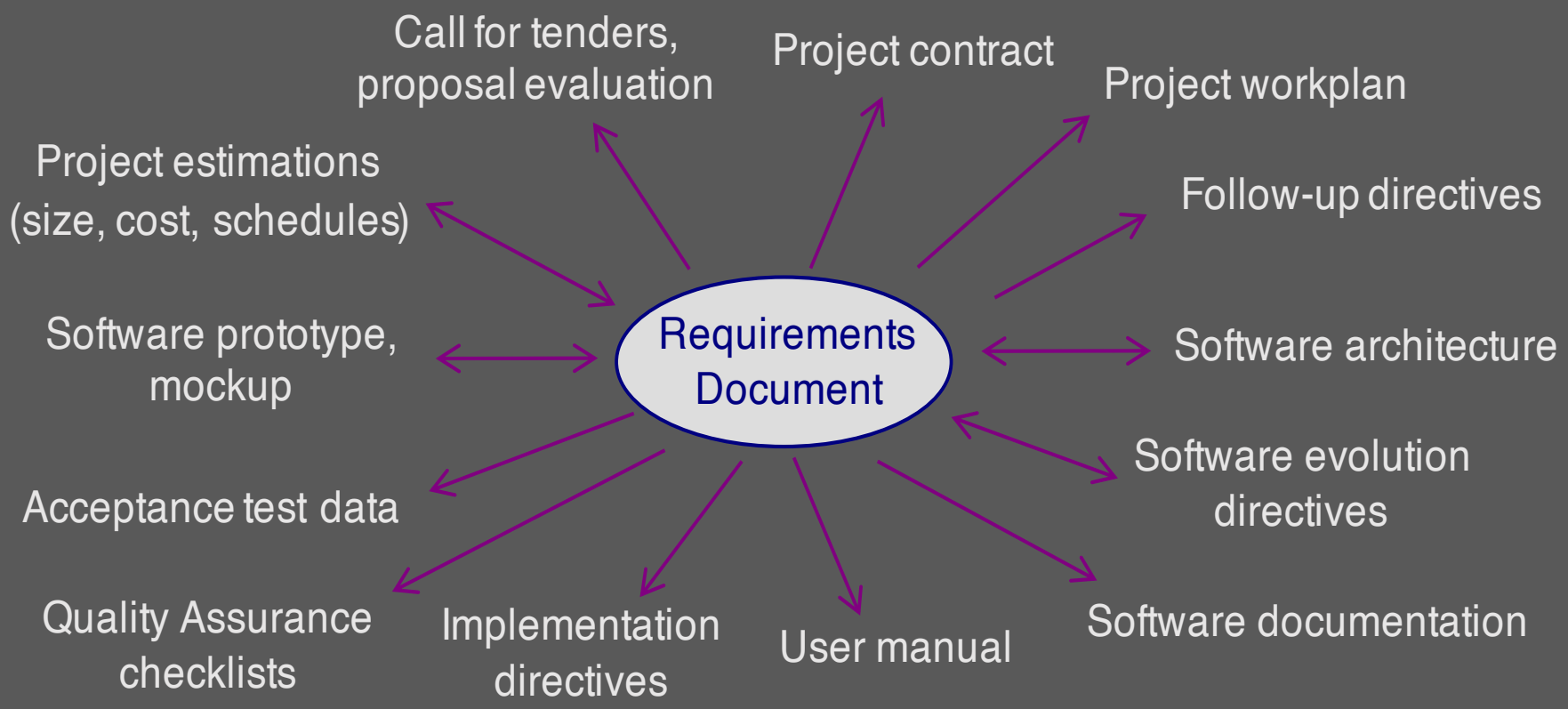
Required functionality?

Nice to have functionality?

Expected qualities?

How fast to deliver at what quality for what price?

Requirements in software projects



What is requirements engineering?

Knowledge **acquisition** – how to capture relevant detail about a system?

- Is the knowledge complete and consistent?

Knowledge **representation** – once captured, how do we express it most effectively?

- Express it for whom?
- Is it received consistently by different people?

You may sometimes see a distinction between the requirements *definition* and the requirements *specification*.

Functional Requirements

What the machine should do

- Input
- Output
- Interface
- Response to events

Criteria:

- Completeness: All requirements are documented
- Consistency: No conflicts between requirements
- Precision: No ambiguity in requirements

Quality/Non-functional requirements

Specify not the functionality of the system, but the quality with which it delivers that functionality.

Can be more critical than functional requirements

- Can work around missing functionality
- Low-quality system may be unusable

(We'll come back to these in a bit.)

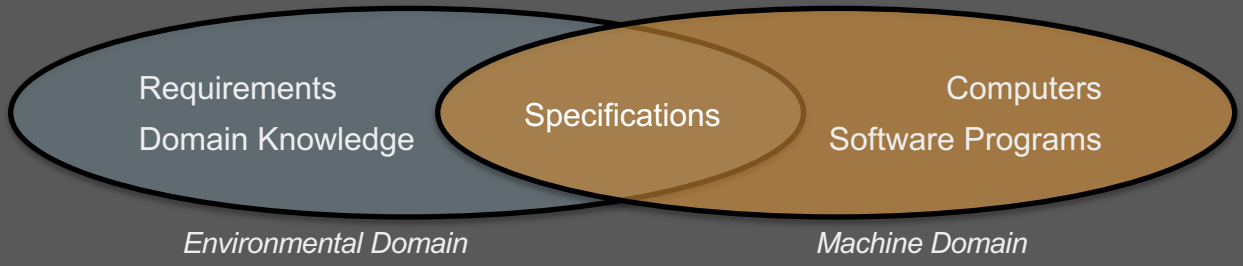
Functional requirements and implementation bias

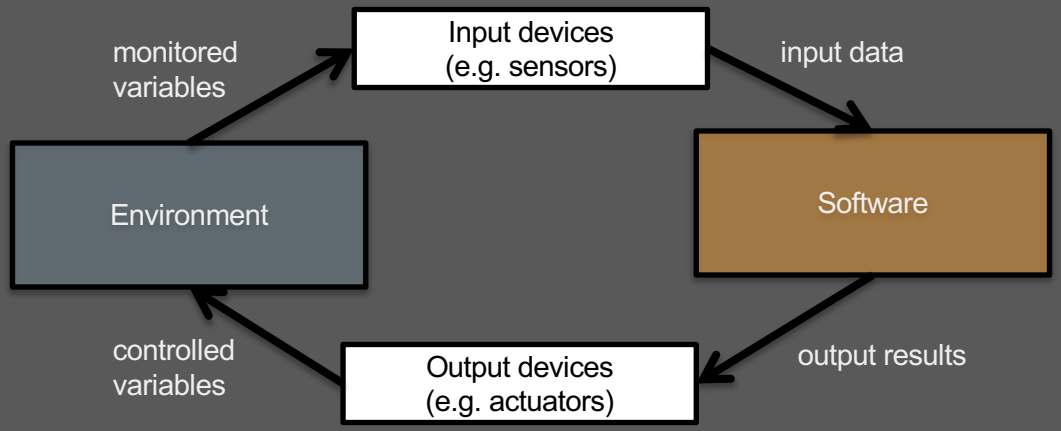
Requirements say what the system will do (and not how it will do it).

Why not "how"?

THE WORLD AND THE MACHINE

Environment and the Machine





Pamela Zave & Michael Jackson, "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology*, 6(1): 1-30, 1997.

Actions of an ATM customer:

withdrawal-request(a, m)

Properties of the environment:

balance(b, p)

Actions of an ATM machine:

withdrawal-payout(a, m)

Properties of the machine:

expected-balance(b, p)

What other **models of the world** do machines maintain?

Domain knowledge

Refinement is the act of translating requirements into specifications (bridging the gap!)

Requirements: desired behavior (effect on the environment) to be realized by the proposed system.

Assumptions or domain knowledge: existing behavior that is unchanged by the proposed system.

- Conditions under which the system is guaranteed to operate correctly.
- How the environment will behave in response to the system's outputs.

Some gaps must remain…

- Unshared actions cannot be accurately expressed in the machine
  - People can jump over gates (enter without unlocking)
  - People can steal or misplace inventory
- Future requirements are also not directly implementable
  - Phone system: "After all digits have been dialed, do *ring-back, busy-tone* or *error-tone*."
  - …how do you know the user is done dialing?

Assumptions?

IMPLEMENTATION BIAS

Requirements say what the system will do
(and not how it will do it).

Why not “how”?

Avoiding implementation bias

Requirements describe what is observable at the environment-machine interface.

Indicative mood describes the environment (as-is)

Optative mood to describe the environment with the machine (to-be).

This can be subtle…

"The dictionary shall be stored in a hash table" vs. "the software shall respond to requests within 5 seconds."

Instead of "what" vs. "how", ask "is this requirement only a property of the machine domain?"

Or is there some application domain phenomenon that justifies it?

QUALITY REQUIREMENTS

Functional Requirements

- What the machine should do
  - Input
  - Output
  - Interface
  - Response to events
- Criteria
  - Completeness: All requirements are documented
  - Consistency: No conflicts between requirements
  - Precision: No ambiguity in requirements

Quality/non-funct.) requirements

Specify not the functionality of the system, but the quality with which it delivers that functionality.

Can be more critical than functional requirements

- Can work around missing functionality
- Low-quality system may be unusable

Examples?

Here's the thing…

Who is going to ask for a slow, inefficient, unmaintainable system?

A better way to think about quality requirements is as *design criteria to help choose between alternative implementations.*

Question becomes: to what extent must a product satisfy these requirements to be acceptable?

Expressing quality requirements

Requirements serve as contracts: they should be testable/falsifiable.

Informal goal: a general intention, such as ease of use.

- May still be helpful to developers as they convey the intentions of the system users.

Verifiable non-functional requirement: A statement using some measure that can be objectively tested.

Examples

Confidentiality requirement: A non-staff patron may never know which books have been borrowed by others.

Privacy requirement: The diary constraints of a participant may never be disclosed to other invited participants without his or her consent.

Integrity req: The return of book copies shall be encoded correctly and by library staff only.

Availability req: A blacklist of bad patrons shall be made available at any time to library staff. Information about train positions shall be available at any time to the vital station computer.

Source: van Lamsweerde. Requirements Engineering: … Wiley 2009

Examples

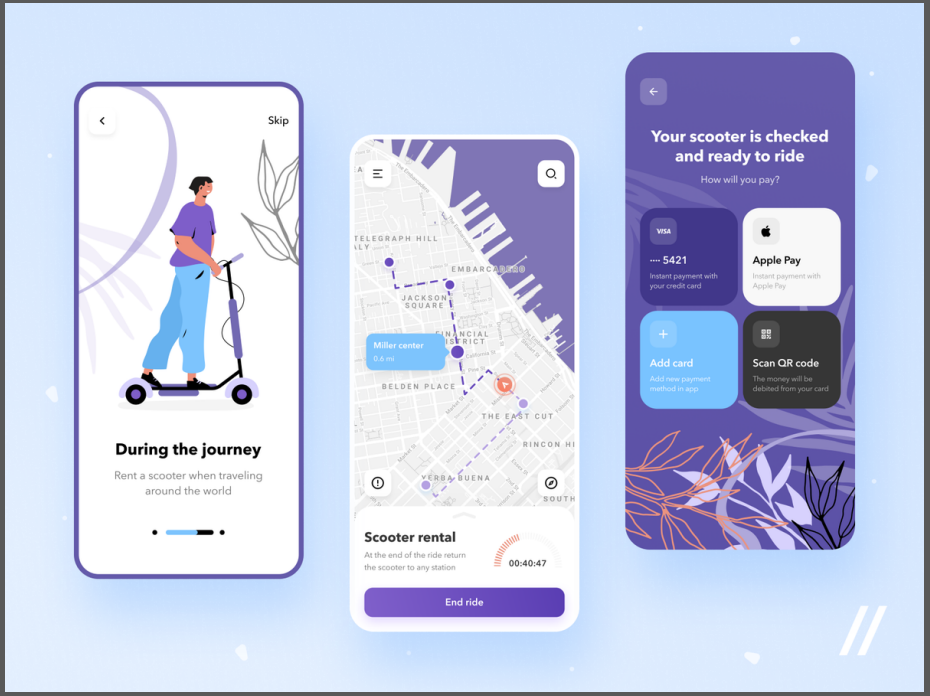
Informal goal: “the system should be easy to use by experienced controllers, and should be organized such that user errors are minimized.”

Verifiable non-functional requirement: “Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day, on average.”

Exercise: Quality REquirments

Let's write some quality requirements!

Try to write an informal goal, and then turn it into a verifiable non-functional requirement.



Requirements metrics

Property	Measure

REQUIREMENTS ELICITATION

Typical Steps

- Identify stakeholders
- Understand the domain
  - Analyze artifacts, interact with stakeholders
- Discover the real needs
  - Interview stakeholders
- Explore alternatives to address needs

Question

Who is the system for?

Stakeholders:

- End users
- System administrators
- Engineers maintaining the system
- Business managers
- ...who else?

Stakeholder

Any person or group who will be affected by the system, directly or indirectly.

Stakeholders may disagree.

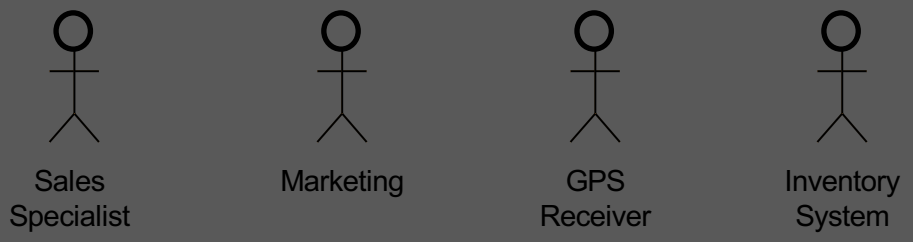
Requirements process should trigger negotiation to resolve conflicts.

Defining actors/agents

An actor is an entity that interacts with the system for the purpose of completing an event [Jacobson, 1992].

- Not as broad as stakeholders.

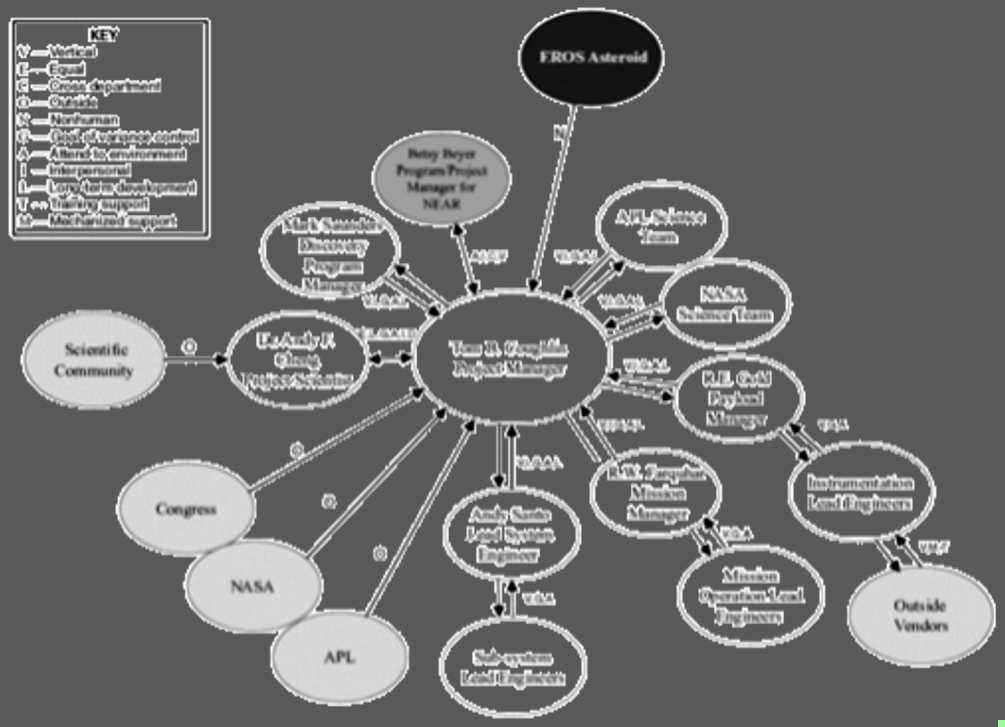
Actors can be a user, an organization, a device, or an external system.



Stakeholder analysis: criteria for identifying relevant stakeholders

- Relevant positions in the organization
- Effective role in making decisions about the system
- Level of domain expertise
- Exposure to perceived problems
- Influence in system acceptance
- Personal objectives and conflicts of interest

Stakeholders, a NASA example



From HSI NAP 11893

FIGURE 6-3 Role network for National Aeronautics and Space Administration (NASA's) Near Earth Asteroid Rendezvous project.

Challenges

Distributed knowledge

Conflicting knowledge

Difficult to access sources

Communication barriers (cultural, terminology, backgrounds)

Hidden needs, tactic knowledge

Politics, unstable conditions

Studying Artifacts (Content Analysis)

Learn about the domain

- Books, articles, Wikipedia

Learn about the system to be replaced

- How does it work? What are the problems? Manuals? Bug reports?

Learn about the organization

Knowledge reuse from other systems?

Interviews

Interview Tradeoffs

- Strengths
  - What stakeholders do, feel, prefer
  - How they interact with the system
  - Challenges with current systems
- Weaknesses
  - Subjective, inconsistencies
  - Capturing domain knowledge
  - Familiarity
  - Technical subtlety
  - Organizational issues, such as politics
  - Hinges on interviewer skill

Interview Process

Identify stakeholder of interest and target information to be gathered.

Conduct interview.

(structured/unstructured, individual/group)

Record + transcribe interview

Report important findings.

Check validity of report with interviewee.

Example: Identifying Problems

What problems do you run into in your day-to-day work? Is there a standard way of solving it, or do you have a workaround?

> Why is this a problem? How do you solve the problem today? How would you ideally like to solve the problem?

Keep asking follow-up questions ("What else is a problem for you?", "Are there other things that give you trouble?") for as long as the interviewee has more problems to describe.

So, as I understand it, you are experiencing the following problems/needs (describe the interviewee's problems and needs in your own words – often you will discover that you do not share the same image. It is very very common to not understand each other even if at first you think you do).

Just to confirm, have I correctly understood the problems you have with the current solution?

Are there any other problems you're experiencing? If so, what are they?

Capturing v. Synthesizing

- Engineers acquire requirements from many sources
  - Elicit from stakeholders
  - Extract from policies or other documentation
  - Synthesize from above + estimation and invention
- Because stakeholders do not always know what they want, engineers must…
  - Be faithful to stakeholder needs and expectations
  - Anticipate additional needs and risks
  - Validate that “additional needs” are necessary or desired

Interview Advice

Get basic facts about the interviewee before (role, responsibilities, …)

Review interview questions before interview

Begin concretely with specific questions, proposals; work through prototype or scenario

- Relate to current system, if applicable.

Be open-minded; explore additional issues that arise naturally, but stay focused on the system.

Contrast with current system/alternatives. Explore conflicts and priorities

Plan for follow-up questions

Bonus: Guidelines for effective interviews

Identify the right interviewee sample for full coverage of issues

- different responsibilities, expertise, tasks, exposure to problems

Come prepared, to focus on right issue at right time

- backgound study first
- predesign a sequence of questions for this interviewee

Centre the interview on the interviewee's work & concerns

Keep control over the interview

Make the interviewee feel comfortable

- Start: break ice, provide motivation, ask easy questions
- Consider the person too, not only the role
- Do always appear as a trustworthy partner

Bonus: Guidelines for effective interviews

Be focused, keep open-ended questions for the end

Be open-minded, flexible in case of unexpected answers

Ask why-questions without being offending

Avoid certain types of questions …

- opinion or biased
- affirmative
- obvious or impossible answer for this interviewee

Edit & structure interview transcripts while still fresh in mind

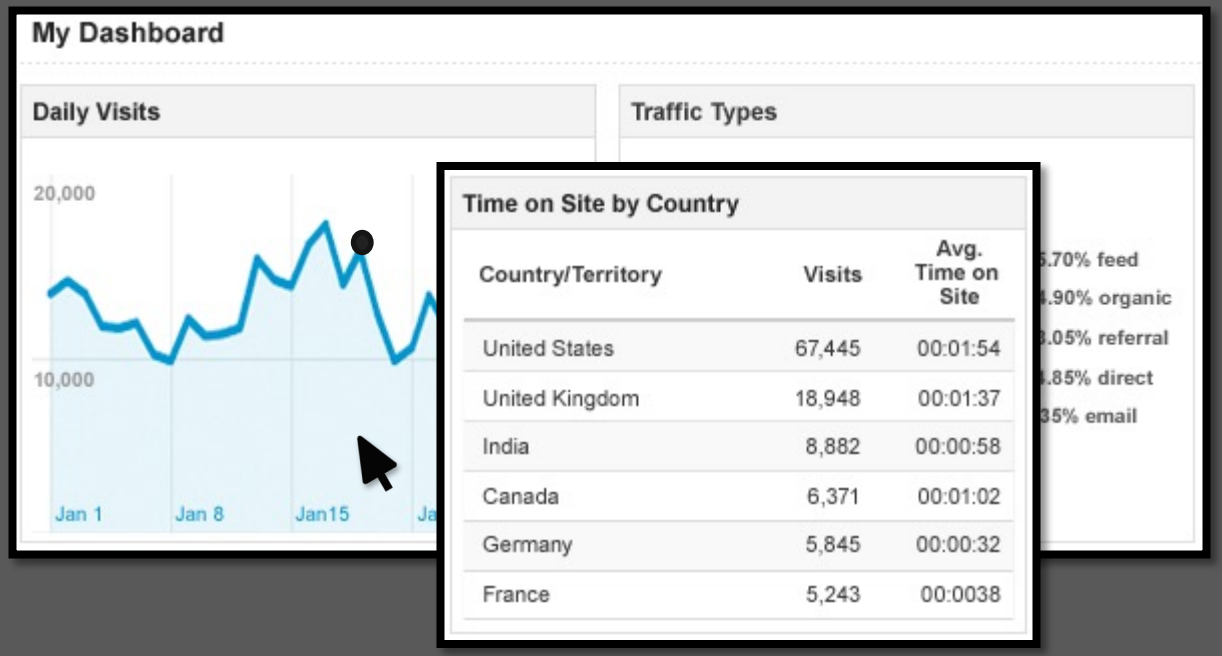
- including personal reactions, attitudes, etc

Keep interviewee in the loop

- co-review interview transcript for validation & refinement

PROTOTYPES, MOCKUPS, STORIES

High- vs low- fidelity mockups



Mockups, Prototypes, Stories

Humans: better at recognizing whether a solution is correct than solving the problem from a blank page.

Mock-ups/prototypes help explore uncertainty in the requirements.

- Validate that we have the right requirements.
- Elicit requirements at the “borders” of the system.
- Assert feasibility of solution space.
- Get feedback on a candidate solution.

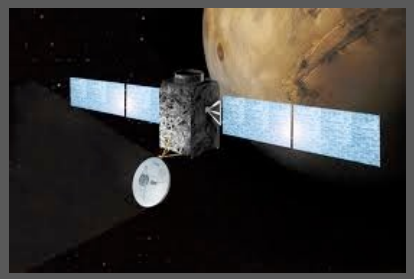
“I’ll know it when I see it”

Rapid prototyping

Throw-away: developed to learn more about a problem, not intended for actual use.

Evolutionary: intended to be incorporated into the final product.

Storyboarding and scenarios




Story

Who the players are

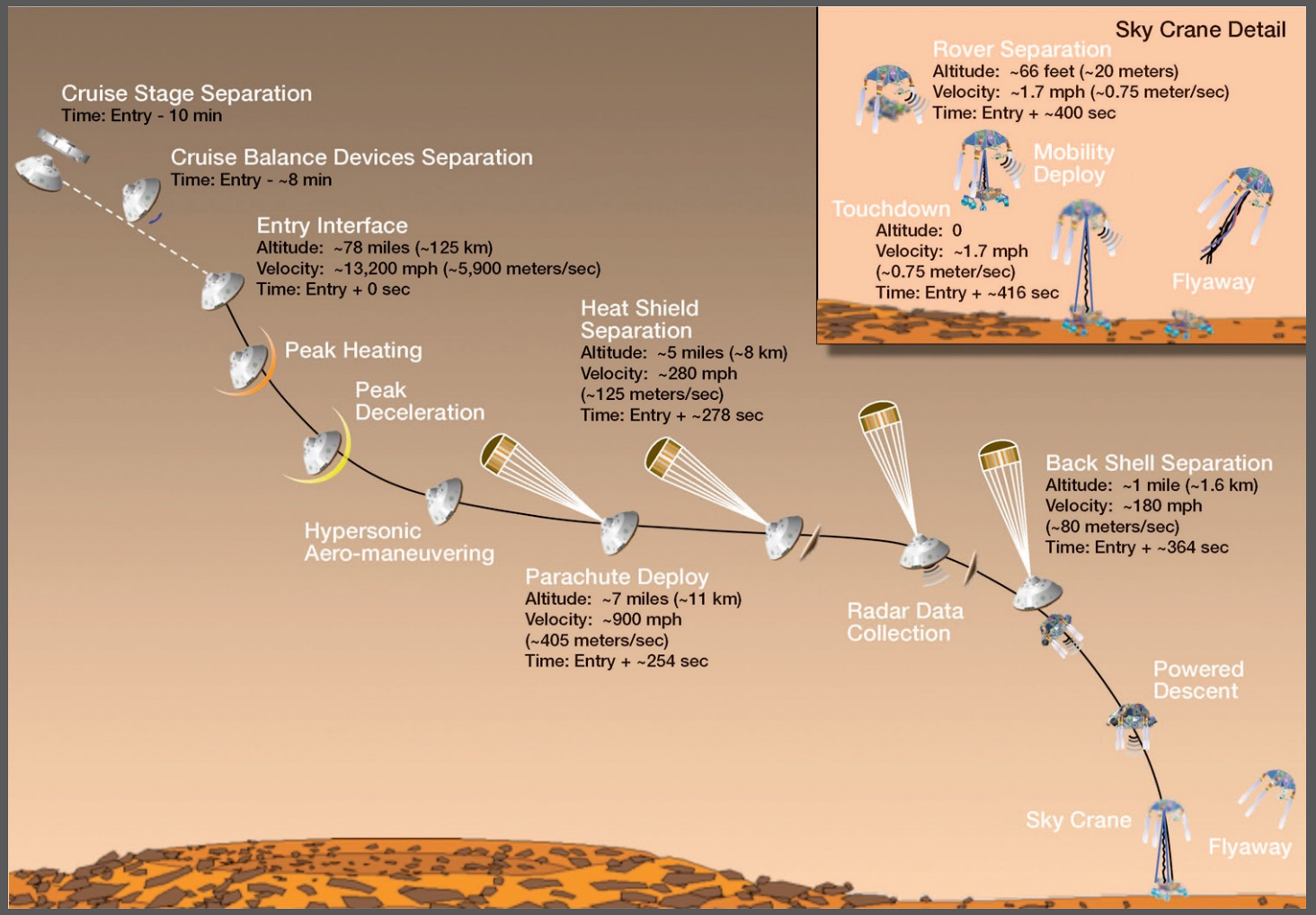
What happens to them

How it happens through specific episode

Why this happens

What if such and such an event occurs

What could go wrong as a consequence


Cruise Stage Separation
Time: Entry - 10 min
Cruise Balance Devices Separation
Time: Entry - ~8 min
Entry Interface
Altitude: ~78 miles (~125 km)
Velocity: ~13,200 mph (~5,900 meters/sec)
Time: Entry + 0 sec
Peak Heating
Peak Deceleration
Heat Shield Separation
Altitude: ~5 miles (~8 km)
Velocity: ~280 mph
(~125 meters/sec)
Time: Entry + ~278 sec
Hypersonic Aero-maneuvering
Parachute Deploy
Altitude: ~7 miles (~11 km)
Velocity: ~900 mph
(~405 meters/sec)
Time: Entry + ~254 sec
Radar Data Collection
Back Shell Separation
Altitude: ~1 mile (~1.6 km)
Velocity: ~180 mph
(~80 meters/sec)
Time: Entry + ~364 sec
Powered Descent
Sky Crane
Flyaway
Sky Crane Detail
Rover Separation
Altitude: ~66 feet (~20 meters)
Velocity: ~1.7 mph (~0.75 meter/sec)
Time: Entry + ~400 sec
Mobility Deploy
Touchdown
Altitude: 0
Velocity: ~1.7 mph
(~0.75 meter/sec)
Time: Entry + ~416 sec
Flyaway

RESOLVING CONFLICTS

Types of inconsistency

Terminology clash: same concept named differently in different statements

e.g. library management: "borrower" vs. "patron"

Designation clash: same name for different concepts in different statements

e.g. "user" for "library user" vs. "library software user"

Structure clash: same concept structured differently in different statements

e.g. "latest return date" as time point (e.g. Fri 5pm)

vs. time interval (e.g. Friday)

Types of inconsistency, 2

- Strong conflict: statements not satisfiable together
  - e.g. “participant constraints may not be disclosed to anyone else” vs. “the meeting initiator should know participant constraints”
- Weak conflict (divergence): statements not satisfiable together under some boundary condition
  - “patrons shall return borrowed copies within X weeks” vs “patrons shall keep borrowed copies as long as needed” contradict only if “needed>x weeks”

Handling inconsistencies

Terminology, designation, structure: Build glossary

Weak, strong conflicts: Negotiation required

- Cause: different objectives of stakeholders => resolve outside of requirements
- Cause: quality tradeoffs => explore preferences

Requirements Traceability

Keep connections between requirements

What follows from what

Requirements prioritization

Cost, time, and other limits

Dependencies among requirements

Nice to have

Strategies to base on value contribution

Summary

Many solicitation strategies, including document analysis, interviews, and ethnography

Do not underestimate the challenge of interviews

Resolving conflicts

Using prototypes to enhance discussions and decision making

Many documentation strategies; our focus is on *user stories*