Foundations of Software Engineering

Lecture 5 – Requirements (1/3)

Claire Le Goues

institute for SOFTWARE RESEARCH

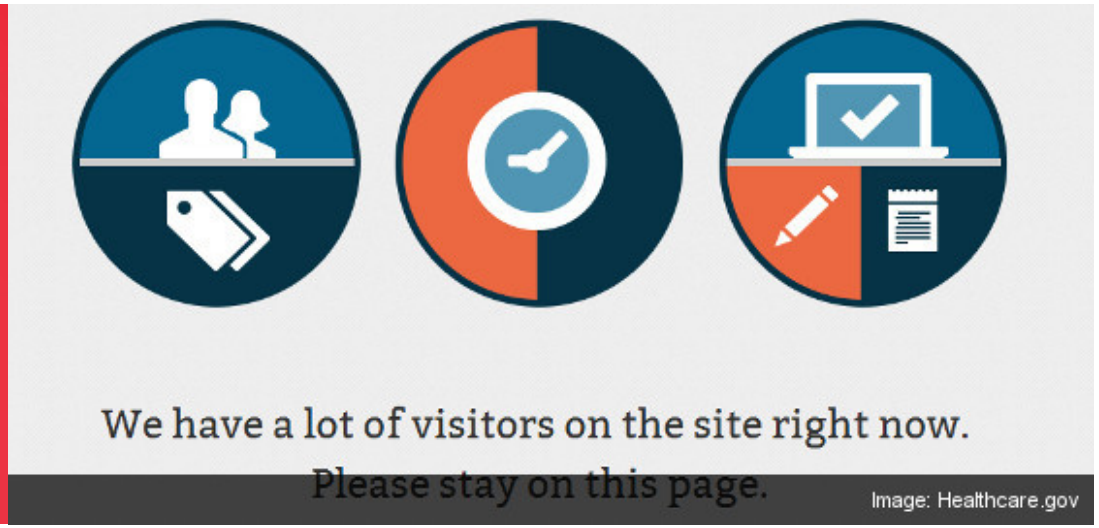
Administrivia

- Canvas OK?
  - Vs Piazza?

Learning goals

- Explain with examples the importance of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment.
- Distinguish between and give examples of: functional and non-functional requirements; informal statements and verifiable requirements.
- Identify system stakeholders and develop approaches on how to interview them.

Healthcare.gov



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:

1. Incomplete requirements (13.1%)
2. Lack of user involvement (12.4%)
3. Lack of resources (10.6%)
4. Unrealistic expectations (9.9%)
5. Lack of executive support (9.3%)
6. Changing requirements and specifications (8.7%)
7. Lack of planning (8.1%)
8. System no longer needed (7.5%) .

Overly simplified definition.

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

WHY IS THIS HARD?

Communication problem

Goal: figure out what should be built.

Express those ideas so that the correct thing is built.

Four Kinds of Denial

- Denial by prior knowledge – we have done this before, so we know **what** is required
- Denial by hacking – our fascination with machines dominates our focus on the **how**
- Denial by abstraction – we pursue elegant models which obscure, remove or downplay the real world
- Denial by vagueness – imply (vaguely) that machine descriptions are actually those of the world

Michael Jackson, "The World and the Machine," *International Conference on Software Engineering*, pp. 283-292, 1995.

Less simplified definition.

- Stories: Scenarios and Use Cases

  "After the customer submits the purchase information and the payment has been received, the order is fulfilled and shipped to the customer's shipping address."

- Optative statements

  The system *shall* notify clients about their shipping status

- Domain Properties and Assumptions

  Every product has a unique product code

  Payments will be received after authorization

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
- Examples?

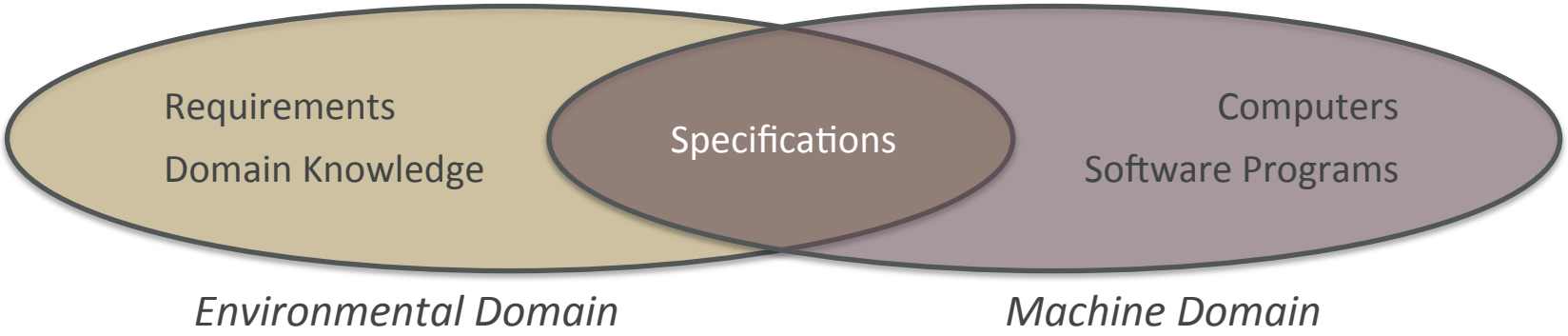
Here's the thing…

- 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?

Functional requirements and implementation bias

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

Environment and the Machine



- The pilot shall decrease airspeed and lower the landing gear prior to decision height
- The plane shall lower the wing flaps to 30° for landing

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

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).

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.

Assumptions?

Assumptions?

Grounding, or: Reality

- **Able**: Two important basic types are *student* and *course*. There is also a binary relation *enrolled*.
- **Baker**: Do only students enroll in courses? I don't think that's true.
- **Able**: But that's what I mean by student!
- Designation: the meaning of a primitive.
  - Will be explained informally, but should still be *precise*, *recorded*, and *maintained*.

Designations ground formal specifications.

- Specifications are logical expressions of shared actions at the interface of the machine

$$\forall\, s\, \forall\, c(\text{enrolled}(s, c) \Rightarrow \text{student}(s) \wedge \text{course}(c))$$

- This includes linking domain properties and agent actions as pre- and post-conditions

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?

Lufthansa Flight 2904 - 1

- The Airbus A320-200 airplane has a software-based braking system that consists of:
  - Ground spoilers (wing plates extended to reduce lift)
  - Reverse thrusters
  - Wheel brakes on the main landing gear
- To engage the braking system, the **wheels of the plane must be on the ground**.

Is this a shared or an unshared action/condition?

Lufthansa Flight 2904 - 2

There are two "on ground" conditions:

1. Either shock absorber bears a load of 6300 kgs
2. Both wheels turn at 72 knots (83 mph) or faster

- Ground spoilers activate for conditions 1 or 2
- Reverse thrust activates for condition 1 on both main landing gears
- Wheel brake activation depends upon the rotation gain and condition 2

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

- **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."

Web video scenario

- You work for a custom software developer as a requirements engineer.
- Client's stated requirement: **"We want to sell videos on the web."**
- You must now engage the client to elaborate the stated requirement
  - How will you proceed?
  - How shall we express what we learn?

Requirements metrics

Property	Measure

Web video scenario

- You work for a custom software developer as a requirements engineer.
- Client's stated requirement: **"We want to sell videos on the web."**
- You must now engage the client to elaborate the stated requirement
  - How will you proceed?
  - How shall we express what we learn?
- Step 1: talk to the relevant **stakeholders**

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.
  - (We will return to conflicts).

Defining actors/agents

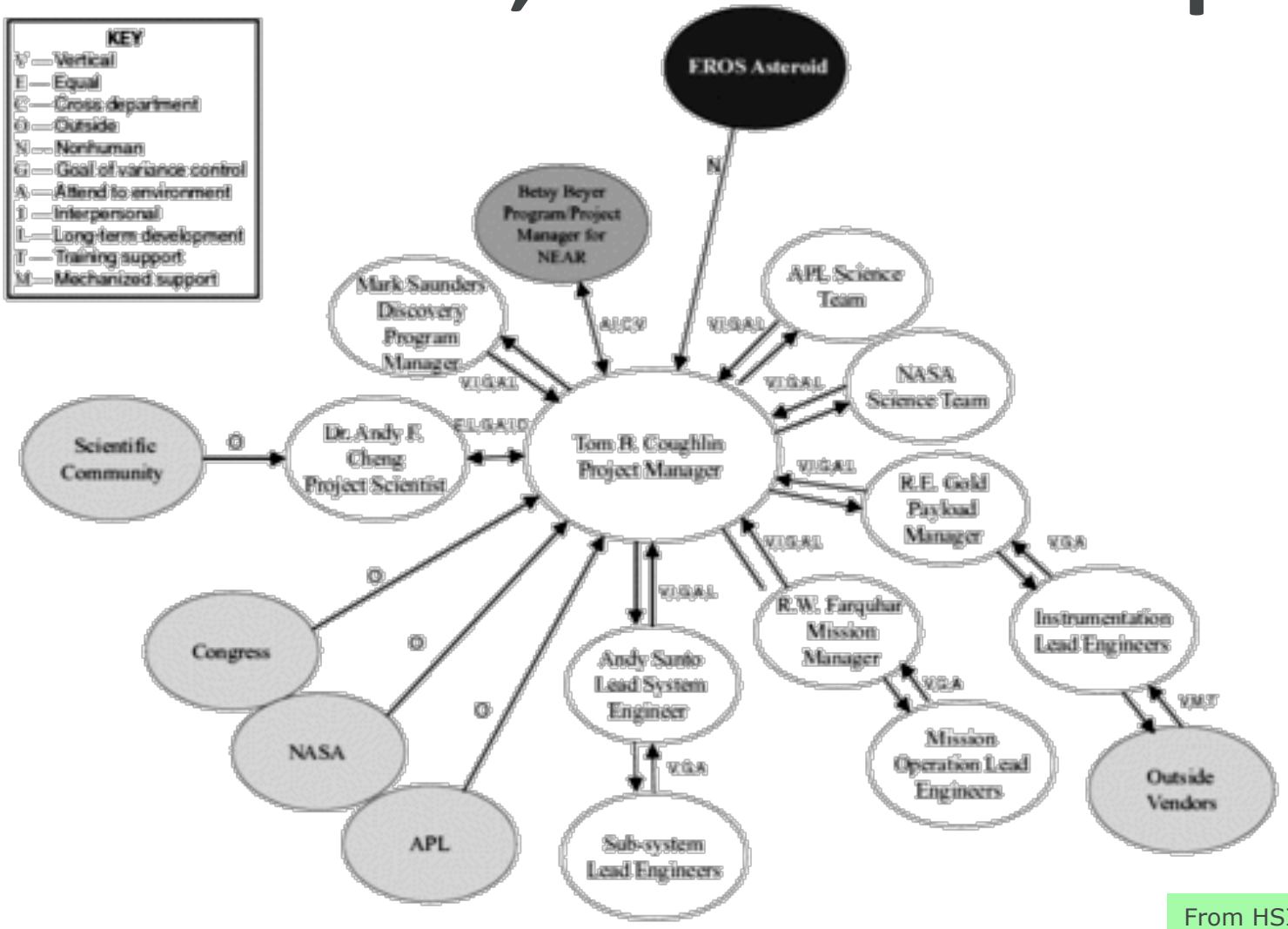
- An actor is an entity that interacts with the system for the purpose of completing an event [Jacobson, 1992].
  - Not as broad as stakeholders.
- Actors can be a user, an organization, a device, or an external system.

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

CHALLENGES?

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.

Interviews

- Goal: understand functional requirements, identify and learn domain-specific concepts, prioritize quality attributes.
- Effective interviewers:
  - Begin concretely with specific questions, proposals, or working through a prototype.
  - Are open-minded and willing to explore additional issues that arise naturally, but stay focused on the system.
- **Stories, scenarios, use cases (informal), hypotheticals, examples.**
- **Understand/contrast with current system (if applicable).**

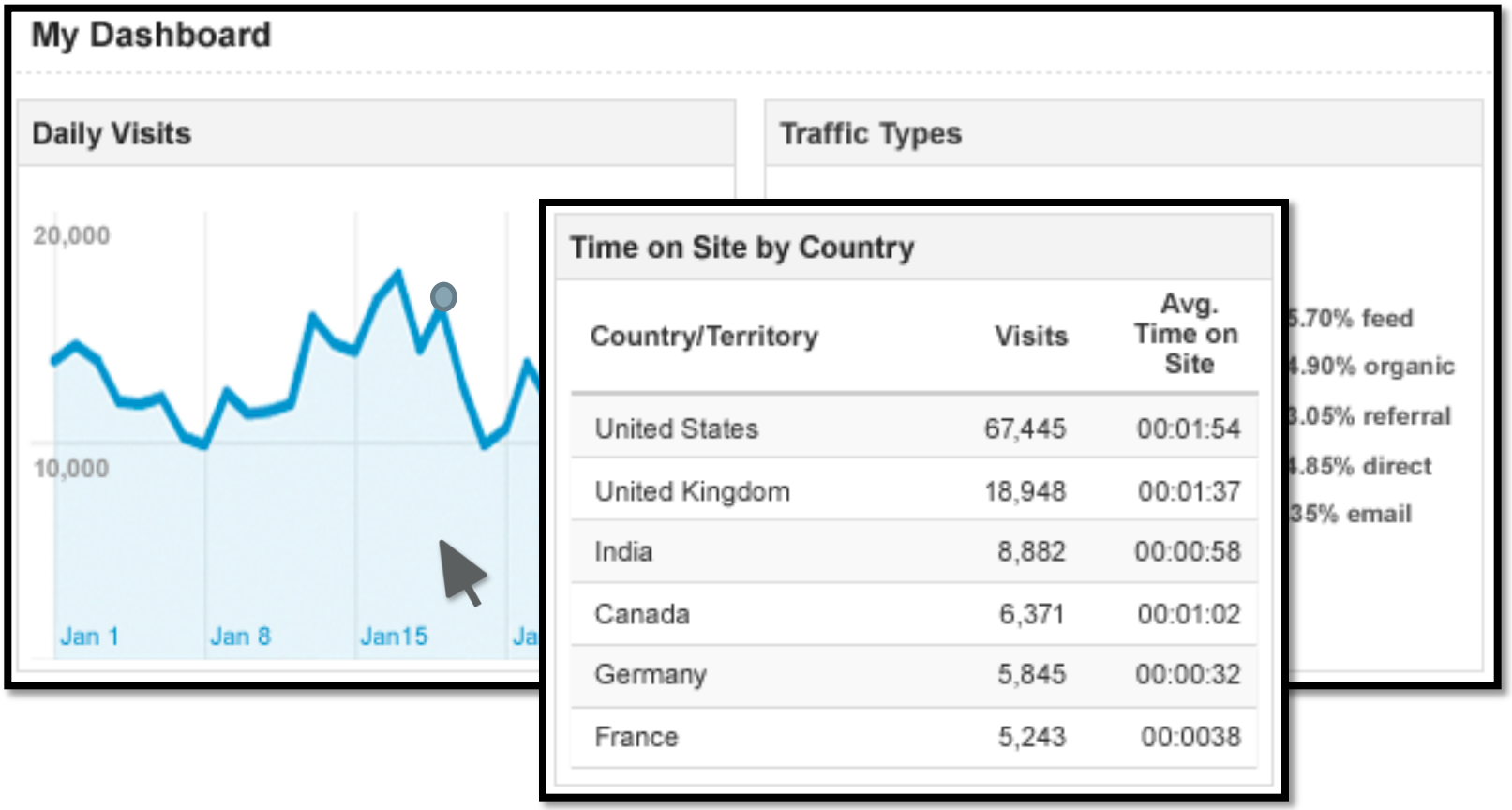
Interview benefits vs drawbacks

- Strengths
  - What stakeholders do
  - How they interact with the system
  - Challenges with current systems
- Weaknesses
  - Capturing domain knowledge
  - Familiarity
  - Technical subtlety
  - Organizational issues, such as politics
  - Completeness

- Yai: non-profit; most employees in social work field
- Currently sells training DVDs for companies on issues related to individuals with developmental disabilities.
- **“Do you know how we can sell course materials online?”**

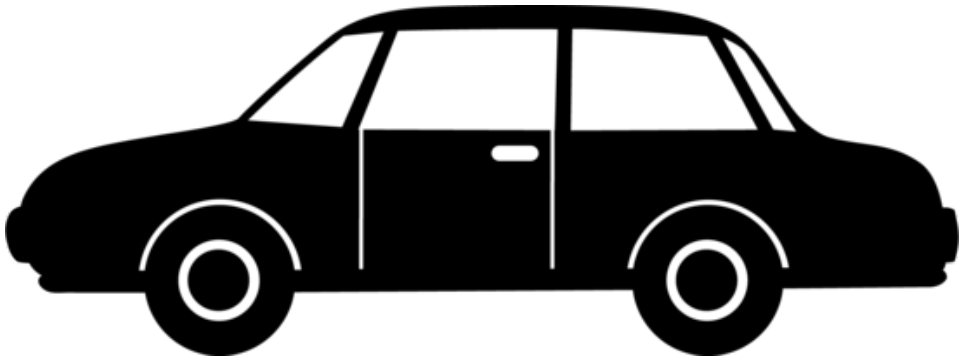
- Humans: better at recognizing whether a solution is correct than solving the problem from a blank page.
- **Mock-ups/prototypes** help explore uncertainty in the requirements.
  - Validate that we have the right requirements.
  - Elicit requirements at the "borders" of the system.
  - Assert feasibility of solution space.
  - Get feedback on a candidate solution.
  - Also good for questions about UI.
- "I'll know it when I see it" ←scary.

High- vs low- fidelity mockups



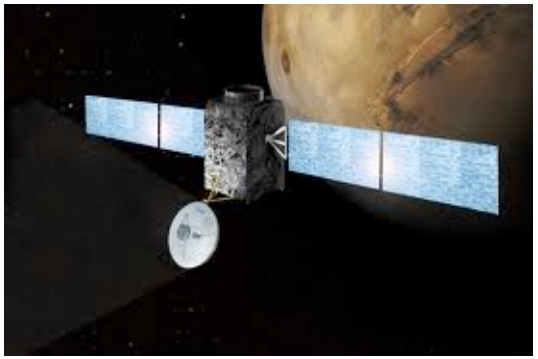
Rapid prototyping

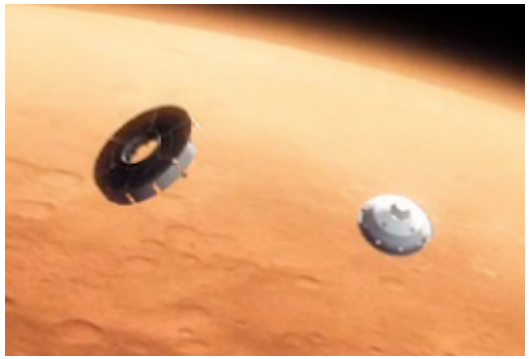
- Throw-away: developed to learn more about a problem, not intended for actual use.



- Evolutionary: intended to be incorporated into the final product.

Storyboarding and scenarios

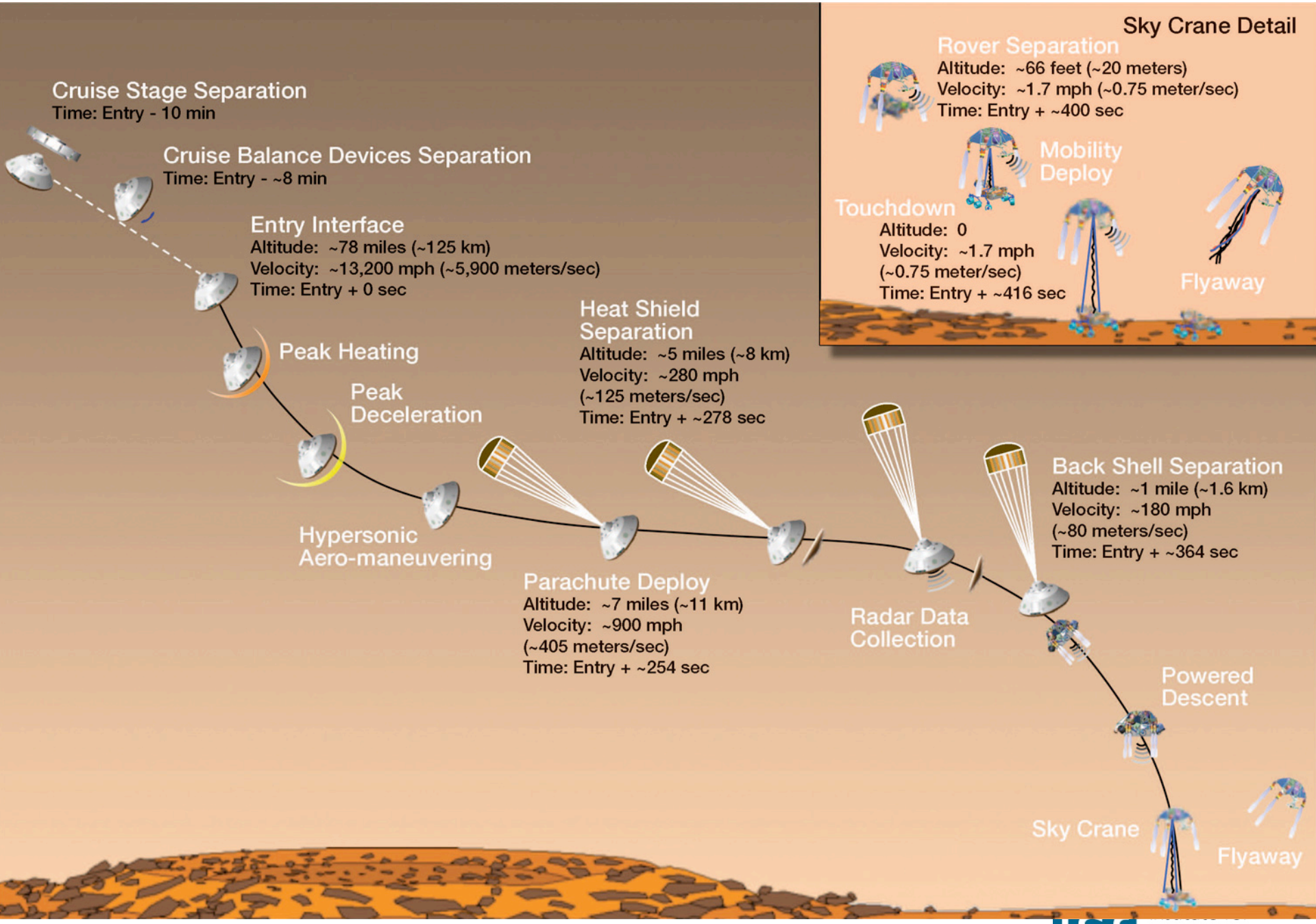




Story

- **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

- **Storyboards illustrate scenarios**: a typical sequence of interaction among system components that meets an implicit objective.
  - **Storyboards** explicitly cover at least **who**, **what**, and **how**.
- Different types:
  - Positive vs negative (should and should not happen)
  - Normal vs abnormal
- As part of elicitation:
  - Learn about current or proposed system by walking through real-life or hypothetical sequences
  - Can ask specific questions
  - Elicit the underlying objectives, generalize into models of desired behaviors.
  - Identify and resolve conflicts
- Pluses: Concrete, support narrative description
- Minuses: inherently partial.


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
Hypersonic Aero-maneuvering
Heat Shield Separation
Altitude: ~5 miles (~8 km)
Velocity: ~280 mph
(~125 meters/sec)
Time: Entry + ~278 sec
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

Scenarios

Test cases

- Questions to consider
  - What tasks does the actor perform?
  - What information is accessed and modified, and where does it come from?
  - What are obligations on the actor to inform the system?
  - What are obligations of the system to inform the actor?
- Heuristics
  - Vertical – one worked-out specific scenario, to understand how to engage the user/stakeholder
  - Horizontal – multiple, less-detailed scenarios, to assess scope and context
  - Mock-ups
  - Alternatives
  - Can be **passive** or **active**.

Where do we find scenarios?

- Reflective
  - Elicit scenarios from stakeholders
  - Analyze reports and case studies of similar systems
- Prospective – Instantiate the use case

Don't only focus on the positive
– include failures, too

Use cases

- We talk about many types, at different granularities:
  - Full use case model (whole-system, higher-level)
  - "Agile" use case: small, concrete pieces of system functionality to be implemented (sometimes conflated with "user stories")
- Used at multiple stages:
  - Requirements elicitation (illustrated, validate, requirements; highlight conflicts, prioritize requirements, etc).
  - Requirements documentation.
  - Concrete design: UML diagrams.

Use cases

- *Text stories* of an actor using a system to meet goals.
- *Use cases are not diagrams, they are text.*
- Primarily serve as functional requirements (by contrast/in conjunction with “the system shall” statements.)

Learning goals

- Explain with examples the importance of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment.
- Distinguish between and give examples of: functional and non-functional requirements; informal statements and verifiable requirements.
- Identify system stakeholders and develop approaches on how to interview them.