



# Lecture 13: Requirements Analysis



# Mars Polar Lander

## Launched

3 Jan 1999

## Mission

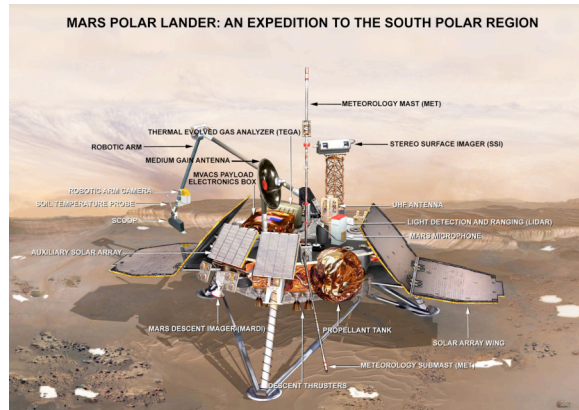
Land near South Pole  
Dig for water ice with a robotic arm

## Fate:

Arrived 3 Dec 1999  
No signal received after initial phase of descent

## Cause:

Several candidate causes  
Most likely is premature engine shutdown due to noise on leg sensors





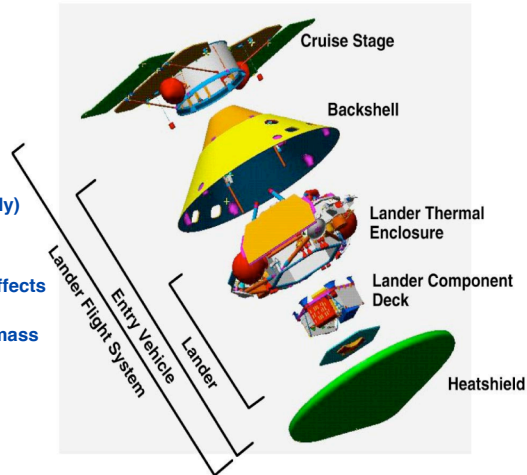
# What happened?

## We don't know for sure:

spacecraft not designed to send telemetry during descent  
This decision severely criticized by review boards

## Possible causes:

1. Lander failed to separate from cruise stage (plausible but unlikely)
2. Landing site too steep (plausible)
3. Heatshield failed (plausible)
4. Loss of control due to dynamic effects (plausible)
5. Loss of control due to center-of-mass shift (plausible)
6. Premature Shutdown of Descent Engines (**most likely!**)
7. Parachute drapes over lander (plausible)
8. Backshell hits lander (plausible but unlikely)



# Premature Shutdown Scenario

## Cause of error

Magnetic sensor on each leg senses touchdown  
Legs unfold at 1500m above surface  
software accepts transient signals on touchdown sensors during unfolding

## Factors

**System** requirement to ignore the transient signals  
**But the software** requirements did not describe the effect  
Engineers present at code inspection didn't understand the effect  
Not caught in testing because:  
Unit testing didn't include the transients  
Sensors improperly wired during integration tests (no touchdown detected!)

## Result of error

Engines shut down before spacecraft has landed  
estimated at 40m above surface, travelling at 13 m/s  
estimated impact velocity 22m/s (spacecraft would not survive this)  
nominal touchdown velocity 2.4m/s



SYSTEM REQUIREMENTS

FLIGHT SOFTWARE REQUIREMENTS

3.7.2.2.4.2 Processing

- 1) The touchdown sensors shall be sampled at 100-Hz rate.  
The sampling process shall be initiated prior to lander entry to keep processor demand constant.  
However, the use of the touchdown sensor data shall not begin until 12 meters above the surface.
- 2) Each of the 3 touchdown sensors shall be tested automatically and independently prior to use of the touchdown sensor data in the onboard logic.  
The test shall consist of two (2) sequential sensor readings showing the expected sensor status.  
If a sensor appears failed, it shall not be considered in the descent engine termination decision.
- 3) Touchdown determination shall be based on two sequential reads of a single sensor indicating touchdown.

- a. The lander flight software shall cyclically check the state of each of the three touchdown sensors (one at 100 Hz during EDL).
- b. The lander flight software shall be able to cyclically check the touchdown event state with or without touchdown event generation enabled.
- c. Upon enabling touchdown event generation, the lander flight software shall attempt to detect failed sensors marking the sensor as bad when the sensor indicates "touchdown state" on two consecutive reads.
- d. The lander flight software shall generate the landing event based on two consecutive reads indicating touchdown from any one of the good touchdown sensors.

Adapted from the "Report of the Loss of the Mars Polar Lander and Deep Space 2 Missions -- JPL Special Review Board (Casani Report) - March 2000".  
See <http://www.nasa.gov/newsinfo/marsreports.html>



# Quality = Fitness for purpose

## Software technology is everywhere

Affects nearly all aspects of our lives

But our experience of software technology is often frustrating/disappointing

## Software is designed for a purpose

If it doesn't work well then either:

...the designer didn't have an adequate understanding of the purpose

...or we are using the software for a purpose different from the intended one

Requirements analysis is about identifying this purpose

Inadequate understanding of the purpose leads to poor quality software

## The purpose is found in human activities

E.g. Purpose of a banking system comes from the business activities of banks and the needs of their customers

The purpose is often complex:

Many different kinds of people and activities

Conflicting interests among them



# Designing for people

## What is the real goal of software design?

- Creating new programs, components, algorithms, user interfaces,...?
- Making human activities more effective, efficient, safe, enjoyable,...?

## How rational is the design process?

### Hard systems view:

- Software problems can be decomposed systematically
- The requirements can be represented formally in a specification
- This specification can be validated to ensure it is correct
- A correct program is one that satisfies such a specification

### Soft systems view:

- Software development is embedded in a complex organisational context
- There are multiple stakeholders with different values and goals
- Software design is part of an ongoing learning process by the organisation
- Requirements can never be adequately captured in a specification
- Participation of users and others throughout development is essential

### Reconciliation:

- Hard systems view okay if there is local consensus on the nature of the problem



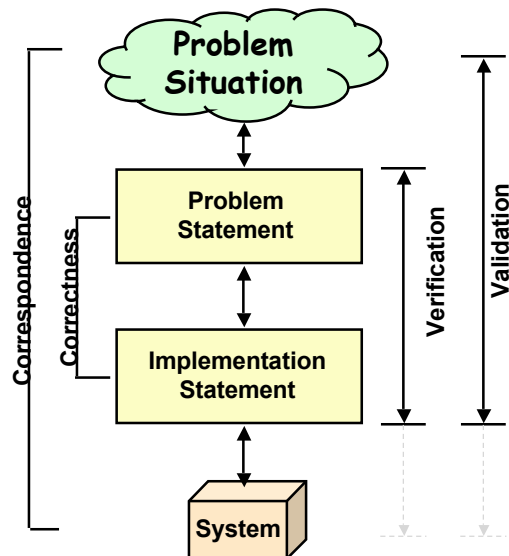
# Separate the problem from the solution

## A separate problem description is useful:

- Most obvious problem might not be the right one to solve
- Problem statement can be discussed with stakeholders
- Problem statement can be used to evaluate design choices
- Problem statement is a source of good test cases

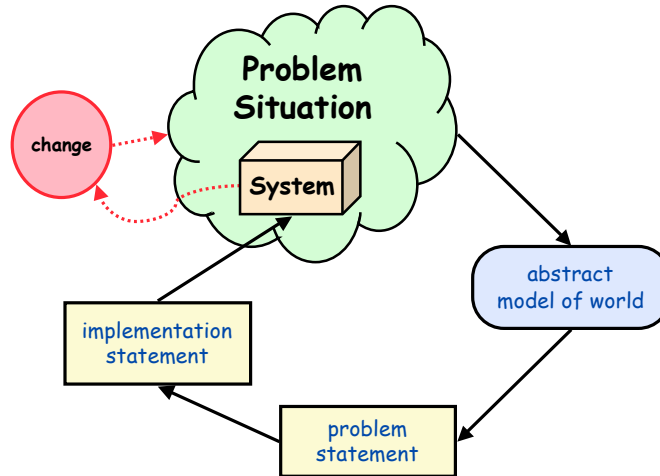
## Still need to check:

- Solution **correctly** solves the stated problem
- Problem statement **corresponds** to the needs of the stakeholders

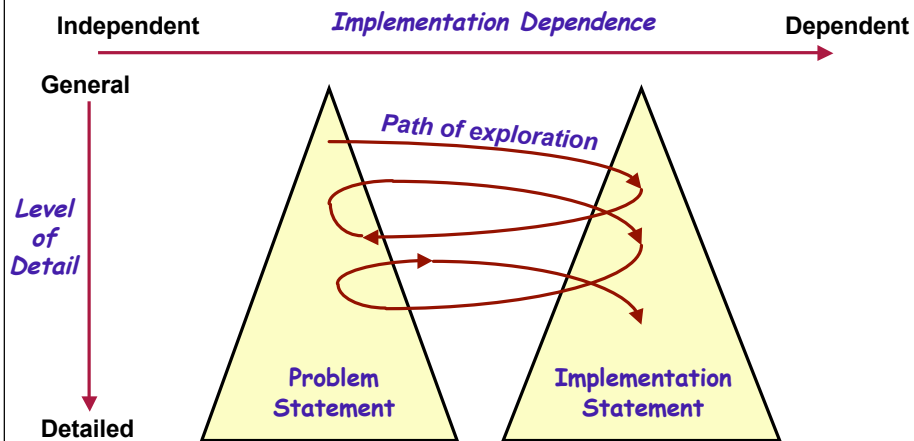




# But design changes the world...



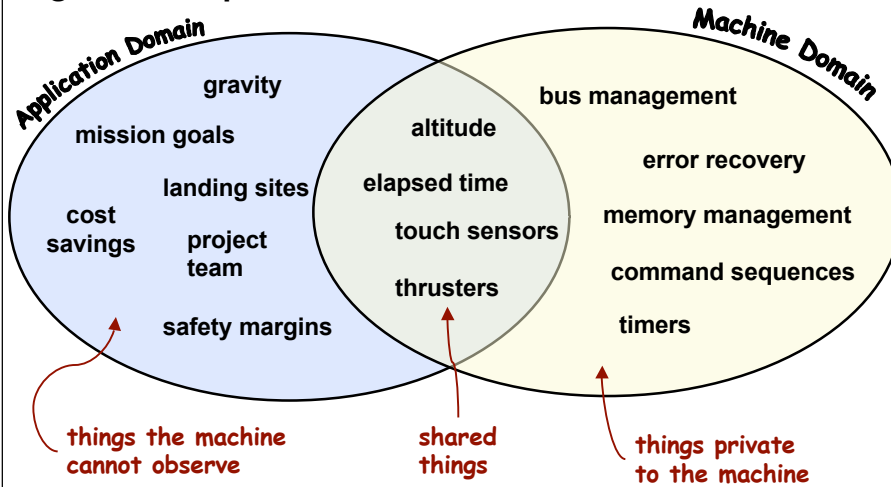
# Intertwining of problems and solutions





# A problem to describe...

E.g. "land a spacecraft on Mars"



# Thinking about Software Requirements



### Domain Properties (assumptions):

things in the application domain that are true whether or not we ever build the proposed system

### (System) Requirements:

things in the application domain that we wish to be made true by delivering the proposed system

Many of which will involve phenomena the machine has no access to

### A (Software) Specification:

is a description of the behaviours that the program must have in order to meet the requirements

Can only be written in terms of shared phenomena!



## Fitness for purpose?

### Two correctness (verification) criteria:

- The Program running on a particular Computer satisfies the Specification
- The Specification, in the context of the given domain properties, satisfies the requirements

### Two completeness (validation) criteria:

- We discovered all the important requirements
- We discovered all the relevant domain properties

### Example:

#### Requirement R:

“Reverse thrust shall only be enabled when the aircraft is moving on the runway”

#### Domain Properties D:

- Wheel pulses on if and only if wheels turning
- Wheels turning if and only if moving on runway

#### Specification S:

Reverse thrust enabled if and only if wheel pulses on

Verification:  $S, D \models R$



## Another Example

### Requirement R:

“The database shall only be accessible by authorized personnel”

### Domain Properties D:

- Authorized personnel have passwords
- Passwords are never shared with non-authorized personnel

### Specification S:

Access to the database shall only be granted after the user types an authorized password

### S + D entail R

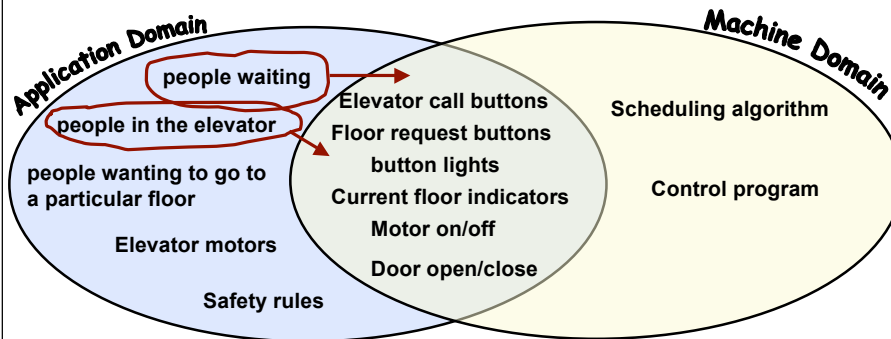
But what if the domain assumptions are wrong?





# But we can also move the boundaries...

E.g. Elevator control system:



→We can shift things around:

- ↳ E.g. Add some sensors to detect when people are waiting
- ↳ This changes the nature of the problem to be solved



# Observations

**Analysis is not necessarily a sequential process:**

- Don't have to write the problem statement before the solution statement
- (Re-)writing a problem statement can be useful at any stage of development
- RE activities continue throughout the development process

**The problem statement will be imperfect**

- RE models are approximations of the world
- will contain inaccuracies and inconsistencies
- will omit some information.
- assess the risk that these will cause serious problems!

**Perfecting a specification may not be cost-effective**

- Requirements analysis has a cost
- For different projects, the cost-benefit balance will be different
- Depends on the consequences of getting it wrong!

**Problem statement should never be treated as fixed**

- Change is inevitable, and therefore must be planned for
- There should be a way of incorporating changes periodically





# What do Requirements Analysts do?

## Starting point

Some notion that there is a “problem” that needs solving

e.g. dissatisfaction with the current system

e.g. a new business opportunity

e.g. a potential saving of cost, time, resource usage, etc.

A Requirements Analyst is an agent of change

## The requirements analyst must:

identify the “problem” / “opportunity”

Which problem needs to be solved? (identify problem **Boundaries**)

Where is the problem? (understand the **Context/Problem Domain**)

Whose problem is it? (identify **Stakeholders**)

Why does it need solving? (identify the stakeholders' **Goals**)

When does it need solving? (identify Development **Constraints**)

What might prevent us solving it? (identify **Feasibility and Risk**)

How might a software system help? (collect some **Scenarios / Use Cases**)

