# Requirements Engineering:
## finding out what customers *really* need

Prof. Steve Easterbrook

Dept of Computer Science
University of Toronto

sme@cs.toronto.edu



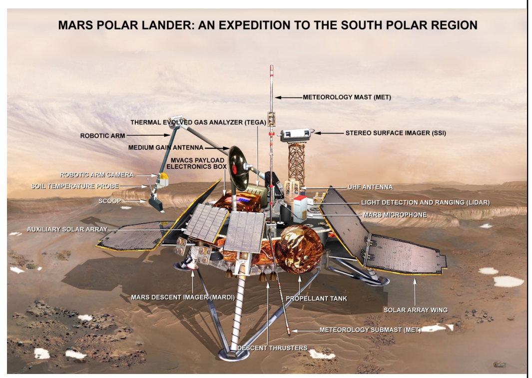



# Outline

- Case Study: Mars Polar Lander
- Basics of Requirements Engineering
  - more than just modeling!
  - roadmap of current research
- Where do requirements come from?
- How are requirements communicated?
- How do requirements evolve?
- Further Reading

# 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



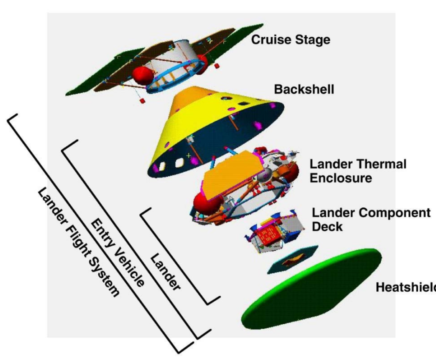






# What happened?

- **Investigation hampered by lack of data**
  - spacecraft not designed to send telemetry during descent
  - This decision severely criticized by review boards
- **Possible causes:**
  - Lander failed to separate from cruise stage (plausible but unlikely)
  - Landing site was too steep (plausible)
  - Heatshield failed (plausible)
  - Loss of control due to dynamic effects (plausible)
  - Loss of control due to center-of-mass shift (plausible)
  - Premature Shutdown of Descent Engines (most likely!)
  - Parachute drapes over lander (plausible)
  - 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
  - transient signals on touchdown sensors during unfolding
  - software accepts touchdown signals if they persist for 2 timeframes
  - transient signals likely to be this long on at least one leg

## ➜ Factors

- *System* requirement to ignore the transient signals
  - But the *software* requirements did not describe the effect
  - s/w designers didn't understand the effect, so didn't include the requirement
- Engineers present at code inspection didn't understand the effect
- Not caught in testing because:
  - Unit testing didn't include the transients (based on S/W reqts)
  - Sensors improperly wired during integration tests (no touchdown detected!)
  - Full test not repeated after re-wiring

## ➜ Result of error

- Engines shut down before spacecraft has landed
  - When engine shutdown s/w enabled, flags indicated touchdown already occurred
  - estimated at 40m above surface, travelling at 13 m/s
  - estimated impact velocity 22m/s (spacecraft would not survive this)
  - (c.f. nominal touchdown velocity 2.4m/s)






**SYSTEM REQUIREMENTS**

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. X

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.

**FLIGHT SOFTWARE REQUIREMENTS**

3.7.2.2.4.2 <u>Processing</u>

a. The lander flight software shall cyclically check the state of each of the three touchdown sensors (one pe 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 land 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

# Requirements Engineering

## → A definition of RE:

- "RE is concerned with identifying the purpose of a software system…
- "…and the contexts in which it will be used.
- "Hence, RE acts as the bridge between:
  - "the real world needs of users, customers, and other constituencies affected by a software system…
  - "…and the capabilities and opportunities afforded by software-intensive technologies."

[RE'01 call for papers see www.re01.org]

## → But what is a requirement?

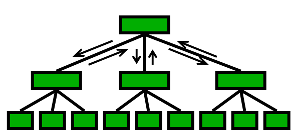
- "A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document…
- "…The set of all requirements forms the basis for subsequent development of the system or system component".

[IEEE Std]

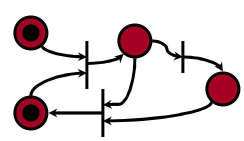




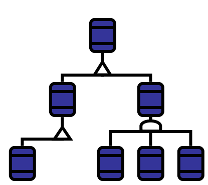

# Traditional RE focuses on modelling



Structured analysis



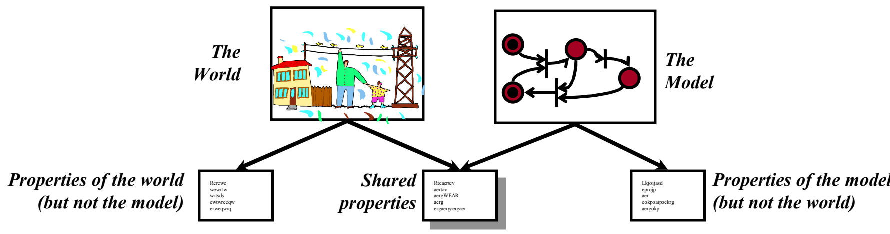
Behavioural analysis



Object models

# But "modelling" is not enough

- S/W modelling is a technical activity…
  - preciseness
  - completeness
  - consistency
- …but RE is a social activity…
- …and all models are approximations:

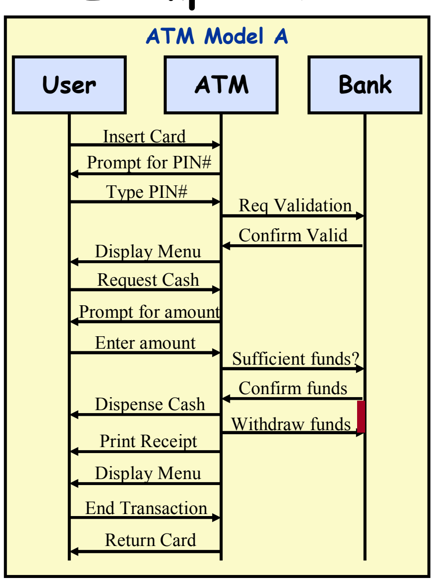


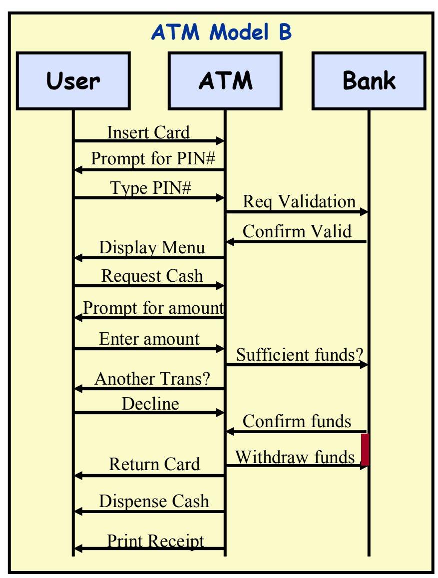

  - models of the social world are inherently subjective
  - and we have little scope for empirical validation





# Example: which is the better model?



# RE: A Roadmap

## How do we gather requirements information?

- Identify:
  - boundaries, stakeholders, views, goals, scenarios
- Techniques:
  - Interviews/questionnaires/focus groups for large user bases
  - Ethnographic techniques for socially-embedded systems
  - Prototyping and participatory design for poorly understood systems

## How do we analyze this information?

- Can't escape some modelling…

## How do we get agreement about the requirements?

- Validate models by making observations of the world
- Support negotiation where there are divergent views/goals

## How do we communicate the requirements?

- Careful mix of natural and formal languages
  - formal languages are precise and unambiguous
  - natural languages are more readable
- Traceability: forward and backward

## How do we keep the requirements up to date?

- Manage change as the real world needs change
- Manage variations across product families





# Requirements Elicitation

## → Starting point

- Some notion that there is a "problem" that needs solving
  - e.g. dissatisfaction with the current state of affairs
  - e.g. a new business opportunity
  - e.g. a potential saving of cost, time, resource usage, etc.
- A Requirements Engineer is an agent of change

## → The requirements engineer 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)
  - How might a software system help? (collect some Scenarios)
  - When does it need solving? (identify Development Constraints)
  - What might prevent us solving it? (identify Feasibility and Risk)
- elicit enough knowledge
  - ...sufficient to analyze requirements for validity, consistency, completeness, etc.
- i.e. become an expert in the problem domain
  - although ignorance is important too [Berry]

# Difficulties of Elicitation

- **Thin spread of domain knowledge**
  - The knowledge might be distributed across many sources
    - It is rarely available in an explicit form (I.e. not written down)
  - There will be conflicts between knowledge from different sources
    - People have conflicting goals
    - People have different understandings of the problem
- **Tacit knowledge (The "say-do" problem)**
  - People find it hard to describe knowledge they regularly use
    - Descriptions may be inaccurate rationalizations of expert behaviour
- **Limited Observability**
  - The problem owners might be too busy solving it using the existing system
  - Presence of an observer may change the problem
    - E.g. the Probe Effect and the Hawthorne Effect
- **Bias**
  - People may not be free to tell you what you need to know
    - Political climate & organisational factors matter
  - People may not want to tell you what you need to know
    - The outcome will affect them, so they may try to influence you (hidden agendas)






# Elicitation Techniques

- **Traditional Approaches**
  - Introspection
  - Existing Documents
  - Data Analysis
  - Interviews
    - Open-ended
    - Structured
  - Surveys / Questionnaires
  - Group elicitation
    - Focus Groups
    - Brainstorming
    - JAD/RAD workshops
  - Prototyping
- **Representation-based approaches**
  - Goal-based
  - Scenario-Based
  - Use Cases
- **Contextual (social) approaches**
  - Ethnographic techniques
    - Participant Observation
    - Enthnomethodology
  - Discourse Analysis
    - Conversation Analysis
    - Speech Act Analysis
  - Participatory Design
  - Sociotechnical Methods
    - Soft Systems Analysis
- **Cognitive approaches**
  - Task analysis
  - Protocol analysis
  - Knowledge Acquisition Techniques
    - Card Sorting
    - Laddering
    - Repertory Grids
    - Proximity Scaling Techniques

# Software Requirements Specification

- How do we communicate the Requirements to others?
  - It is common practice to capture them in an SRS
    - But an SRS doesn't need to be a single paper document...
- Purpose
  - Communicates an understanding of the requirements
    - explains both the application domain and the system to be developed
  - Contractual
    - May be legally binding!
    - Expresses an agreement and a commitment
  - Baseline for evaluating subsequent products
    - supports system testing, verification and validation activities
    - should contain enough information to verify whether the delivered system meets requirements
  - Baseline for change control
    - requirements change, software evolves
- Audience
  - Users, Purchasers
    - Most interested in system requirements
    - Not generally interested in detailed software requirements
  - Systems Analysts, Requirements Analysts
    - Write various specifications that inter-relate
  - Developers, Programmers
    - Have to implement the requirements
  - Testers
    - Determine that the requirements have been met
  - Project Managers
    - Measure and control the analysis and development processes





# Desiderata for Specifications

**Source:** *Adapted from Blum 1992, pp164-5 and the IEEE-STD-830-1993*

- Valid (or "correct")
  - Expresses actual requirements
- Complete
  - Specifies all the things the system must do
  - ...and all the things it must not do!
  - Conceptual Completeness
    - E.g. responses to all classes of input
  - Structural Completeness
    - E.g. no TBDs!!!
- Consistent
  - Doesn't contradict itself
    - I.e. is satisfiable
  - Uses all terms consistently
  - Note: inconsistency can be hard to detect
    - especially in timing aspects and condition logic
    - Formal modeling can help
- Necessary
  - Doesn't contain anything that isn't "required"
- Unambiguous
  - Every statement can be read in exactly one way
  - Clearly defines confusing terms
    - E.g. in a glossary
- Verifiable
  - A process exists to test satisfaction of each requirement
  - *"every requirement is specified behaviorally"*
- Understandable (Clear)
  - E.g. by non-computer specialists
- Modifiable
  - It must be kept up to date!

# Typical mistakes

- Noise
  - the presence of text that carries no relevant information to any feature of the problem.
- Silence
  - a feature that is not covered by any text.
- Over-specification
  - text that describes a feature of the solution, rather than the problem.
- Contradiction
  - text that defines a single feature in a number of incompatible ways.
- Ambiguity
  - text that can be interpreted in at least two different ways.
- Forward reference
  - text that refers to a feature yet to be defined.
- Wishful thinking
  - text that defines a feature that cannot possibly be validated.
- Jigsaw puzzles
  - e.g. distributing requirements across a document and then cross-referencing
- Duckspeak requirements
  - Requirements that are only there to conform to standards
- Unnecessary invention of terminology
  - E.g., 'the user input presentation function', 'airplane reservation data validation function'
- Inconsistent terminology
  - Inventing and then changing terminology
- Putting the onus on the development staff
  - i.e. making the reader work hard to decipher the intent
- Writing for the hostile reader
  - There are fewer of these than friendly readers


***Source:*** *Adapted from Kovitz, 1999*




# Traceability Tools

- Approaches:
  - hypertext linking
    - hotwords are identified manually, tool records them
  - unique identifiers
    - each requirement gets a unique id; database contains cross references
  - syntactic similarity coefficients
    - searches for occurrence of patterns of words
- Limitations
  - All require a great deal of manual effort to define the links
  - All rely on purely syntactic information, with no semantics or context
- Examples
  - single phase tools:
    - TeamWork (Cadre) for structured analysis
  - database tools, with queries and report generation
    - RTM (Marconi)
    - SLATE (TD Technologies)
    - DOORS (Zycad Corp)
  - hypertext-based tools
    - Document Director
    - Any web browser
  - general development tools that provide traceability
    - RDD-100 (Ascent Logic) - documents system conceptual models
    - Foresight - maintains data dictionary and document management


***Source:*** *Adapted from Palmer, 1996, p372*

# Limitations of Current Tools

## ➜ Informational Problems

- Tools fail to track *useful* traceability information
  - e.g cannot answer queries such as "who is responsible for this piece of information?"
- inadequate pre-requirements traceability
  - "where did this requirement come from?"

## ➜ Lack of agreement…

- …over the quantity and type of information to trace

## ➜ Informal Communication

- People attach great importance to personal contact and informal communication
  - These always supplement what is recorded in a traceability database
- But then the traceability database only tells part of the story!
  - Even so, finding the appropriate people is a significant problem


**Source:** *Adapted from Gotel & Finkelstein, 1993, p100*




# Laws of Program Evolution

**Source:** *Adapted from Lehman 1980, pp1061-1063*

## ➜ Continuing Change

- Any software ***that reflects some external reality*** undergoes continual change or becomes progressively less useful
  - The change process continues until it is judged more cost effective to replace the system entirely

## ➜ Increasing Complexity

- As software evolves, its ***complexity*** increases…
  - …unless steps are taken to control it.

## ➜ Fundamental Law of Program Evolution

- Software evolution is self-regulating with statistically determinable trends and invariants

## ➜ Conservation of Organizational Stability

- During the active life of a software system, the work output of a development project is roughly constant (regardless of resources!)

## ➜ Conservation of Familiarity

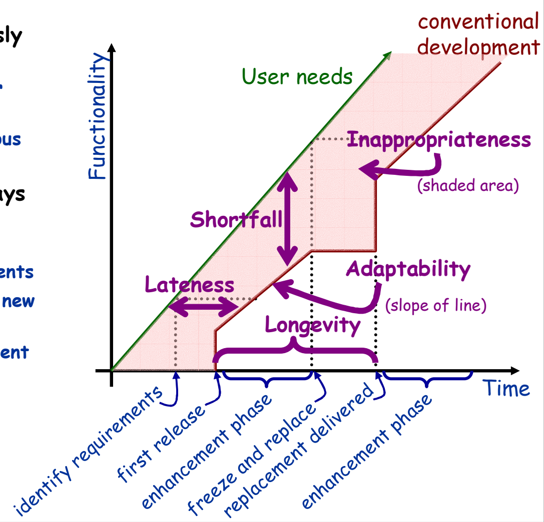
- During the active life of a program the amount of change in successive releases is roughly constant

# Requirements Growth

*Source: Adapted from Davis 1988, pp1453-1455*

## ➜ Davis's model:

- User needs evolve continuously
  - Represent this as a graph showing growth of needs over time
  - May not be linear or continuous (hence no scale shown)
- Traditional development always lags behind needs growth
  - first release implements only part of the original requirements
  - functional enhancement adds new functionality
  - eventually, further enhancement becomes too costly, and a replacement is planned
  - the replacement also only implements part of its requirements,
  - and so on...









# Summary

## ➜ Requirements Engineering is hard

- Junction of technical, social, and organisational worlds
- RE is about change, and change is politically sensitive
- And getting it wrong is expensive!

## ➜ Current challenges for RE

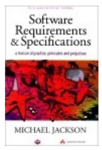
- Elicitation is a socially-embedded problem
- Communication is more than writing a specification
- Coping with change is a huge problem
- Requirements Engineering have to live with inconsistency

# Further Reading

B. A. Nuseibeh and S. M. Easterbrook, "Requirements Engineering: A Roadmap",

In A. C. W. Finkelstein (ed) "*The Future of Software Engineering*" ACM Press

http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf




Michael Jackson "Software Requirements & Specifications, a lexicon of practice, principles and prejudices". Addison-Wesley, 1995


Benjamin L. Kovitz "Practical Software Requirements A Manual of Content & Style". Manning, 1999

Book reviews at:
http://easyweb.easynet.co.uk/~iany/reviews/reviews.htm