If Software is the Solution, What is the Problem?

Bashar Nuseibeh

Computing Department

The Open University

Distinguished Lecture Series
St. Andrews, 1st December 2006

The Open University (OU)

- Founded in 1969 to widen access to higher education
  - No entry conditions (except for post-graduates)
  - Part-time, distance education
  - Inspired other similar universities around the world
- Over 200,000 students at any one time
  - 70% of students in full-time employment
    - 50,000 sponsored by their employer
- Mostly mature students, but
  - more younger students recently
    - 20% of undergraduates under 25

OU Student numbers

- First students in 1971: 25,000
  - 130,000 total in other universities
- Since then over 2 million students; currently
  - 150,000 UG and 30,000 PG students
  - 25,000 overseas students
  - 10,000 students with disabilities
- Among world's 20 largest universities by student number and the UK's largest

OU in Scotland

- 13 OU Regional Centres in the UK
- Scottish regional centre in Edinburgh
  - Supporting 15600 students
  - Supported by
    - 500 tutors
    - 87 members of staff
- http://www3.open.ac.uk/near-you/scotland/

Computing at the OU

- **Teaching:** Department of Computing
  - 43 academics, 14 staff tutors, 4000 students
- **Research:** Centre for Research in Computing
  - Department of Computing
  - Knowledge Media Institute (KMi)
  - Institute for Educational Technology (IET)
- Research Areas
  - Software Engineering
  - Human-Computer Interaction
  - Computational Linguistics and Information Retrieval
  - Knowledge Technologies

Today's Three Lectures …

- **10:00-11:00**
  - A roadmap of requirements engineering
- **11:30-12:30**
  - Problem-oriented requirements engineering
- **14:00-15:00**
  - Security requirements engineering

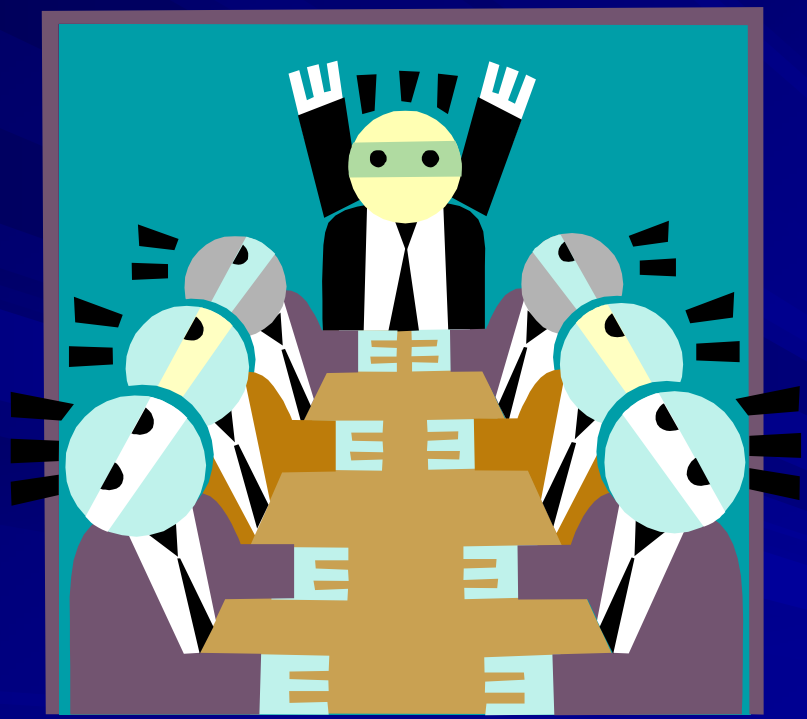
Warning: these lectures contains no explicit descriptions of programs or code, which some members of the audience may find disturbing. Viewer discretion is advised.

Lecture 1

A Roadmap of Requirements Engineering

… and some detours

The “voice of the customer”



A story that's probably not true

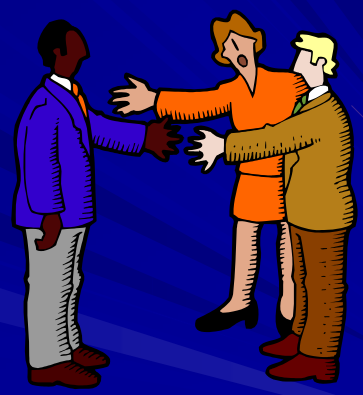
At the height of the space race between the US and the USSR in the 1960's, there was a requirement for a pen that worked in zero gravity.

To meet this requirement, NASA spent a considerable amount of money developing such a pen that was hailed by Americans as a great success.

The Russians faced with the same problem, used a pencil!

Requirements Engineering (RE)

- **Requirements** are:
  - expressions of **stakeholder needs** of a **system** to achieve particular **goals**.
  - expressed in the vocabulary of the **problem domain**, rather than the system (solution) domain.



- **Requirements Engineering** is about:
  1. **Discovering** stakeholder goals, needs, and expectations
     - » **Adjusting** stakeholder expectations
  2. **Communicating** these to system implementers
     - » **Adjusting** implementer expectations

A Roadmap of RE

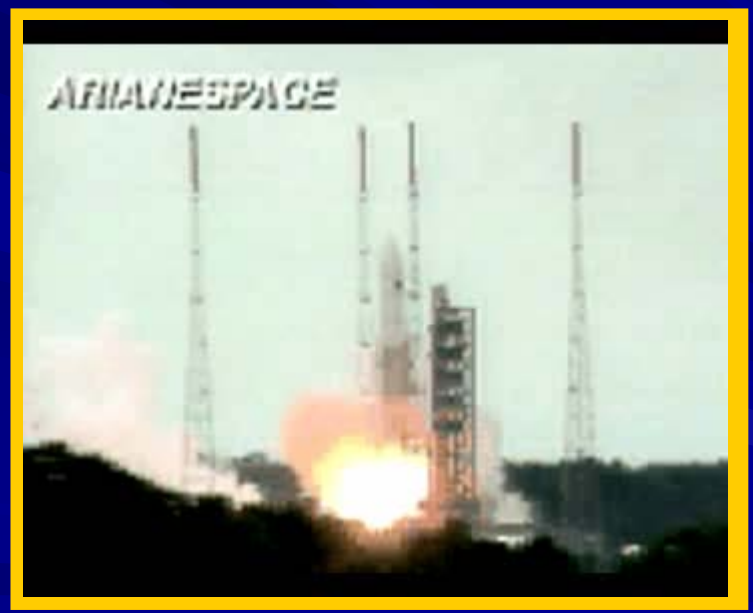
- **A little (more) motivation**
  - Or, why RE is important
- **A little background**
  - Or, before we begin RE
- **A roadmap**
  - Or, what is RE?
- **"You are here"**
  - Or, the RE state-of-the-art
- **A little speculation**
  - Or, where to go from here …

Motivating requirements engineering …

Motivation – Part 1: Scare Tactics

Many software failures can be attributed to ineffective requirements engineering

Ariane 5:



So, who dunnit?

Motivation – Part 1: Scare Tactics

- If you don't do RE, your software will fail …
  - Many software failures can be attributed to failure to do RE effectively.

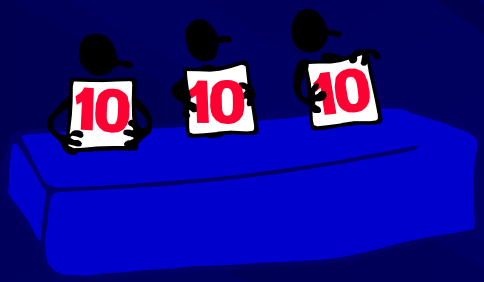
Ariane 5:

Spectacular failures almost always happen for systemic reasons.

Motivation – Part 2: Economics

- RE saves you money …
  - » Errors found 'earlier' in the software development life cycle are cheaper and easier to fix than those found later in the development life cycle [Boehm].

The studies that make this claim also assume a waterfall life cycle.



Motivation – Part 3: Quality

- RE helps you build better products …
  - that will satisfy your customer,
  - (and therefore make you money).

- This is an engineering argument because it addresses:
  - Fitness for purpose, as expressed by stakeholders

The Bottom Line

"If you build software without [requirements and] specifications, it can never be incorrect – it can only be surprising."

B. Kernighan

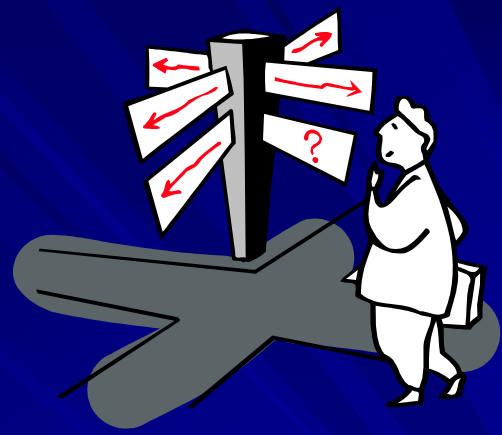
So, what is requirements engineering?

A Definition of RE

"**Requirements engineering** is the branch of **systems engineering** concerned with the real-world **goals** for, **services** provided by, and **constraints** on a large and complex software-intensive system. It is also concerned with the relationship of these factors to precise **specifications** of system **behaviour**, and to their **evolution** over time and across **system families**."

[adapted from Zave 1997]

Orientation

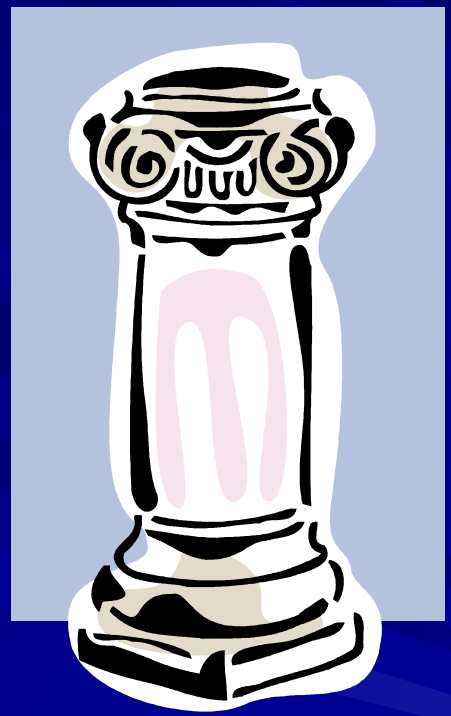


- Foundations
- Context and Groundwork
- Eliciting Requirements
- Modelling and Analysing Requirements
- Communicating Requirements
- Agreeing Requirements
- Evolving Requirements

Based on: B. Nuseibeh and S. Easterbrook, Requirements Engineering: A Roadmap Proceedings of International Conference on Software Engineering (ICSE-2000), The Future of Software Engineering, A. Finkelstein (ed.), 4-11 June 2000, Limerick, Ireland, ACM Press.

Foundations of RE

- Computer Science
- Logic
- Linguistics
- Systems Theory
- Cognitive Psychology
- Anthropology
- Sociology
- Philosophy … epistomology… phenomenology …ontology…



Context and Groundwork

- **Context**
  - Organisational setting
  - Contract and procurement procedures
  - Process improvement and maturity
  - Personnel and staffing

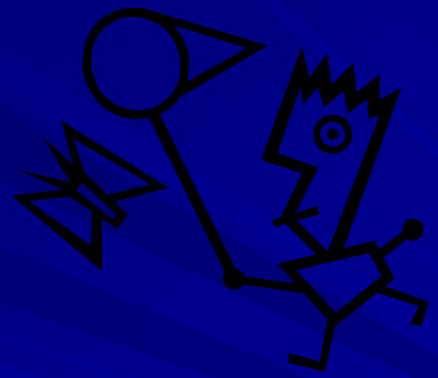
- **Groundwork**
  - Feasibility
  - Risk

[from Finkelstein 1993]

Eliciting Requirements – what & where

- Requirements elicitation is partly a process of **discovering** stakeholder expectations, and **adjusting** these expectations.

- **Things to elicit**
  - Boundaries
  - Stakeholders
  - Goals
  - Tasks … use cases … scenarios
  - Feasibility
  - Risk



- **Where to elicit requirements from**
  - Stakeholders
  - Application domain
  - Existing documentation

Eliciting Requirements - how

- **Traditional techniques**
  - Questionnaires, surveys, interviews, analysis of existing documentation, etc.
- **Group elicitation techniques**
  - Brainstorming, focus groups, RAD/JAD workshops, etc.
- **Prototyping**
  - For early feedback from stakeholders
- **Model-driven techniques**
  - Goal-based, use case/scenario-based, etc.
- **Cognitive techniques**
  - Protocol analysis, card sorting, laddering, etc.
- **Contextual techniques**
  - Ethnography, conversation analysis, etc.

Modelling and Analysing Requirements

- **Enterprise** modelling
- **Data** modelling
- **Behavioural** modelling
- **Domain** modelling
- Modelling **non-functional requirements** (NFRs)
- **Analysing** Requirements Models
  - Animation
  - Automated reasoning
  - Consistency checking
  - …

Detour 1: From Fuzzy to Formal

"Everybody loves my baby ... but my baby loves only me"

- **Formalisation**

  $\forall$ x · Loves (x, MyBaby) // Formalise Line 1 of song

  $\forall$ y · Loves (MyBaby, y) $\rightarrow$ y = Me // Formalise Line 2 of song

- **Analysis**

  $\forall$ x · Loves (x, MyBaby)

  Loves (MyBaby, MyBaby)

  $\forall$ y · (Loves (MyBaby, y) $\rightarrow$ y = Me)

  Loves (MyBaby, MyBaby) $\rightarrow$ MyBaby = Me

Conclusion: I am my baby !

Example due to Gries

A ‘formal’ specification

- Rule:
  - All departmental visitors give invited lectures
- Fact:
  - Bashar is a departmental visitor
- Observation:
  - Bashar gives an invited lecture

Formal Analysis

Allows the requirements engineer to ask about properties of a software system to be developed.

Three interesting kinds of formal analysis:

Deduction Induction Abduction

(Natural) Deduction

- Rule:
  All departmental visitors give invited lectures
- Fact:
  Bashar is a departmental visitor

- **Deduction concludes that:**
  Bashar gives an invited lecture

Induction (Learning)

- Fact:
  Bashar is a departmental visitor
- Observation:
  Bashar gives an invited lecture

Induction learns the rule that:
All departmental visitors give invited lectures

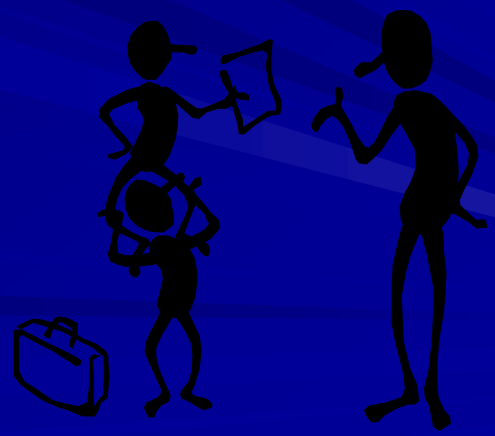
Abduction (Explanation)

- Rule:
  All departmental visitors give invited lectures
- Observation:
  Bashar gives an invited lecture

- **Abduction explains the fact that:**
  Bashar is a departmental visitor

Communicating Requirements

- **RE facilitates communication among stakeholders**
- Requirements **documentation**
  - is often the focus of such communication
  - affects choice of **specification language**
  - sometimes makes use of documentation **standards**
- Requirements **traceability**
- Requirements **management**



Agreeing Requirements

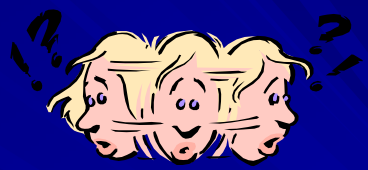
- To design and implement a system, the requirements have to be **agreed**.

- To get agreement requirements have to be
  - **Validated**
  - **Negotiated**, and **conflicts** resolved
  - **Prioritised**

Living with Inconsistency

Detour 2: Living with Inconsistency

- Rule:
  All departmental visitors give invited lectures
- Fact:
  Bashar is a departmental visitor
  Bashar is NOT a departmental visitor



} Inconsistency!

- **What can we conclude???**
  Does: Bashar gives an invited lecture … or NOT?

Inconsistency: Live and Let D.A.I.

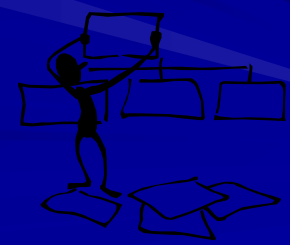
Deduction
(Reasoning about Inconsistency)

Abduction
(Explaining Inconsistency)

Induction
(Learning from Inconsistency)

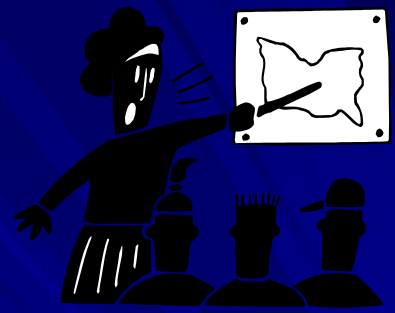
Evolving Requirements

- **Successful systems will evolve**
  - When the environment in which they operate changes
- **Managing change** is a fundamental RE activity
  - **Adding** new requirements & requirements **scrubbing**
  - **Fixing errors** & **managing inconsistency**
  - **Impact analysis** & **configuration management**
- Requirements for **product families, COTS & Services**
  - Identify **core** requirements
  - **Reuse** requirements
  - Match requirements to software **architectures**



So, where are we in terms of state-of-the-art?

You Are Here!



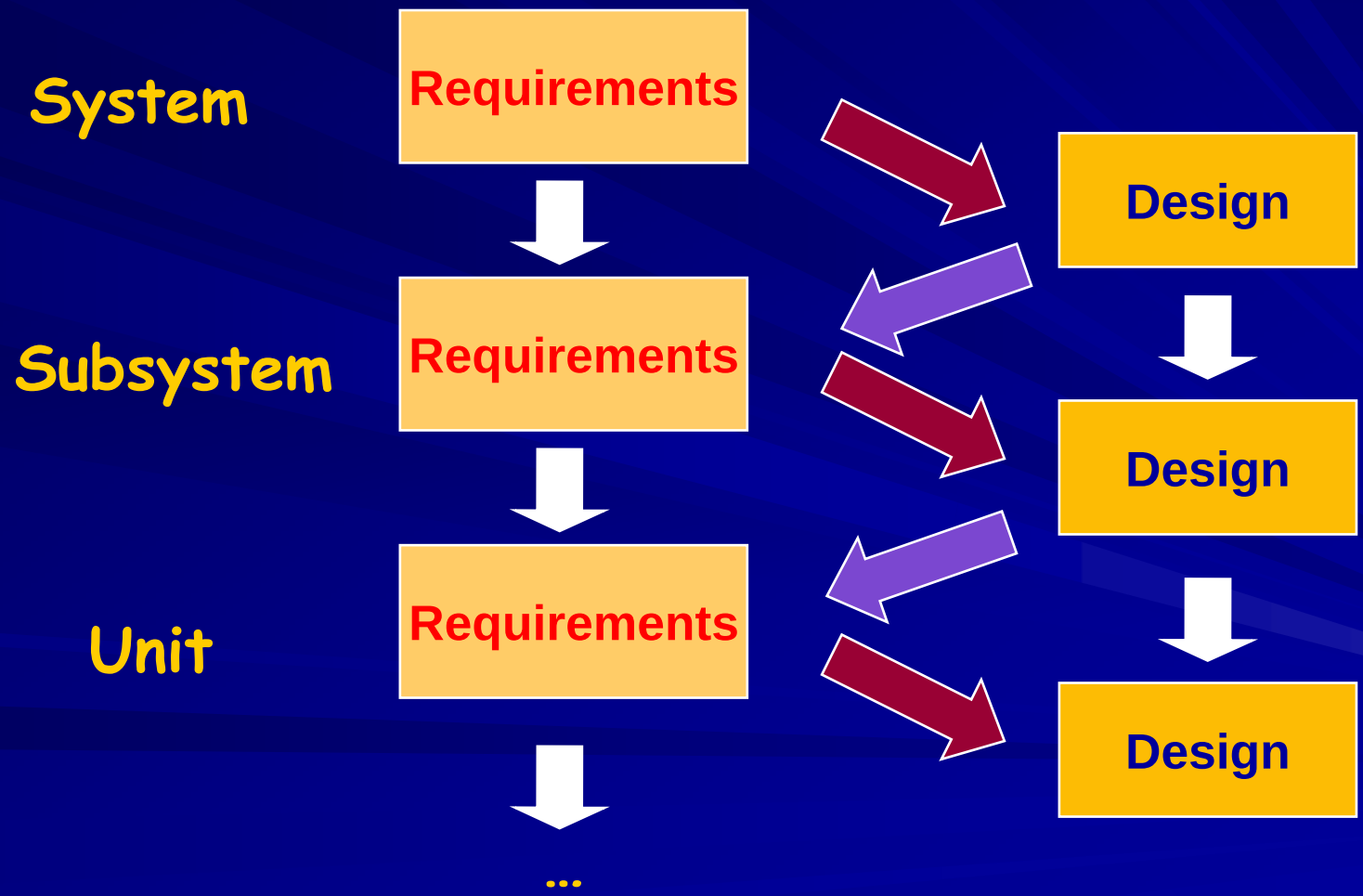
- Modelling in **context**
- Describing indicative and optative properties of the **environment**
- **Inconsistency happens, live with it!**

- **The RE Community:**
  - REJ, RE Conference, REFSQ, AWRE…
  - In the UK: BCS RESG (www.resg.org.uk)

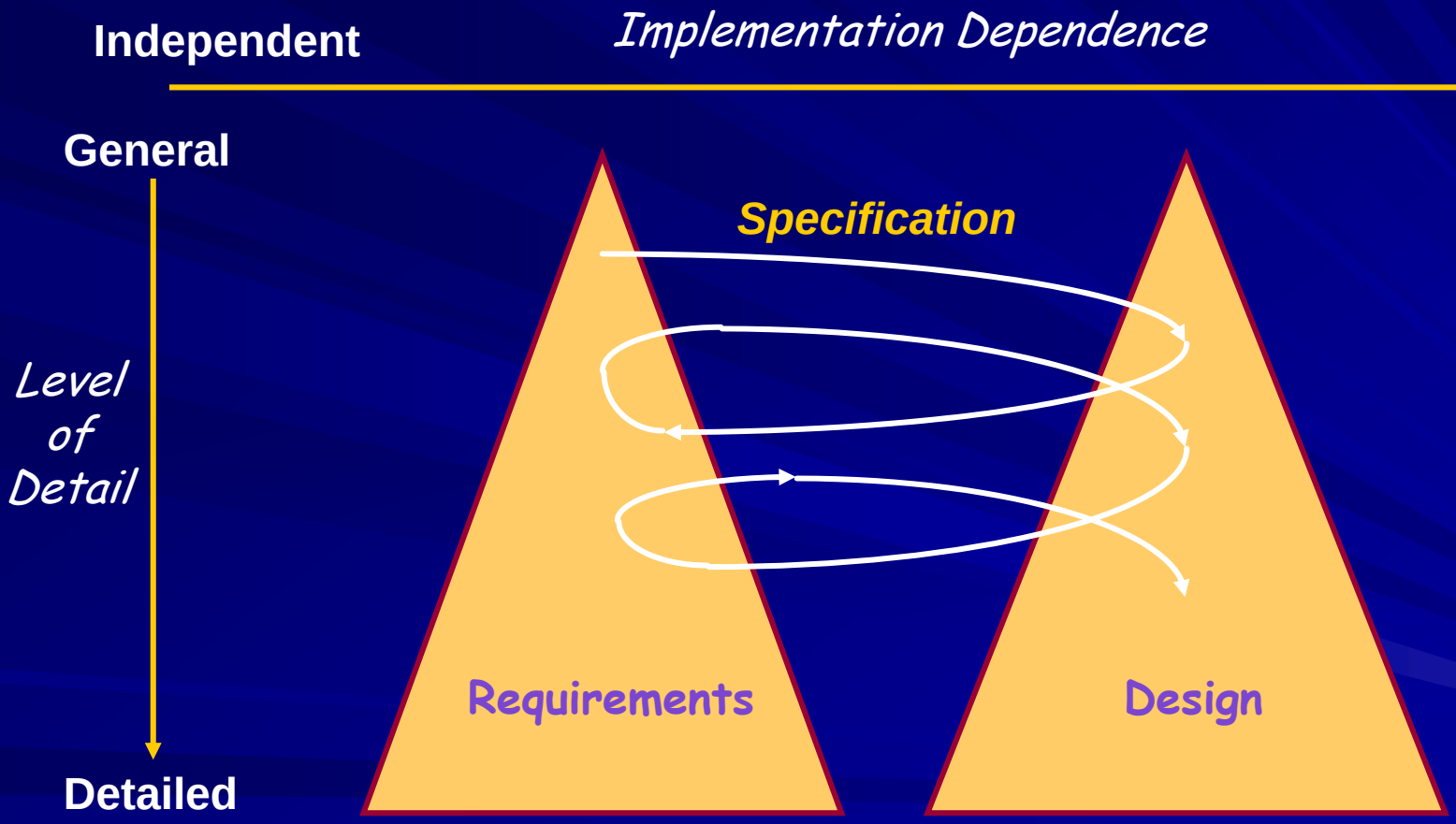
Journey Planner – a wish list

- Richer models for capturing and analysing **non-functional requirements**.
- Techniques for modelling and analysing **properties of the environment**
  - to deal with incomplete, inconsistent & evolving models
  - To deal with a changing environment (e.g. mobility context)
- **Reuse** of requirements models.
  - to adapt products into product families
- **Bridging the gap between elicitation approaches** based on contextual enquiry and more formal specification and analysis approaches.

Detour 3: Requirements & Design

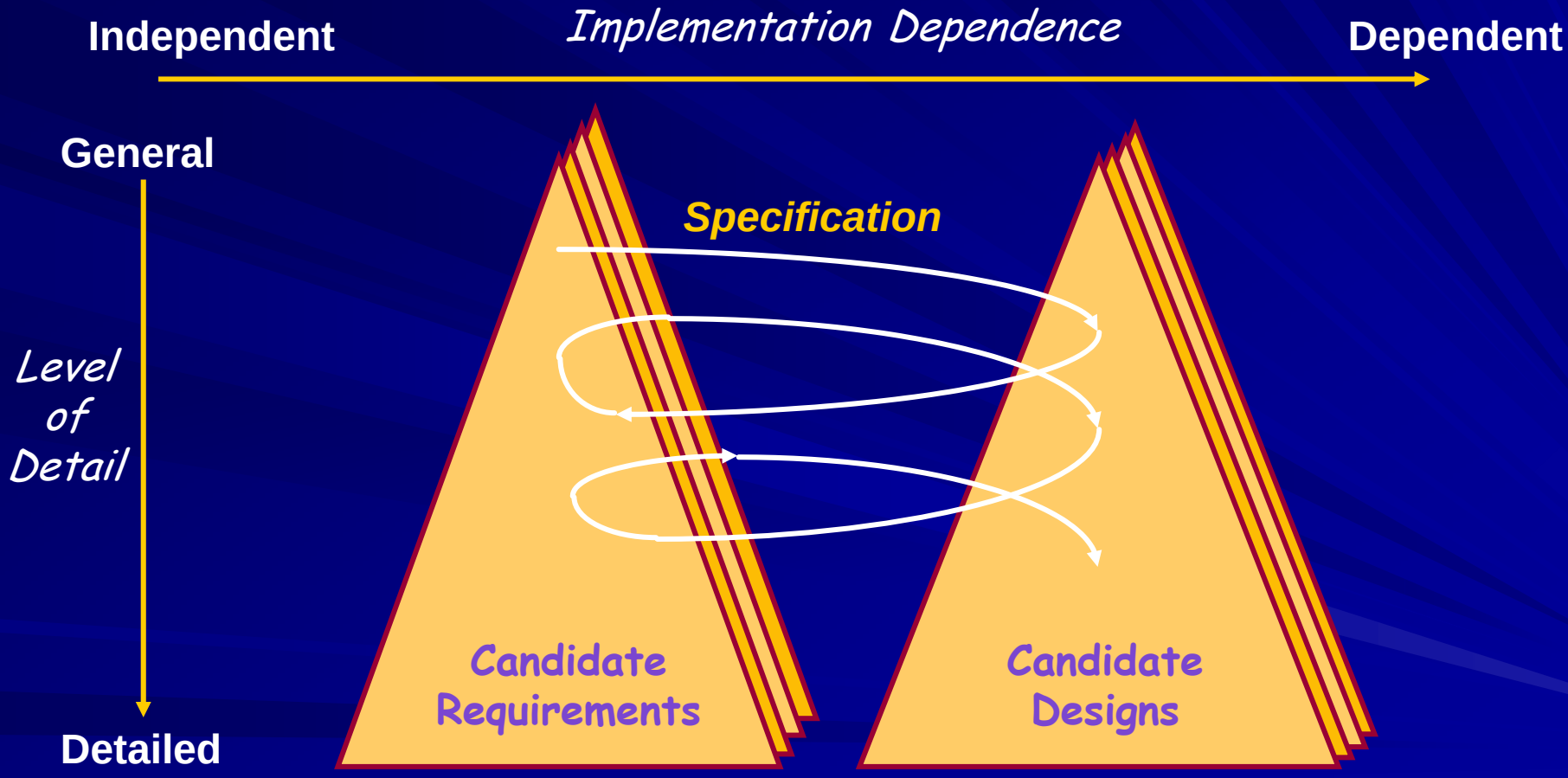


Twin Peaks: A finer grain process?



[B.Nuseibeh, IEEE Computer, 34(3):115-117, March 2001]

Mountain Range: exploring alternatives



Some difficult questions

- **What is a requirements engineer?**
  - A software architect?
  - A systems engineer?
  - An anthropologist?
  - … ?
- **The end of RE, as we know it?**
  - Refinement – not realistic?
  - Documentation – not necessary?
  - Time scales – too long?

A final thought …

Consider the following two projects:

- Project 1: completed on time, but
  - Estimated cost: **$4M** → actual cost: **$9M**
  - **Post release: 30%** additional performance developed
  - Annual **maintenance** costs: **$3M**
- Project 2:
  - Budgeted time to develop: **5 years** → actual time: **14 years**
  - Estimated **cost: $7M** → actual cost: **$102M**
  - Post release: **$40M** of adaptive maintenance costs
  - Current (preventative) **maintenance**: **$20M** over 10 years.

Are these projects successes or failures?

- In software engineering, they would be used as illustrations of the **'software crisis'**.
- The projects are actually regarded as great **examples of civil engineering success:**

Project 1

Project 2

Summary: RE Rules OK!

RE can help discover, adjust, and communicate user expectations of software, leading to high(er) quality systems that are fit for purpose.

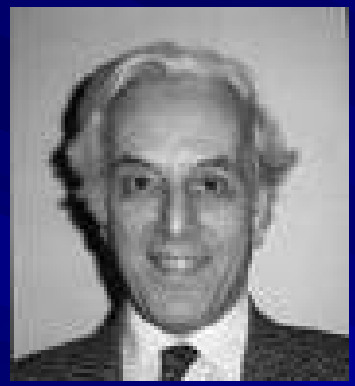
Lecture 2

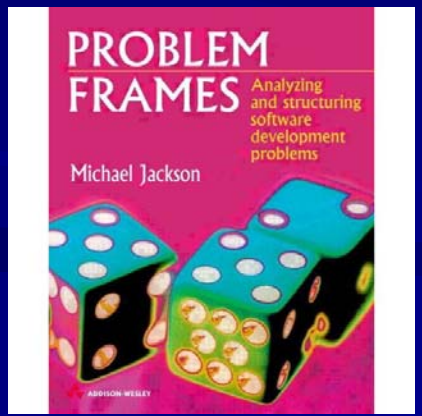
Problem-Oriented Requirements Engineering

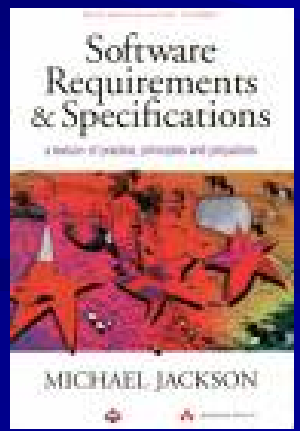
… requirements and specifications

References

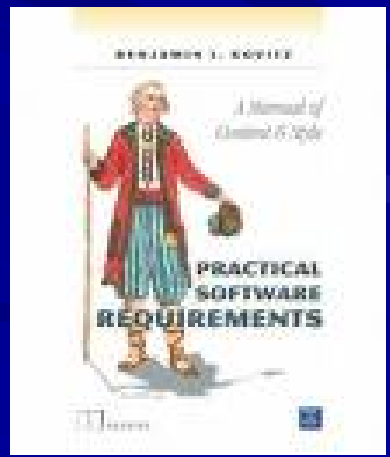
Michael Jackson







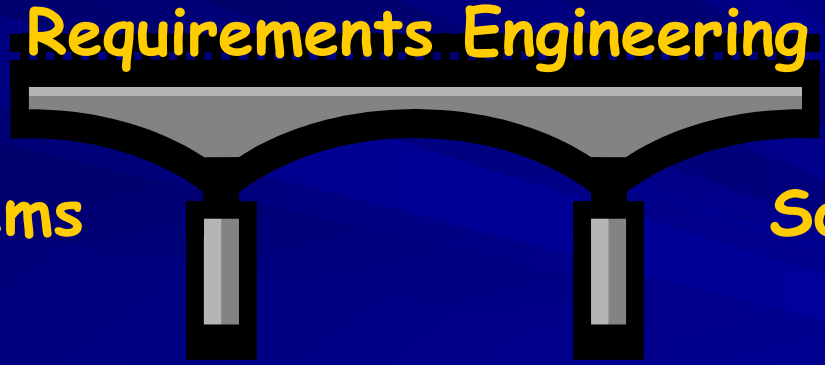
Ben Kovitz



The big picture

Software Systems Engineering

Requirements Engineering



Problems

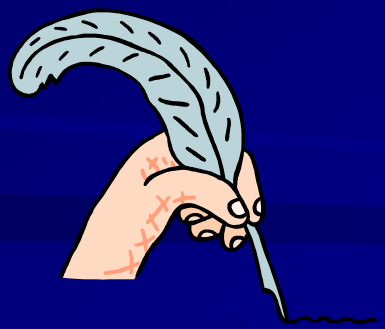
Solutions

People – and how to please them

A Perspective on Software Engineering

Behaviour

Descriptions



Writing

A Discipline of Description

A Problem Specification

It is necessary to transport an egg over a distance of at least 1 metre without direct intervention. The egg must not be broken or cracked. The egg must not make contact with the ground. No person is allowed within 1 metre of the stopping point of the egg.

Types of Specification

- **Requirements Specification**
  - Details the concerns of customers and users
  - Defines functions to be performed, and constraints
- **System Specification**
  - Defines a system boundary and interactions between the system and its environment (i.e. a “black box” view)
- **Architectural Design Specification**
  - Identifies the major subsystems, and interactions between them
  - Allocates functional requirements to subsystems
- **Detailed Design Specification**
  - Describes the details of the decomposed components of a system

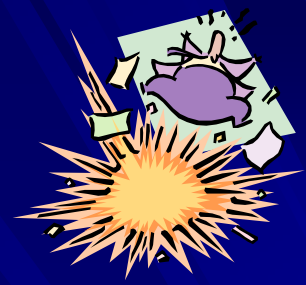
Roles of Specifications

- A **contract**
  - Specifies a job to be done
  - Acts as a basis for judging completion of the job (and hence payment!)
- A **communication medium**
  - Conveys and understanding of the domain
  - Passes information between different teams in the software development process
- A **statement of commitment**
  - Whether legally binding or otherwise

Audience for Requirements Specifications

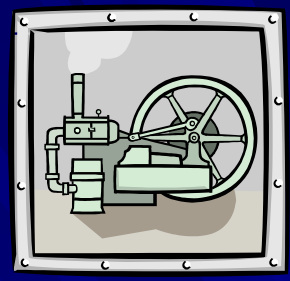
- **Users, Purchasers**
  - Most interested in system requirements
  - Not generally interested in detailed software requirements
- **Systems Analysts, Requirements Analysts**
  - Write various specifications that inter-relate
- **Developers, Programmers**
  - Have to implement the requirements
- **Testers**
  - Determine that the requirements have been met
- **Project Managers**
  - Measure and control the analysis and development processes

Specification Perils



- **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 some 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 can not possible be validated.

The World and the Machine

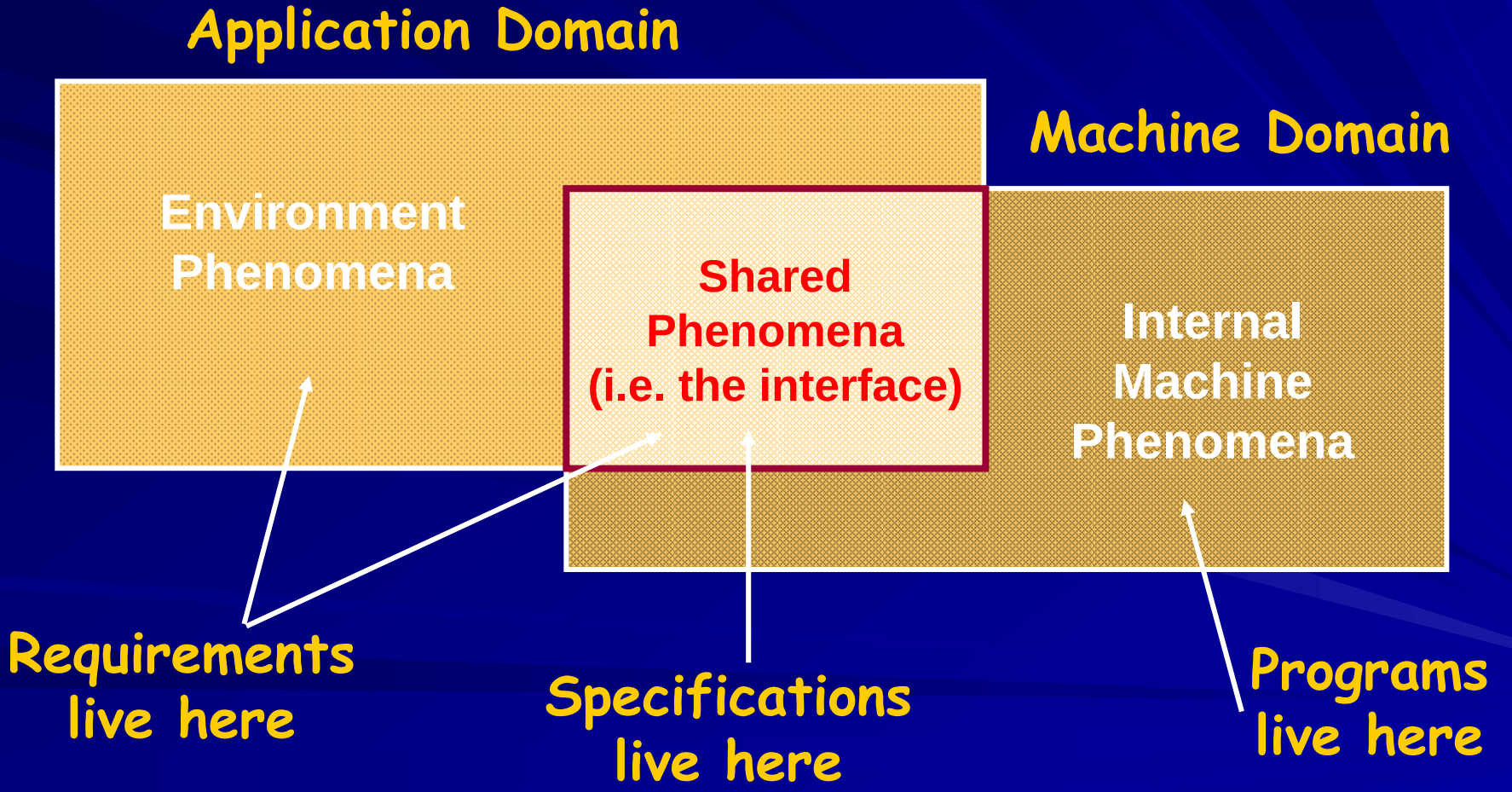


- The **Machine**
  - We are interested in software systems
  - We will call the software system to be developed the ‘machine’
  - The hardware exists only to run the software, hence it is also part of the machine

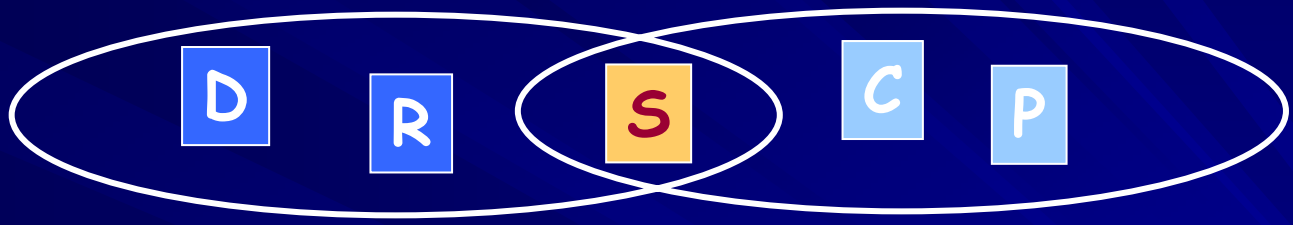
- The **Application Domain**
  - A machine will interact with its environment
  - A machine is built to serve some purpose in the world
  - The aspect of the environment that defines the machine’s purpose is it’s application domain
  - The application domain is often a human activity system

[Adapted from **Jackson 1995**, p.72]

A Little Phenomenology



Requirements as Application Phenomena



- **For a program to satisfy a requirement**, we need to consider:
  - The properties of the **computer (C)**
  - The properties of the **program (P)**
  - The properties of the **domain (D)** independent of the machine
  - The **requirements (R)** for the machine
  - The properties of the machine in the application domain; i.e. the **specification (S)**

- Demonstration that P satisfies R is then a two step process:
  - **Do C and P imply S? … verification**
  - **Do S and D imply R? … validation**

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 the wheels are turning
  - Wheels are turning if and only if moving on the runway
- *Specification S:*
  - Reverse thrust enabled if and only if wheel pulses are on
- *S + D imply R*
  - But what if the domain model is wrong?

In the mood

- **Mood** (of a verb):
  - **Indicative:** asserts a fact (“you sing”)
  - **Interrogative:** asks a question (“are you singing”)
  - **Imperative:** conveys a command (“Sing!”)
  - **Subjunctive:** states a possibility (“I might sing”)
  - **Optative:** expresses a wish (“may you sing”)
- **‘Shall’ and ‘will’ can be used in different moods:**
  - “I shall drown. No one will save me”
  - “I will drown. No one shall save me”
- For requirements engineering:
  - use the **indicative mood for domain properties**
  - use the **optative mood for requirements**

[Adapted from **Jackson 1995**, p.126]

Exercise

- In developing a system to control a lift, which of the following descriptions are indicative and which are optative:

  (a) The elevator never goes from the *n*th to the *n+2*th floor without passing the *n+1*th floor.

  (b) The elevator never passes a floor for which the floor selection light inside the car is illuminated without stopping at that floor.

  (c) If the motor polarity is set to *up*, and the motor switch setting changed from off to on the elevator starts to rise within 250ms.

  (d) If the *up* arrow indicator at a floor is not illuminated when the lift stops at the floor, it will not leave in an *upwards* direction.

  (e) The doors are never open at a floor unless the elevator is stationary at that floor.

  (f) When the elevator arrives at a floor, the *elevator-present* sensor at the floor is set to on.

  (g) If an *up* call button at a floor is pressed when the corresponding light is off, the light comes on, and remains on until the call is serviced by the elevator stopping at that floor and leaving in an *upwards* direction.

Descriptions

- A **designation**
  - singles out a phenomenon of interest; tells you how to recognise it; gives it a name
  - is always informal, as it maps from the fuzzy phenomena to formal language
- A **definition**
  - gives a formal definition of a term that may be used in other descriptions
  - can be more or less useful, but never right or wrong
- A **refutable description**
  - states some property of a domain that could in principle be refuted; might not be practical to refute it, but refutation should be conceivable
  - refutability depends on an appeal to the designated phenomena of the domain being described
- A **rough sketch**
  - is a tentative description that is being developed
  - may contain undefined terms

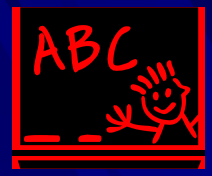
Examples

- Designation:
  - Mother(x, m) denotes that m is the genetic mother of x
- Definition:
  - Child (x, y) is defined as mother(y, x) or father (y, x)
- Refutable Description:
  - For all m and x, Mother(x, m) implies not(Mother(m, x))
- A rough sketch:
  - ‘Everyone really belongs to just one family’.

Natural Language

- Requirements specifications are often written in **natural language**
- Natural language is **accessible** to many people, and is often **suitable for expressing designations and rough sketches**.
- However, using natural language may make lead to specifications whose **consistency, correctness and completeness** is difficult to assess.

Some fun with natural language



- **Dry Cleaners Window**: 38 years on the same spot.
- **Clothes Shop:** Wonderful bargains for men with 16 and 17 necks.
- **Used Cars:** Why go elsewhere to be cheated? Come here first!
- **Clothes Factory:** We do not tear your clothing with machinery. We do it carefully by hand.
- **Jewellers:** Now is your chance to have your ears pierced and get an extra pair to take home too.
- **Church Bulletin:** Don't let worry kill you - let the church help.

Why Document?

- **Extends what the mind can grasp** and remember
- Gives the **same story** to each member of the team
- Introduces **new team members** to the project
- **Protects** intellectual equity
- Helps the writer to **better understand the problem**

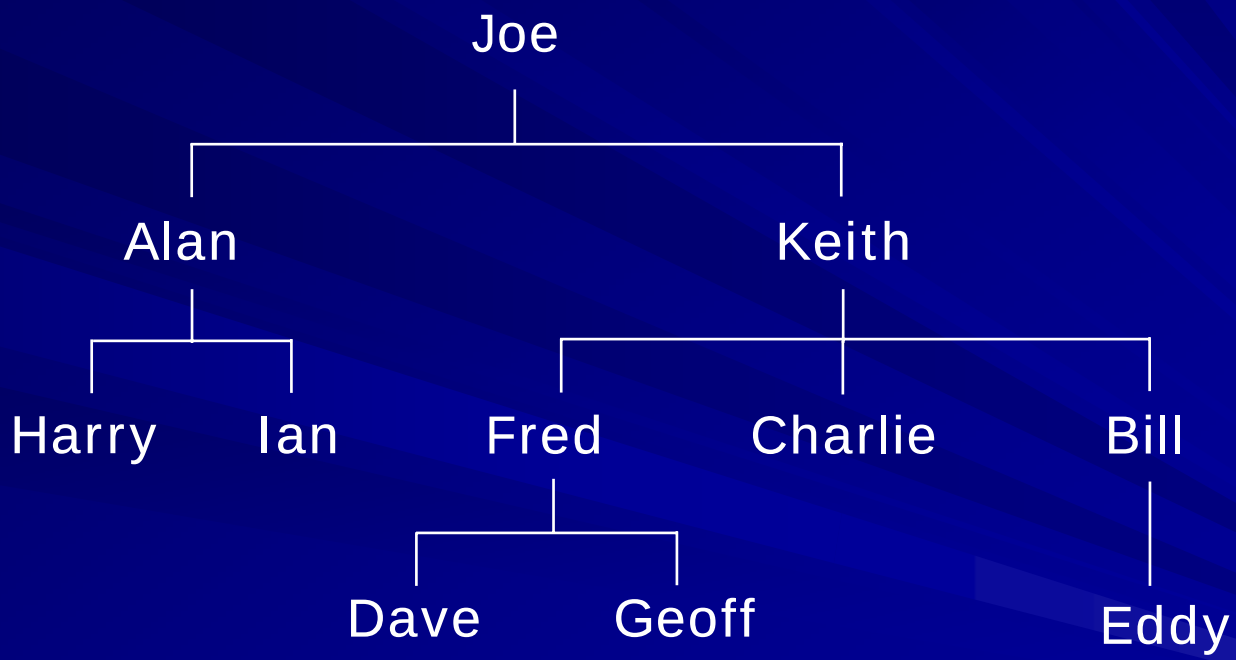
[From **Kovitz 1998**, Chapter 13]

Arboricide

- "Alan, Bill, Charlie, Dave, Eddy, Fred, Geoff, Harry, Ian, Joe and Keith are all related. Geoff's uncle's brother is Harry's cousin. Eddy's grandfather is Ian's uncle. Alan is not Fred's nephew. Harry's father is Keith's brother. Alan is older than Ian. Fred plays tennis with Charlie’s brother.”

- **“Who is Geoff’s cousin?”**

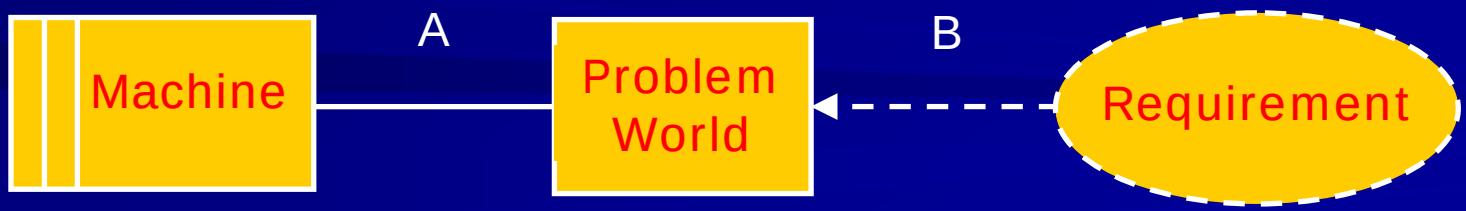
Arboricide: the Destruction of Trees



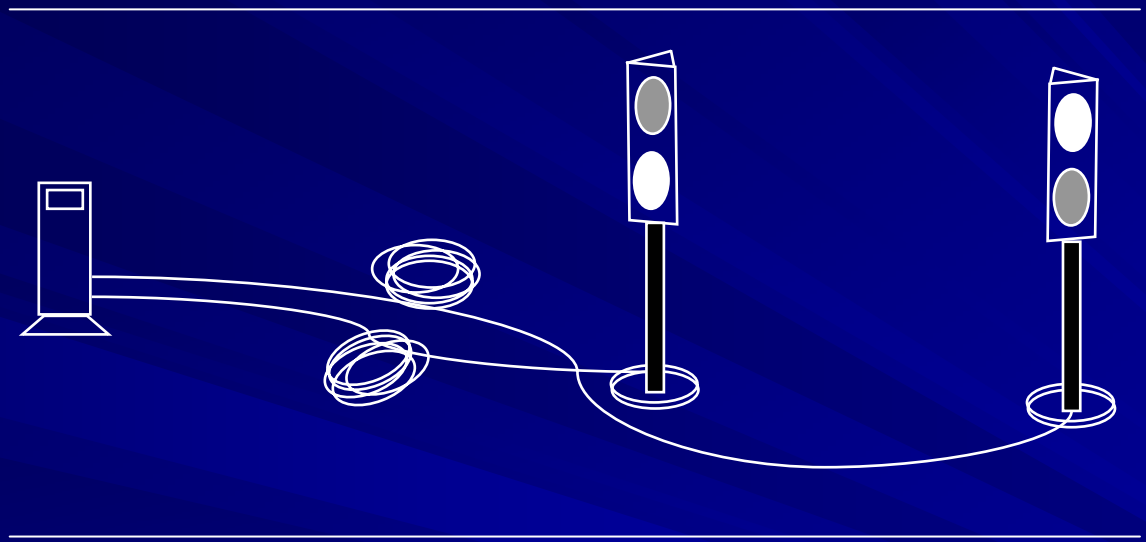
- “Who is Geoff’s cousin?”

From problem descriptions to problem structures: problem frames

- Machine and problem world are relative to problem
  - The machine is what we must build
  - The problem world is given
- The requirement is a condition on the problem world
  - The machine interacts with the problem world at A
  - The requirement is about the problem world in terms of phenomena B

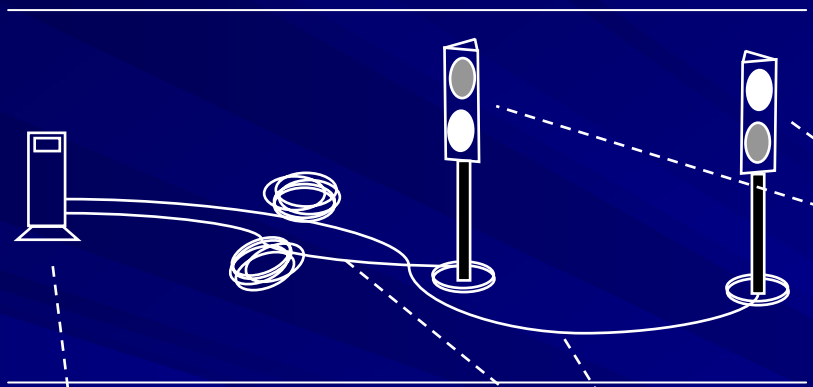


One-Way Traffic Lights: a Little Problem



- The lights are to be controlled so that they show Stop and Go in a specified sequence of phases of specified durations
- The computer can cause R and G pulses
  - But how are Stop and Go phenomena related to R and G?

Phenomena in the Problem

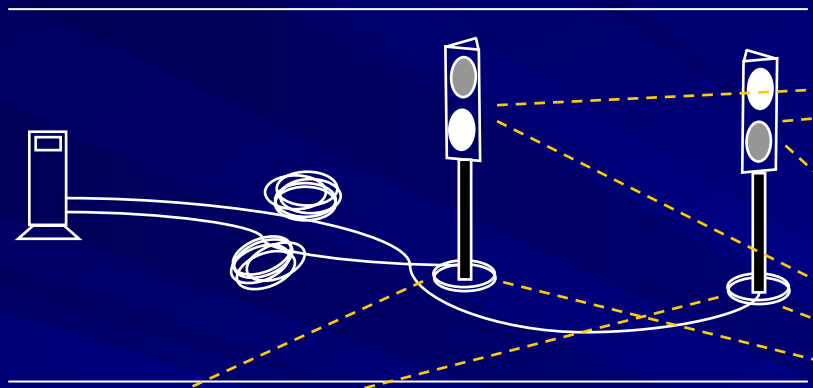


Private phenomena of the World (not shared with the Machine) e.g.: whether Stop or Go is showing

Shared phenomena (belonging both to the World and to the Machine) e.g.: R, G pulse events

Private phenomena of the Machine (not shared with the World) e.g.: program counter register, value of disk record

Descriptions in the Problem



Stop, Go states

R: requirement

(all phenomena of the World)

S: program specification

(phenomena shared by the Machine and the World)

R, G pulse events

R, G pulse events and Stop, Go states

D: domain properties

(all phenomena of the World)

D describes how the world is (indicative): how Stop and Go respond to the R and G pulses

R describes how we want the world to be (optative): desired sequence of Stop and Go lights

S describes how we want the interface to be (optative): eg “(R1; R2; wait 50; …)*”

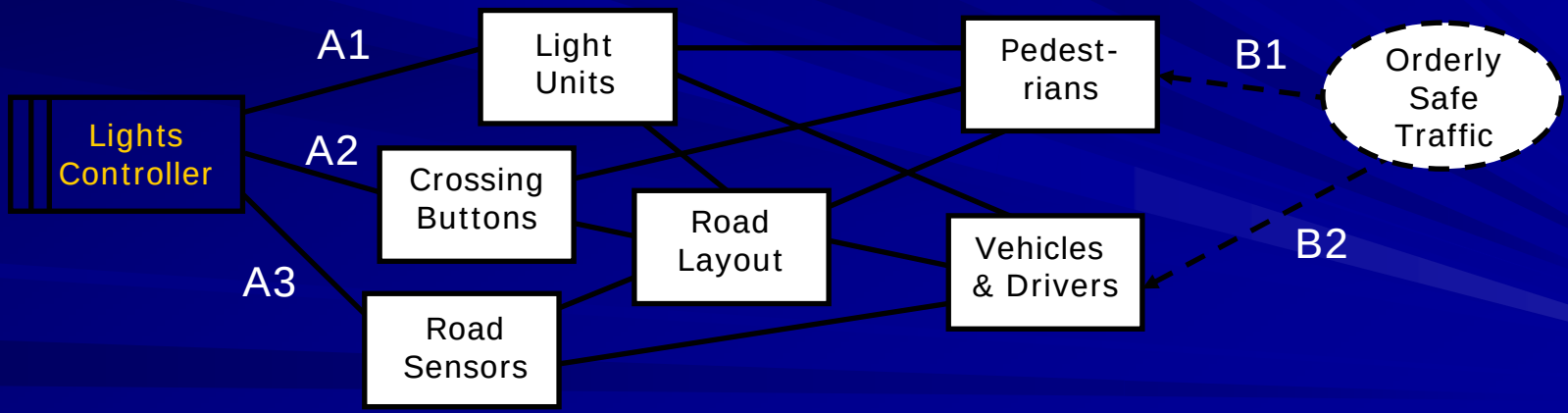
Eventually we must show that **S ,D |- R**

One-Way Traffic Lights:
Problem Diagram

Problem World Decomposition: An Example

- Controlling a complex traffic intersection with traffic lights, pedestrian crossings, road sensors

- The problem world:



- Problem world decomposition can open up design options

Problem Frames (types)

- Jackson identifies four types of simple problems which have an identifiable structure
  - Information Display
  - Workpieces
  - Commanded Behaviour
  - Required Behaviour

- The key is to try to decompose problems you don't understand into subproblems that you do understand, and for which there are known solutions.
  - http://en.wikipedia.org/wiki/Problem_Frames_Approach

Summary

- Specifications can provide precise descriptions that bridge the gap between problems and solutions.
- Specifications can have defects that are misleading and that need to be identified and addressed.
- Requirements (that live in the problem world) can be vague and difficult to analyse systematically.
- Problem structures can help clarify and organise requirements and the elements of the application domain to which they relate.

Lecture 3

Security Requirements Engineering

A security problem?

Requirements and Security Engineering

R. Crook, D. Ince, L. Lin, and B. Nuseibeh, Security Requirements Engineering: When Anti-requirements Hit the Fan, Proceedings of IEEE International Requirements Engineering Conference (RE'02), Essen, Germany, 9-13 September 2002.

Conclusions

- Many (but not all) security issues arise in the problem world, so we need rigorous problem analysis
  - **Security requirements** arise from such problem analysis
  - Analysing security requirements can benefit security analysis
- **Security requirements engineering** gives rise to research challenges:
  - Relating software and system security requirements
  - Relating security problems to security solutions
  - Understanding scope and context
  - Knowing when to stop

Ingredients of this talk

- ✓ A little bit of requirements engineering
- A little bit of security engineering
- A little bit of social engineering
- A question of software engineering?
- Some common sense
- A research agenda

A little bit of security…terminology

- **Security** is concerned with the **protection** of **assets** from **(intentional) harm**
  - **Protection:** achieved through prevention or prohibition
  - **Asset:** something in the system that has direct or indirect value
  - **Threat:** Harm that can happen to an asset
  - **Attack:** A threatening event
  - **Attacker:** The agent causing an attack (not necessarily human)
  - **Vulnerability:** a weakness in the system that makes an attack more likely to succeed

Security engineering

- A mature discipline with many techniques, mechanisms, and standards for implementing security
  - e.g., firewalls, cryptography, access control, etc.
- Security risk analysis and management

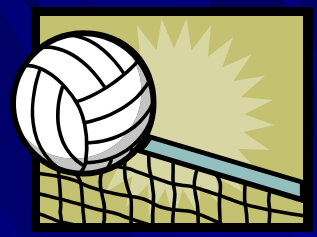
Security goals – CIA … A

- **Confidentiality** – ensure that an asset is visible only to actors authorized to see it.
- **Integrity** – ensure that the asset is not corrupted.
- **Availability** – ensure that the asset is readily accessible to agents that need it, when they need it
- **Authentication** – ensure that the identity of the asset or actor is known.
  - **… accountability … non-repudiation … authorisation …**

A wicked problem

- **Security is a ‘wicked problem’ [Rittel]**, for which there is no perfect solution;
  - security implementations are a trade-off between cost and effectiveness;
  - some assets are not worth protecting,
  - acceptable solutions vary from stakeholder to stakeholder,
  - the solution space is bounded by what the customer is willing to spend and what technology can provide.

Security is not football



- Do we need to model attackers in security analysis?
  - **Security is not a zero sum game**:
    - there is no exact equivalence between the losses incurred by the asset owner and the gains of the attacker.
  - So, the evaluation of possible harm to an asset can sometimes be carried out without reference to particular attackers; and
  - consideration of the goals of attackers cannot be used simply to arrive at the goals of a defender to prevent harm.

Security Requirements

- Security requirements may be usefully expressed as:
  - **constraints on functional requirements**
  - … in order to achieve security goals.

C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh, "A Framework for Security Requirements Engineering," in *Proceedings of the 2006 Software Engineering for Secure Systems Workshop (SESS'06), co-located with ICSE'06,* Shanghai China, 20-21 May 2006, pp. 35-42.

Core Security Requirements Artefacts

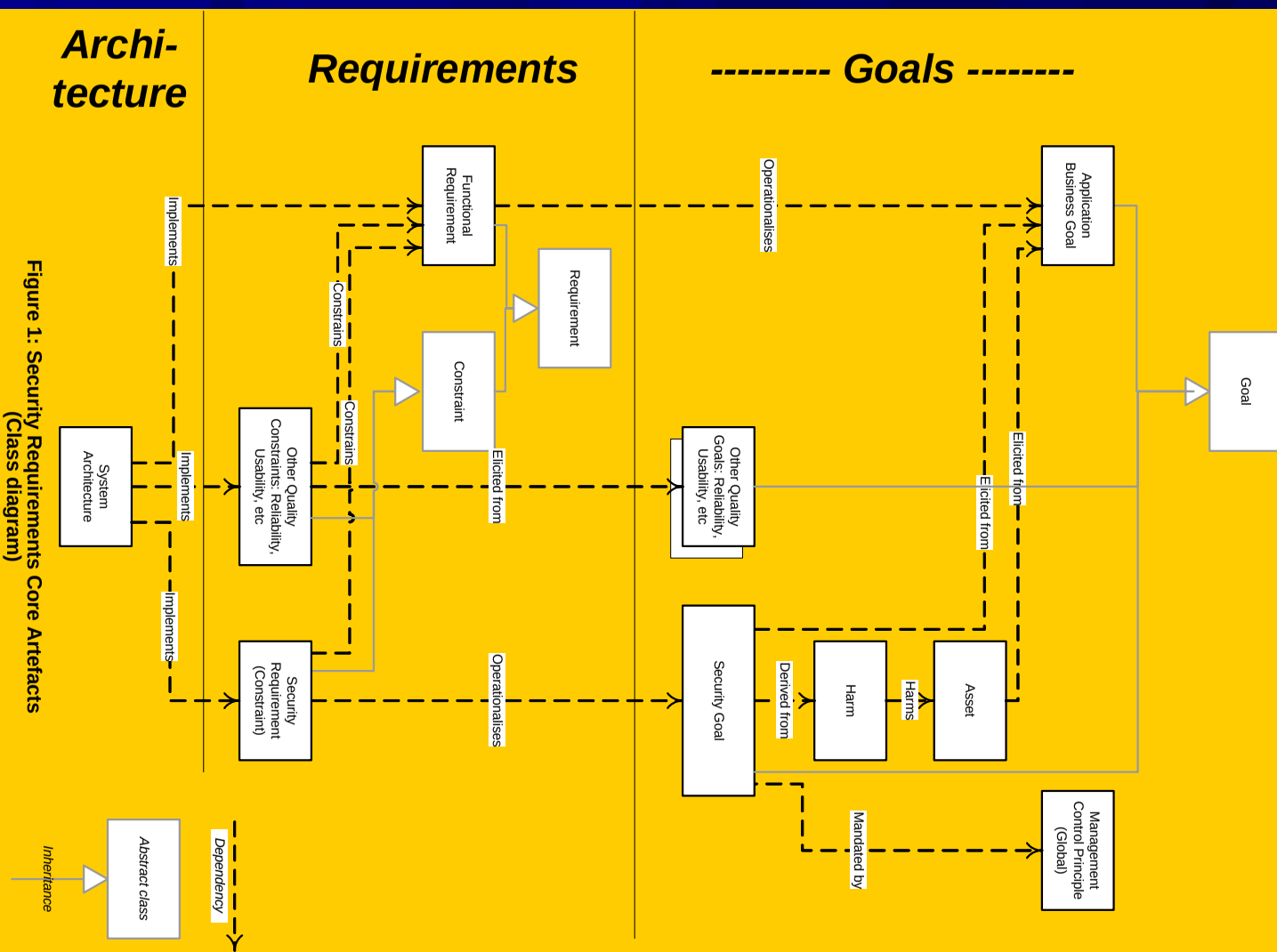
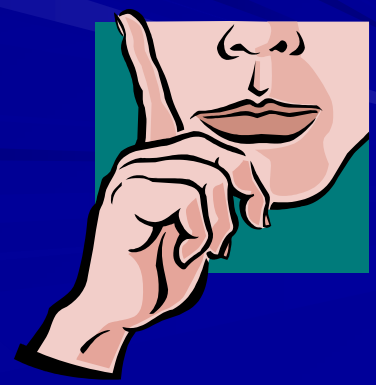


Figure 1: Security Requirements Core Artefacts (Class diagram)

J. D. Moffett, C. B. Haley, and B. Nuseibeh, "Core Security Requirements Artefacts," Department of Computing, The Open University, Milton Keynes UK, Technical Report 2004/23, June 2004.

The role of analysis in security requirements engineering

- the ability to show that proposed security goals adequately express what is needed by the stakeholders,
- the proposed security requirements adequately satisfy the goals, and
- the system adequately satisfies the security requirements.



Challenges of Security Requirements Engineering

1. **Scoping** – bounding the scope of security problems.
2. **Representation** – representing the
   - security problem context, and
   - negative requirements of a malicious user.
3. **Analysis** – reasoning about the satisfaction of security requirements.
4. **Integration** – relating security requirements and design.

Problems of scope …

- This cash machine has been designed with the most sophisticated password encryption.
- Special precautions have been taken to ensure that only authorised users with valid smart cards can withdraw money.

Problems of scope …

- Is it secure?

A Problem

- Not if the whole machine is stolen!

Not an isolated incident

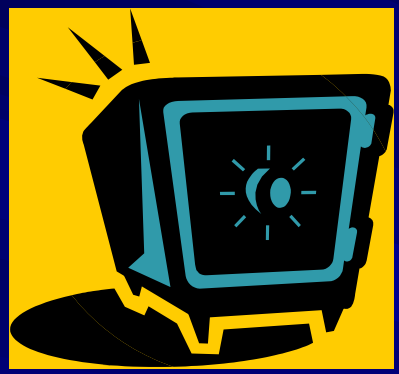
In a hotel room in Shanghai (May 2006)

This is a demo only!

A question of scope

- Bounding the scope of security problems is crucial
  - … and is the bread and butter of requirements engineering

Still on scope



- Do I need to put my money in a safe in the bank?

Still on scope

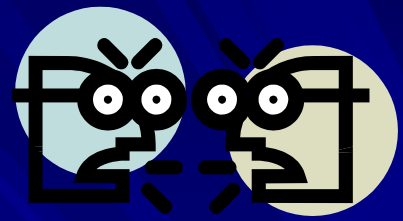
- Not if the bank building is adequately protected.

Trust Assumptions

- **Are the raw materials of the problem boundary**

- C.B. Haley, R. Laney, J.D. Moffett, and B. Nuseibeh, The Effect of Trust Assumptions on the Elaboration of Security Requirements , Proceedings of 12th IEEE International Requirements Engineering Conference (RE'04), Kyoto, Japan, 6-10 September 2004.

Arguing Security



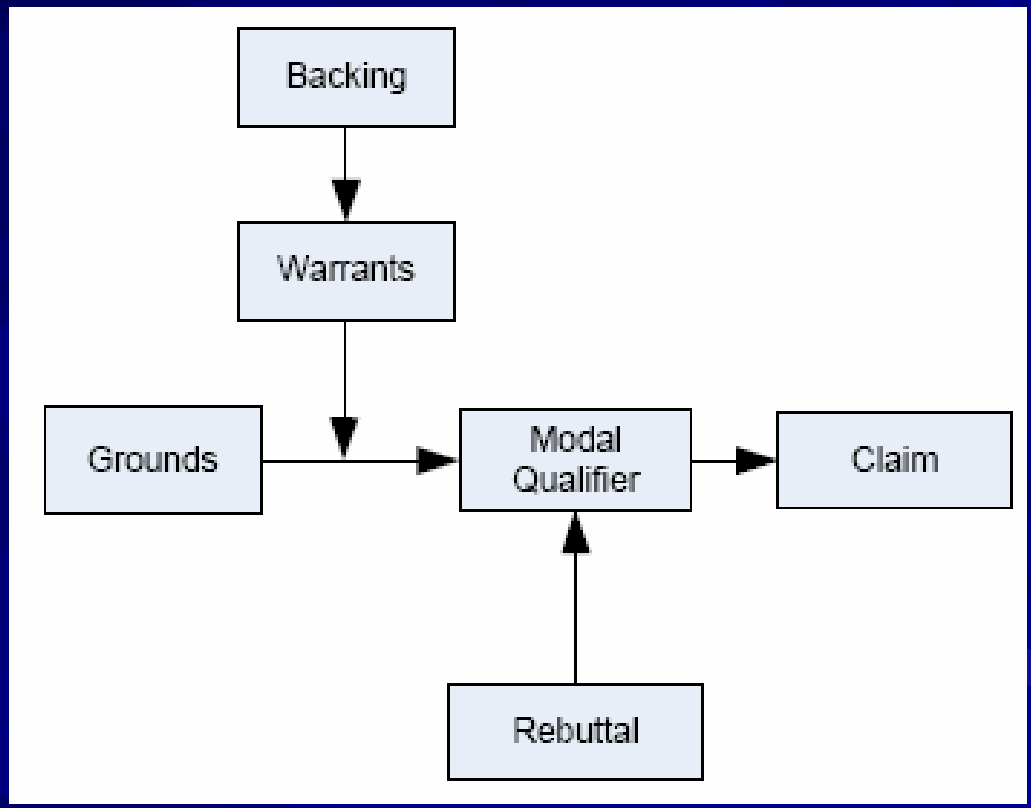
- … and knowing when to stop
- **There is a need to convince oneself and others of system security**
  - Through the construction of satisfaction arguments that a system meets its security requirements.
  - Proof versus argument
    - Absolute “shall not” is (usually) not provable
    - Context is (usually) much too large to analyse
    - Therefore “sufficiently convincing” argument must suffice

Combining arguments

1. **Formal argument**
   - Proof that system meets security requirements
   - Premises constructed from system context and behaviour
   - Assume closed word assumption
   - D, S ⊢ SecReq

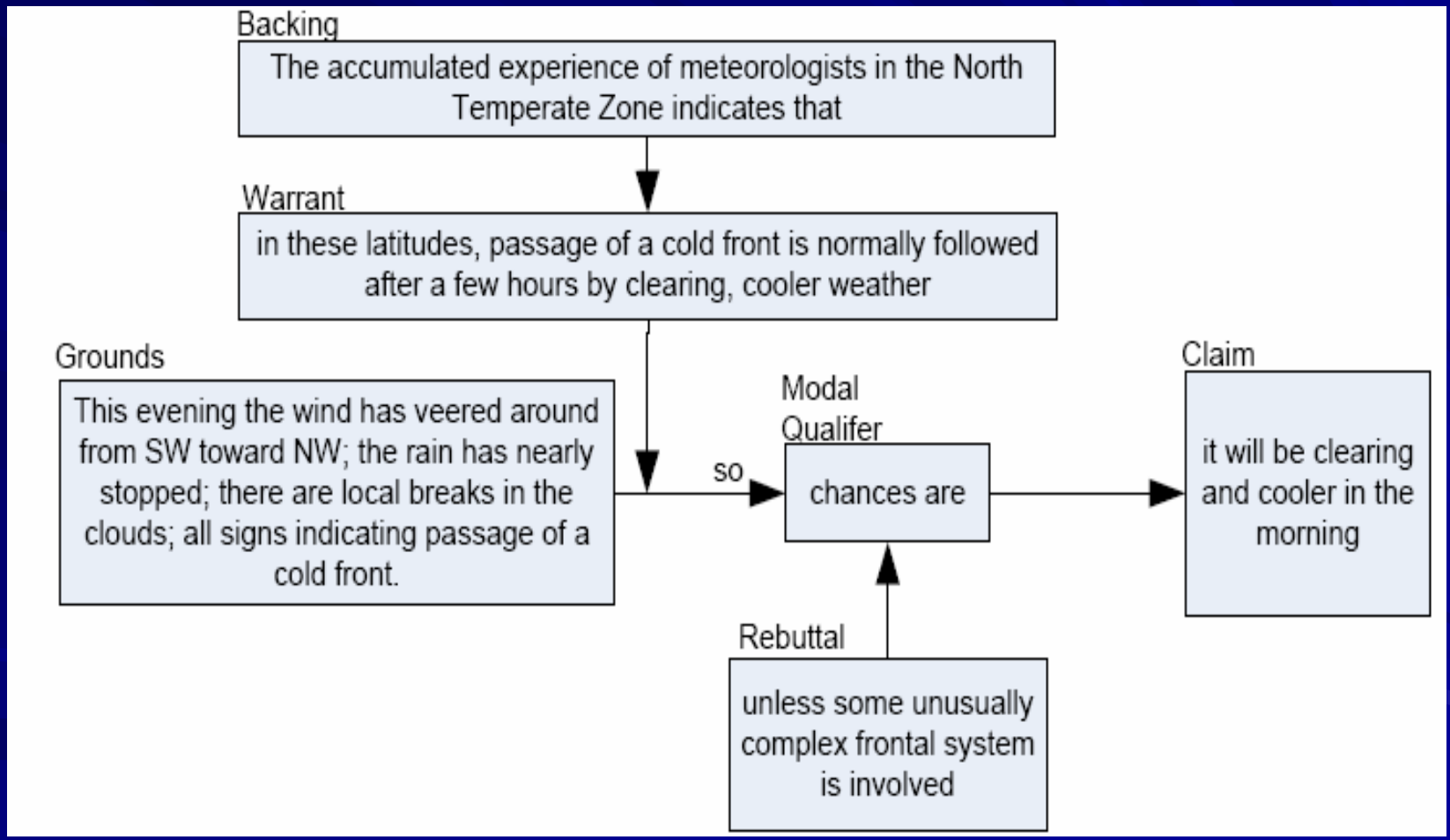
2. **Informal argument**
   - Structured argument that premises are valid
   - Brings trust assumptions to the surface
   - Challenge every premise

Toulmin – evidence based arguments



C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh, "Arguing Security: Validating Security Requirements Using Structured Argumentation," in *Proceedings of the Third Symposium on Requirements Engineering for Information Security (SREIS'05),* Paris, France, 29 August 2005.

Example argument

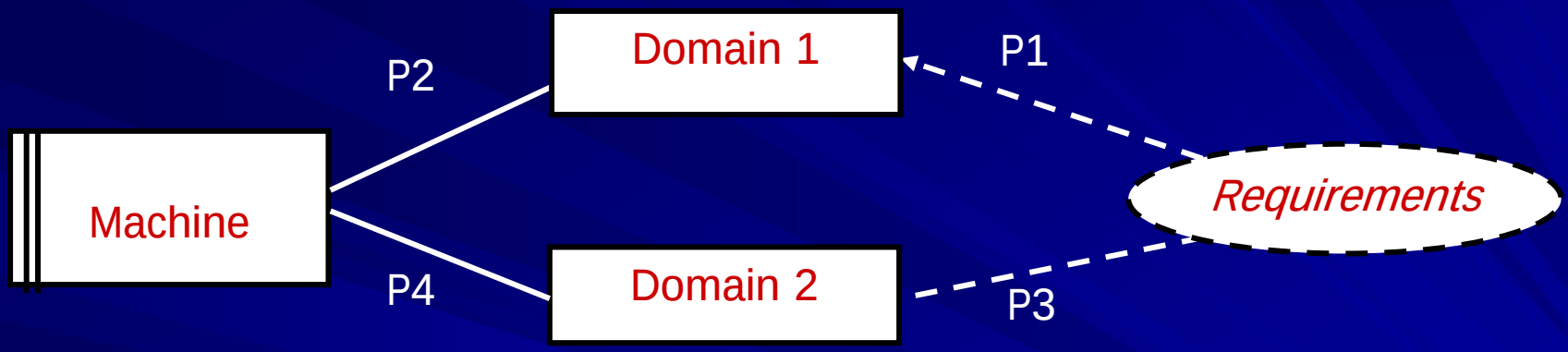


Anti-requirements

- We define an **anti-requirement** as the requirement of a malicious user that subverts an existing requirement.

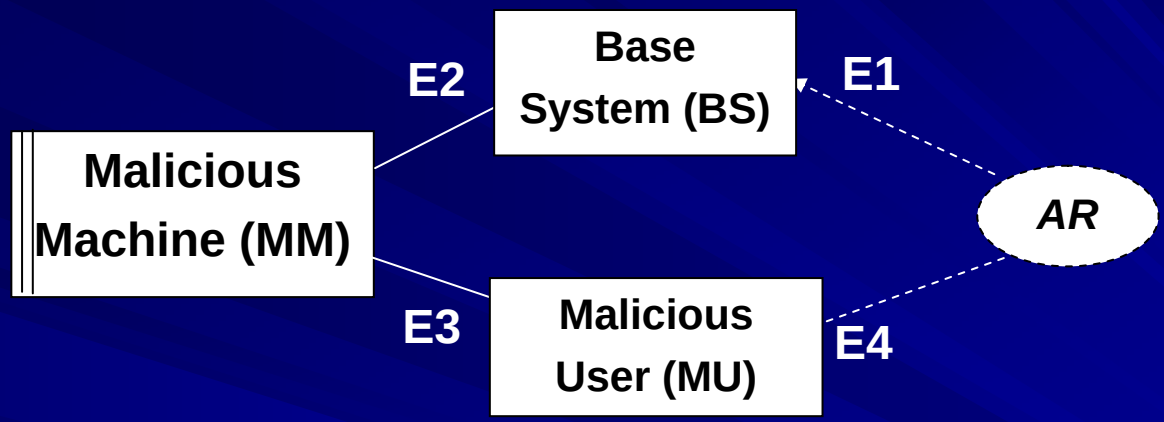
- This is useful because:
  - If we can find circumstances in which both a requirements and an anti-requirement hold (compose), then we hypothesise that the conditions of composition identify a potential vulnerability in a system that implements both requirements.

Problem Frames and Anti-requirements



- Consider an **anti-requirement (AR)** as the requirement of a malicious user that subverts an existing requirement.
  - It defines a set of undesirable phenomenon that will ultimately cause the system to reach a vulnerable state.

Abuse Frames



- The Base System (BS) is the system attacked.
- The anti-requirement (AR) specifies the undesirable phenomena in terms of *E1* in the Base System (BS).
- *E4* indicates that the Malicious User (MU) can interact with the *BS* through or unexpected phenomena.
- The specification of the *MM* describes the interface over the *E3* of the *MU* and the *E2* of the *BS* that will *existentially* satisfy the AR.

Threat analysis Using Abuse Frames

- **Scope the problem and identify the subproblems**
  - Describe the security concerns on the functionality to be achieved in each problem frame diagram.
- **Identify the threats and constructing abuse frames**
  - Identify the anti-requirements.
- **Identify security vulnerabilities**
  - Describe the domain properties.
  - Backward search.
- **Address security vulnerabilities**
- **Iterate!**

Abuse Frame Classes (Patterns)

- Interception
- Modification
- Behavioural

L. Lin, B. Nuseibeh, D.C. Ince, and M. Jackson, Using Abuse Frames to Bound the Scope of Security Problems, Poster paper, Proceedings of 12th IEEE International Requirements Engineering Conference (RE'03), Monterey, USA, September 2004, 354-355.

L. Lin, B. Nuseibeh, and D. Ince, Using Abuse Frames to Bound the Scope of Security Problems, Proceedings of the Third International Workshop on Requirements for High Assurance Systems (RHAS 2004), co-located with RE'04, 6th September 2004, Kyoto, Japan. Available an CMU/SEI Technical Report and downloadable from: http://www.sei.cmu.edu/community/rhas-workshop/lin.pdf

Lessons Learned (so far)

- Must understand the system context
  - What does your software interact with, and how?
  - Understand organisational context
- Know and test your assumptions
  - What do you know, and how do you know it?
  - Argue (reason) systematically

Research Agenda

- **Boundary issues:** problem scoping and decomposition
  - Boundaries of security attacks are often fuzzy
  - Patterns: from radical to normal engineering
- **Representation issues**
  - Lack of specification notations for “prevention” or “prohibition” (what should NOT happen)
- **Problem composition and analysis**
  - Composing security properties
- **Integrating Security RE within SE process**
  - Relating security requirements to security architectures and mechanisms

Selected Related Work

- van Lamsweerde et al: antigoals in KAOS
- Antòn et al: privacy requirements and policies
- Chung, Liu, Mylopoulos, Yu: *i** security softgoals
- Giorgini, Massacci, Silva, Castro et al: Tropos
- Kelly et al: extension of GSN to security
- Sindre & Opdahl; and Alexander: misuse cases
- McDermott & Fox: abuse cases
- Taguchi et al: using RBAC, KAOS, and Common Criteria

Thank you.

Acknowledgements:
Karim Adam
Francis Chantree
Bob Crook
Charles Haley
Jon Hall
Robin Laney
Luncheng Lin
Michael Jackson
Jonathan Moffett
Armstrong Nhlabatsi
Blaine Price
Lucia Rapanotti
Mohammed Salifu

Financial Support:
The Royal Academy of Engineering
The Leverhulme Trust
EPSRC