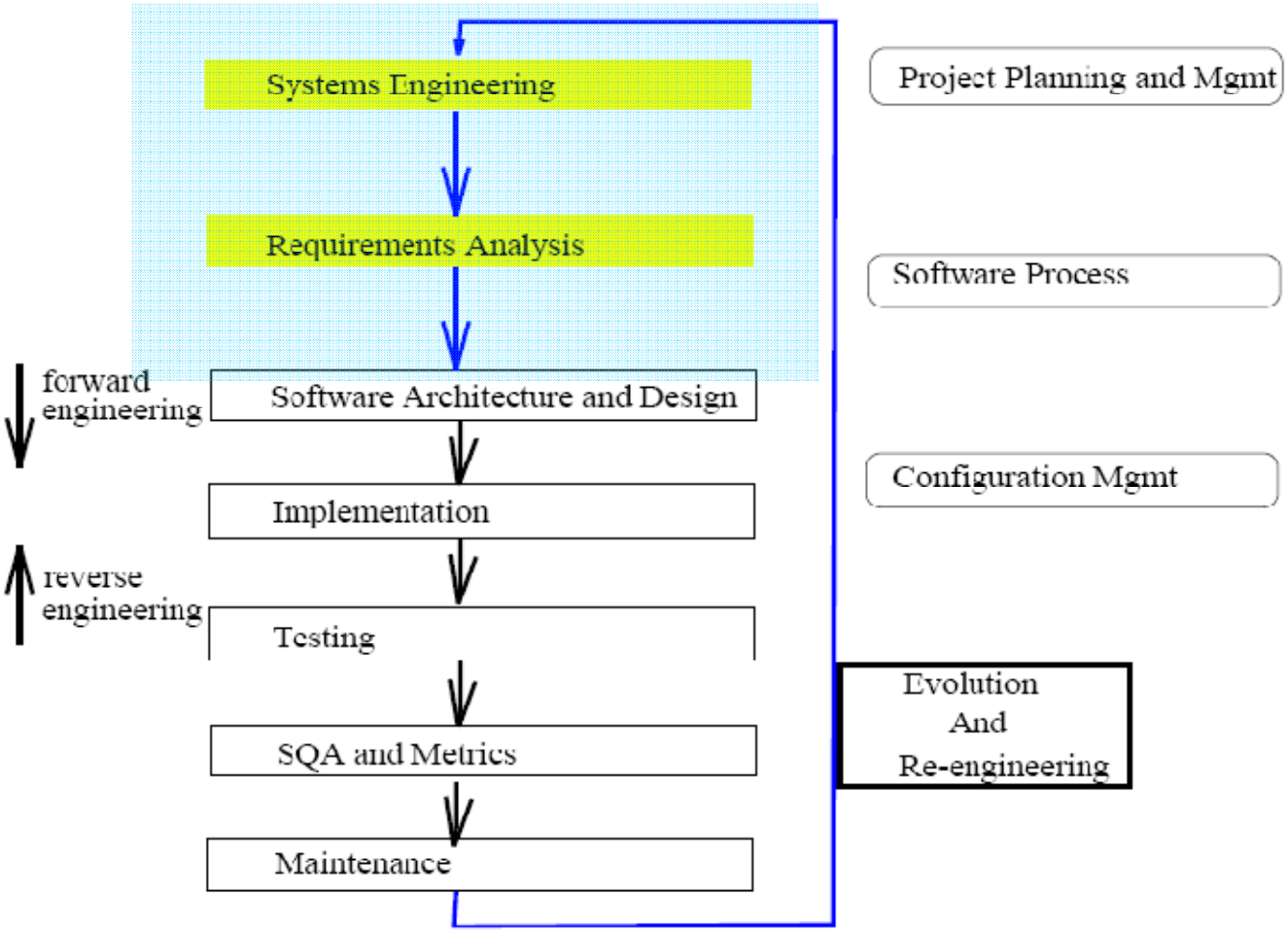
Requirements Engineering: Introduction

- Why RE?
  - Software Lifecycle and Error Propagation
  - Case Studies and The Standish Report
- What is RE?
  - Role of Requirements
  - Ross's Definition of Requirements Definition
  - The Reference Model (WRSPM) vs. The 4-Variable Model
  - Zave's Definition of RE
- How to do RE?
  - Types of Errors
  - Desirable Properties

Why RE?

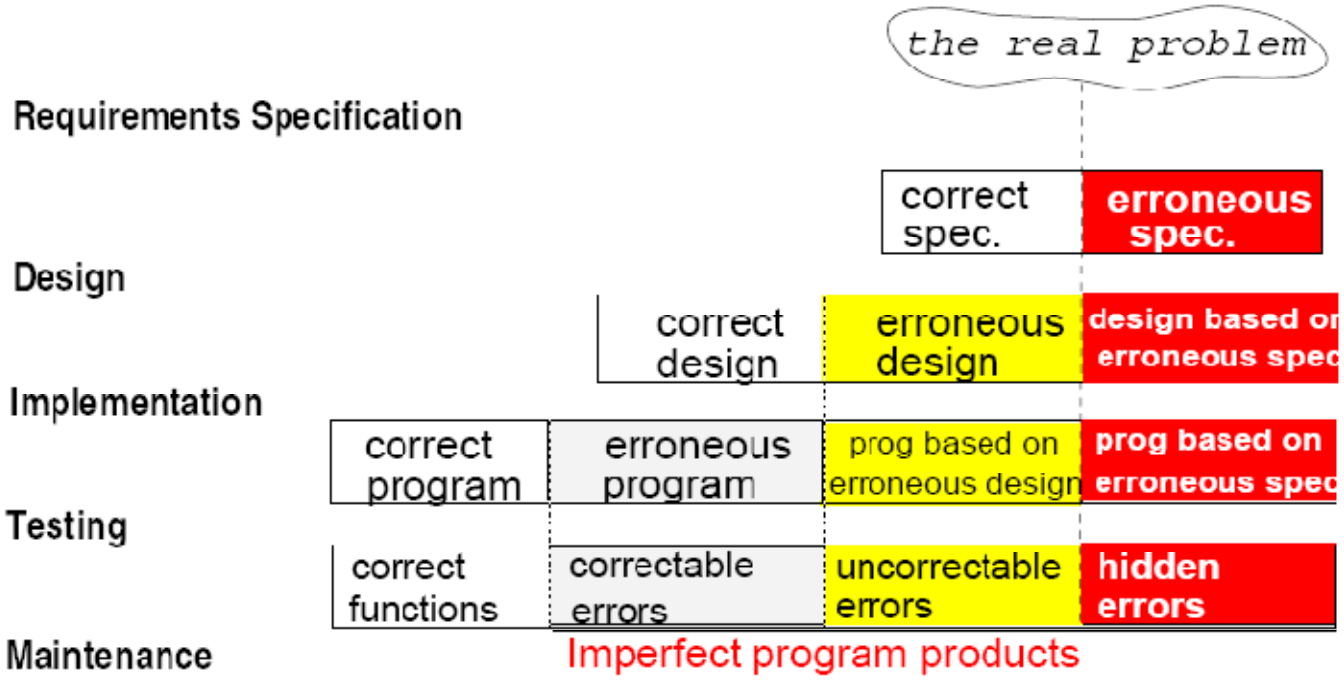
Software Lifecycle Revisited



Why RE?

Error Propagation in Lifecycle [Mizuno82]

Simplified Lifecycle | **Cumulative Effects of Error**



How big is the erroneous spec.?
How costly is it?

Why RE?

How big is the "erroneous specification"?

- **Bell Labs and IBM studies**

  80% of all defects are inserted in the requirements phase.
  Improving the requirements definition process reduces
  the amount of testing and rework required.

  And the above figures do not include the end user losses
  who have to live with poor software on a daily basis. *[Testing Techniques Newsletter]*

- **U.S. Air Force projects**

  36% of all defects were due to faulty requirements translation.
  Only 9% of these errors were resolved (in the requirements phase) *[Sheldon92]*

- **Voyager and Galileo spacecraft**

  Of the 197 significant software faults found during integration & system testing,
  only 3 of those errors were programming errors;
  the vast majority of the faults were requirements problems. *[Lutz93]*

- **Application Specific Integrated Circuits [ASICs)**

  >1/2 are faulty on first fabrication. A majority of these faults are related to reqs. errors.

- **[UK Health and Safety] Executive**

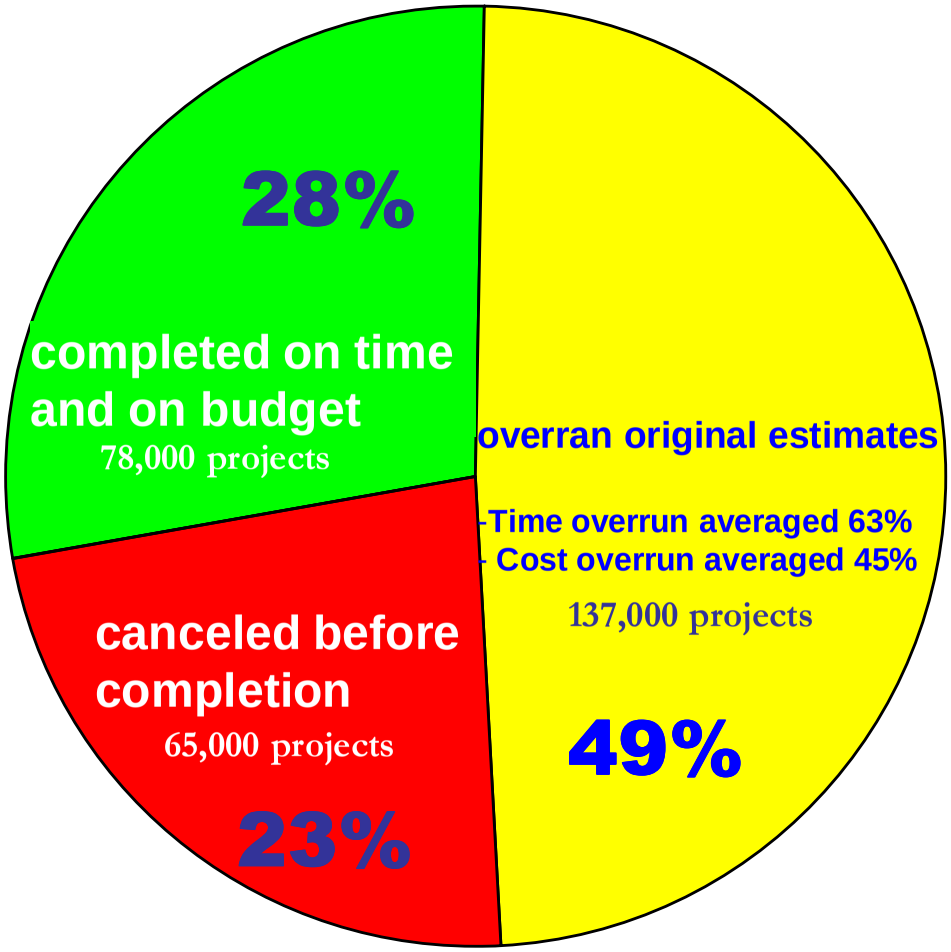
  Specification 44.1%
  Design and Implementation 14.7%
  Installation and Commissioning 5.9%
  Operation and Maintenance 14.7%
  Changes after commissioning 20.6%

  [Her Majesty's Stationary Office 1995 ISBN 0 7176 0847 6]

Issues

What Factors Contribute to Project Success?

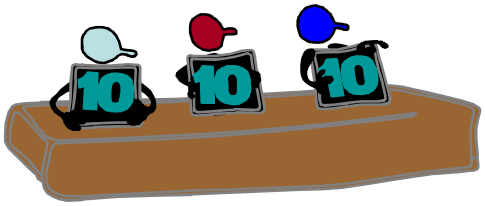
The Standish Group Report, '01 – The "Chaos" Report (www.standishgroup.com), yearly since 1994



The CHAOS Ten
Project Success Factors
1. Executive Management Support
2. **User Involvement**
3. Experienced Project Manager
4. **Clear Business Objectives**
5. **Minimized Scope**
6. Standard Software Infrastructure
7. **Firm Basic Requirements**
8. Formal Methodology
9. Reliable Estimates
10. Other

Issues

What Factors Contribute to Project Failure?



The CHAOS Ten

Project Challenged Factors

1. Lack of User Input
2. Incomplete Requirements & Specifications
3. Changing Requirements & Specifications
4. Lack of Executive Support
5. Technology Incompetence
6. Lack of Resources
7. Unrealistic Expectations
8. Unclear Objectives
9. Unrealistic Time Frames
10. New Technology

Standish Group, '01 (www.standishgroup.com)

The CHAOS Ten

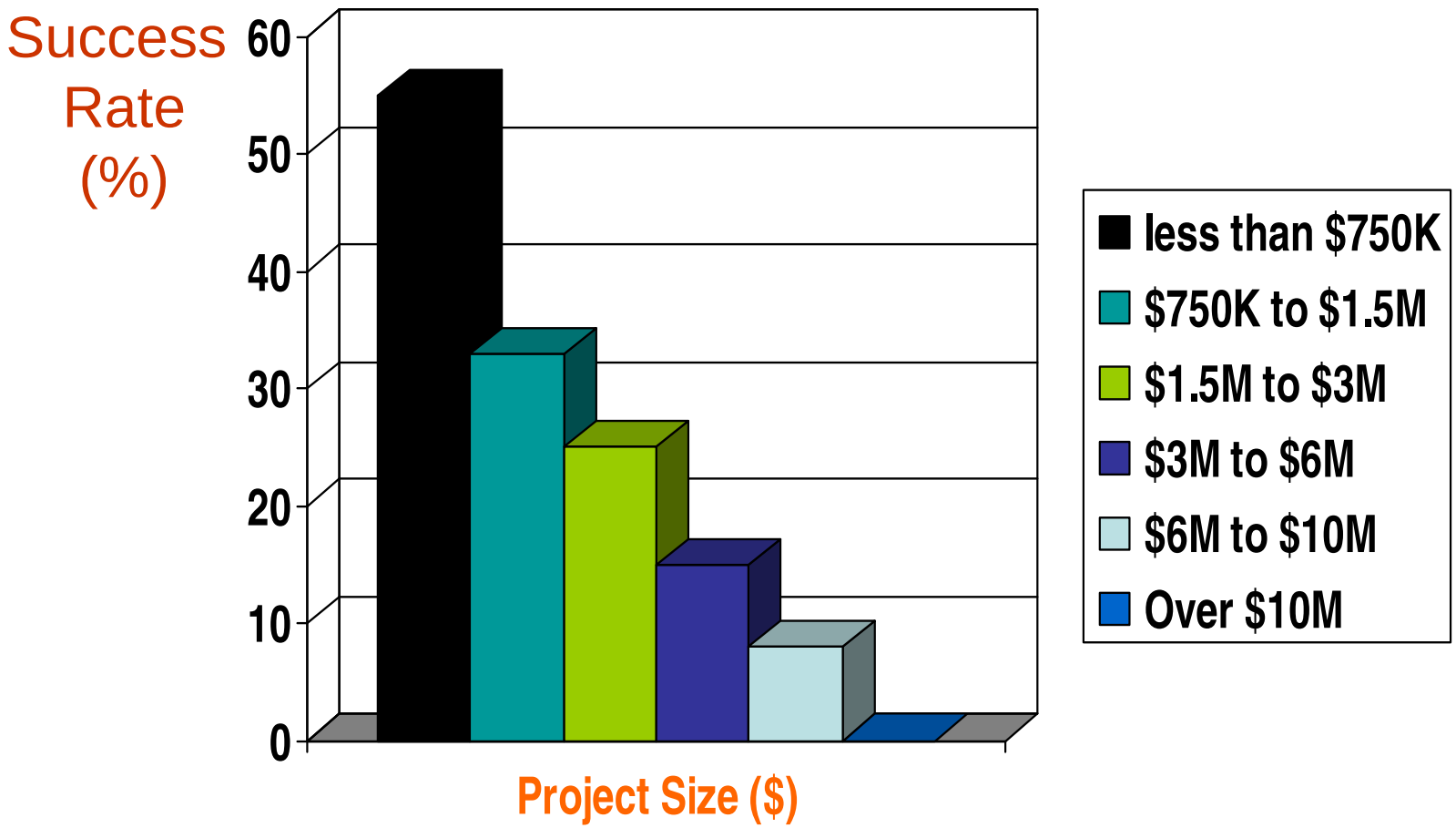
Project Impaired Factors

1. Incomplete Requirements
2. Lack of User Involvement
3. Lack of Resources
4. Unrealistic Expectations
5. Lack of Executive Support
6. Changing Requirements & Spec
7. Lack of Planning
8. Didn't Need It Any Longer
9. Lack of IT Management
10. Technology Illiteracy

"The definition of insanity is doing the same thing over and over again and expecting a different result." [Albert Einstein]

Size Is Important: Success by Project Size

Standish Group, '99 (www.standishgroup.com)



Why?

Why RE?

How costly are requirements errors?

- *[Lindstrom93]*

  Get the requirements wrong, you'll destroy the project.

- *[Boehm87]*

  COST (correcting design/implementation errors)
  = 100 X COST (correcting requirements errors)

- *[Humphrey, Managing the Software Process, Ch1, p11-12]*

  a useful rule of thumb: It takes about 1 to 4 working hours to find and fix a bug through inspections and about 15 to 20 working hours to find and fix a bug in function or system test.

- *[Curtis88]*

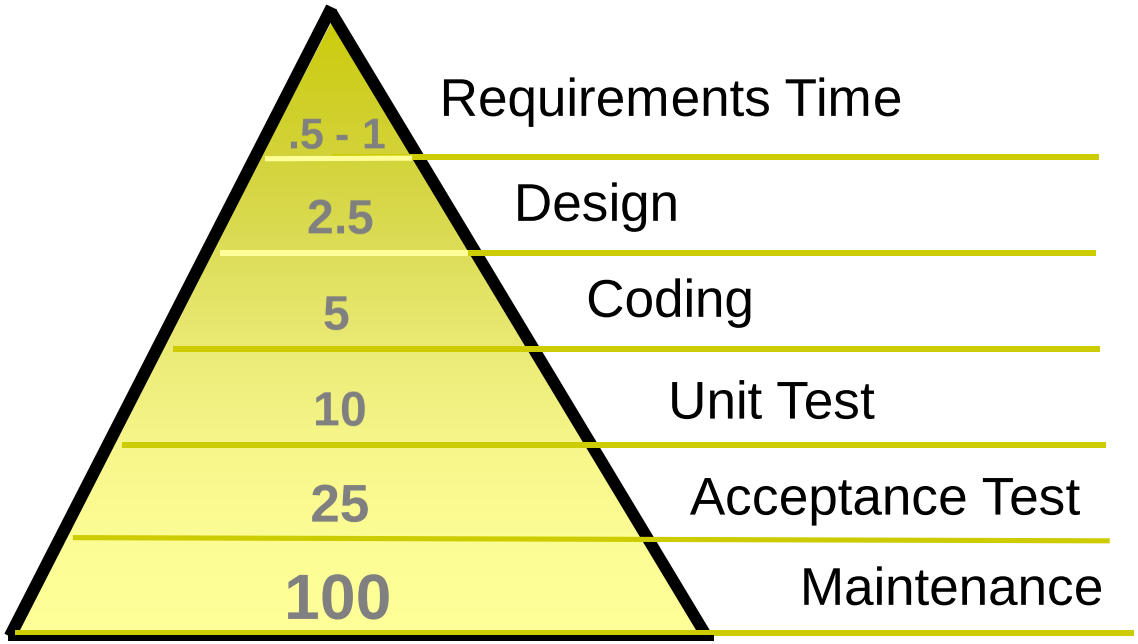
  Three most frequent problems plaguing large software systems:

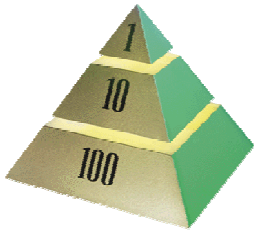
  communication and coordination
  thin spread of domain application knowledge
  changing and conflicting requirements

  Defining the problem is The Problem

The High Cost of Requirement Errors

The 1-10-100 Rule





"All together, the results show as much as a 200:1 cost ratio between finding errors in the requirements and maintenance stages of the software lifecycle."

Relative cost to repair errors:
When introduced vs. when repaired.
[Davis 1993]

Average cost ratio 14:1
[Grady1989] [Boehm 1988]

Why RE?

Productivity *Follows* Quality

[Capers Jones]

- Productivity was the wrong issue.
- Most poor productivity is due to long testing, fixing.
- If get the quality up in the original build, testing goes fast and overall productivity goes up.
- So aim at quality, and you get productivity.
  Aim at productivity rather than quality, get neither.

[Brooks, Jr.]

Why RE?

- "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, including all the interfaces to people, to machines, and other software systems.

  No part of the work so cripples the resulting system if done wrong.
  No other part is more difficult to rectify later."

[Brooks, 1987]

What is RE?

Role of requirements

- **agreement regarding the requirements between system developers, customers, and end-users.**

  => *legal contract (flexible, inflexible)*
  => *multi-party*
  => *communication and coordination*
  –> *conflicting views*
  => *changing views*

We want to make a change in the environment
We will build some system to do it
This system must interact with the environment

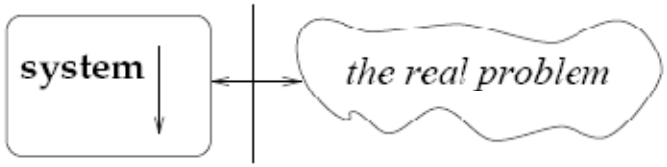
should be written in the user's language!

- **the basis for software design**

  –> *defect-free as much as possible*
  => *technically feasible*

- **support for verification and validation**

complete & sound I/O
complexity in the # of I/O items, and relationships between them and constraints on them



- **support for system evolution**

  => *system evolution = change (old system, new system)*
  change (old requirements, new requiremens)

What is RE?

What are requirements?

"... Requirements definition is a careful assessment of the needs that a system is to fulfill.

It must say why a system is needed,

based on current and and foreseen conditions,
which may be internal operations or an external market

It must say what system features will serve and satisfy this context.

And it must say how the system is to be constructed ..." *[Ross77]*

Not about the design

	Fn	NFn
E/D/B/W		
R		
S		

traditional emphasis here or in S + D

What is RE?

What are requirements?

why?

- **Enterprise requirements**

  for "context analysis" - the reasons why the system is to be created.
  (e..g, why IS for BPR, organizational structure, agents, goals)

  constraints on the environment in which the system is to function
  (e..g, airplane running beyond runway, AT&T Internet service)

  the meaning of system requirements
  (symbols, relationships, ontology, vocabulary)

what?

- **(System) functional requirements**

  a description of what the system is to do;
  what information needs to be maintained?
  what needs to be processes?
  {f: I -> O}

  traditional emphasis here or in S + D

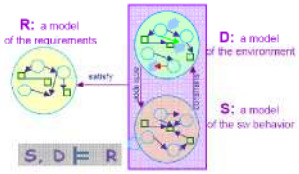
how?

- **(System) non-functional requirements**

  (global) constraints on how the system is to be constructed and function.
  E.g., -ilities and -ities
  {bcfh(f: I -> O)}

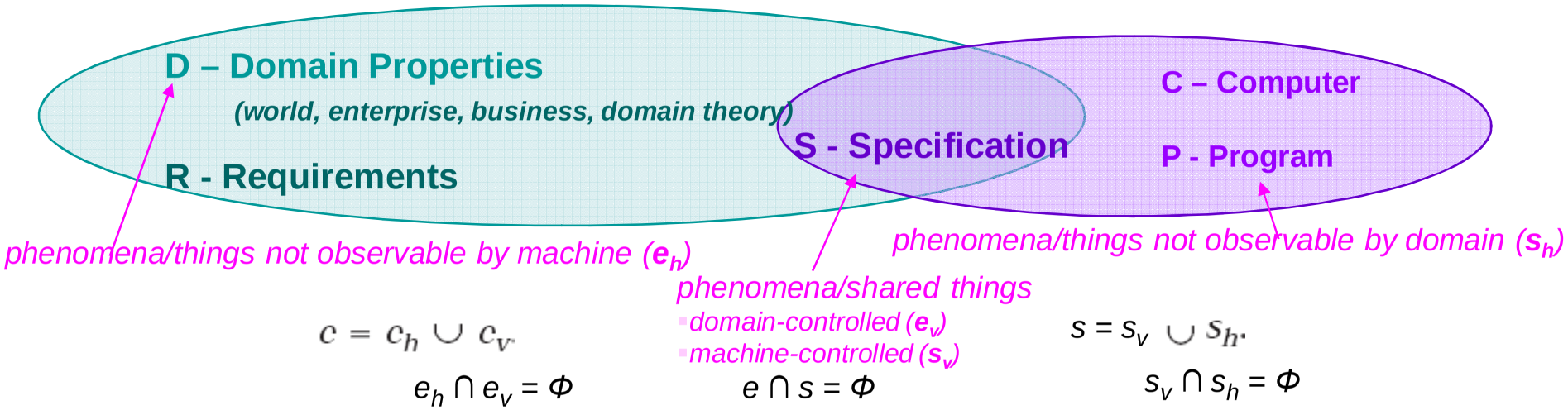
	Fn	NFn
E/D/B/W		
R		
S		

What are requirements?



[P. Zave and M. Jackson, Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology* 6(1) 1-30. ACM Press. 1997]
[C. Gunter, E. Gunter, M. Jackson and P. Zave, "A Reference Model for Requirements and Specifications," IEEE Software, May/June 2000. pp. 37-43]

The WRSPM Model



$c = c_h \cup c_v$ $\quad$ $e_h \cap e_v = \Phi$ $\quad$ $e \cap s = \Phi$ $\quad$ $s = s_v \cup s_h$ $\quad$ $s_v \cap s_h = \Phi$

- **Domain Properties:** **(indicative, = assumptions=domain knowledge)**
  things in the environment (application domain) that are true regardless of the proposed system
- **Requirements:** **(optative)**
  things in the application domain that we wish to be made true through the proposed system
  "Many phenomena not accessible by the machine"
- **Specification:**
  a description of the behaviors that the program must have in order to meet the requirements
  "Can, and should, only be written in terms of shared phenomena"

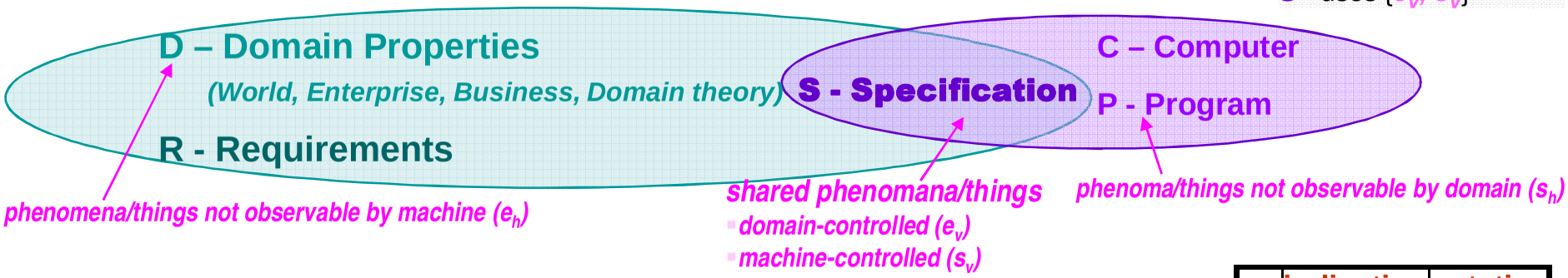
❖ ***Designated Terminology*** – **names/vocabulary to describe W, (R), S, M in terms of phenoma – typically states or events**

Requirements should contain *nothing but* information about the environment.

[P. Zave and M. Jackson, Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology* 6(1) 1-30. ACM Press. 1997]

The WRSPM Model

W, **R** - uses $\{e_h, e_v, s_v\}$
P, **M** - uses $\{e_v, s_v, s_h\}$
S - uses $\{e_v, s_v\}$



	indicative	optative
e_h	✓	✓
e_v	✓	✓
s_v	✓	✓

- **R**equirements describe what is observable at the interface between the environment and the machine – hence exist only in the **e**nvironment;
- Anything else is regarded as **i**mplementation bias;
- States in **s**pecifications should describe states of the **e**nvironment – *hence, specification languages intended to describe internal (program) states of the machine are inadequate.*

Verification: S, D ⊨ R

P, C |= S

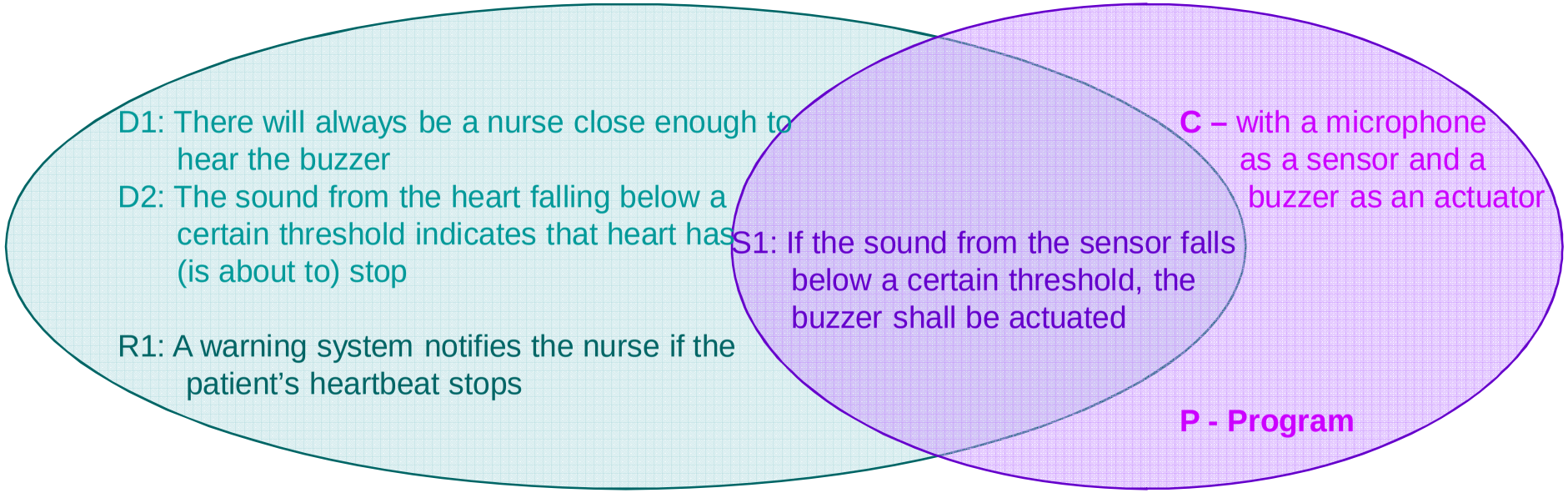
Consequences

- Freedom to collect and record information about the **e**nvironment even before we are sure it will be needed
  (i.e., no minimality restriction - there must be nothing that is not necessary to carry out the currently proposed machine functions.);
- Designations refer to the real **w**orld, and **m**achine states may have NO direct correspondence to it

What are requirements?

[C. Gunter, E. Gunter, M. Jackson and P. Zave, "A Reference Model for Requirements and Specifications," IEEE Software, May/June 2000. pp. 37-43]

Example 1: Patient Monitoring



Designation Categories:

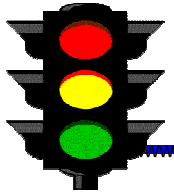
e_h: the nurse and the heartbeat of the patient.

e_v: sounds from the patient's chest.

s_v: the buzzer at the nurse's station.

s_h: internal representation of data from the sensor.

What if the domain assumptions are wrong?



What are requirements?

Example 2: Traffic lights

D1: Drivers stop at red lights
D2: Pedestrians walk when green

R1: Allow pedestrians to cross the road safely

S1: Show a red light to the cars and a green light to the pedestrians

C – Computer

P - Program

Example 3: Traffic Lights - Safety

D1. Drivers stop at red lights
D2. Pedestrians stop at red lights
D3. Drivers drive at green lights
D4. Pedestrians walk when green

R1: Pedestrians and cars cannot be in the intersection at the same time

S1: Never show a green light to both pedestrians and cars

C – Computer

P - Program

What if the domain assumptions are wrong?

What are requirements?

Example 4: Aircraft Control

D1: Wheel pulses on if and only if wheels turning
D2: Wheels turning if and only if moving on runway

R1: Reverse thrust shall only be enabled
when the aircraft is moving on the runway

S1: Reverse thrust enabled
if and only if wheel pulses on

C – Computer

P - Program

Example 5: Security

D1: Authorized personnel have passwords
D2: Passwords are never shared with
non-authorized personnel

R1: The database shall only be accessible by
authorized personnel

S1: Access to the database shall only
be granted after the user types an
authorized password

C – Computer

P - Program

What if the domain assumptions are wrong?

What are requirements?

Example 6: Coffee Machine

D1: Before the switch is moved to the On position, the user must add ground coffee to the filter and insert it in the coffee machine.

D2: Before the switch is moved to the On position, the user must add water to the reservoir.

R1: When the user moves the three-way switch to the On position, coffee shall be brewed.

S1: If the three-way switch is On, the coffee brewer shall be actuated

C – with a switch as a sensor and a brewer as an actuator

P - Program

Designation Categories:

e_h: ?

e_v: ?

s_v: ?

s_h: ?

What Are Requirements?

Example 7: In a single-customer banking environment,

- Notation: equational logic
- *action(a):* atomic and sequential;
- *earlier(a1, a2):* a nondense total order on actions;
- *pause(p):* a unique pause between each adjacent pair of actions
- *begins(a,p):* action *a* precedes pause *p* immediately in the temporal sequence
- *ends(a,p):* action *a* succeeds pause *p* immediately in the temporal sequence
- *(*i | R(i) : P(i)):* the accumulation of values *P*(*i*), using operator *, over all values *i* for which predicate *R*(*i*) holds.

D

- *deposit(a,m):* *a* is an action in which amount *m* is deposited
- *withdrawal-request(a,m):* *a* is an action in which a withdrawal of amount *m* is requested
- *withdrawal-payout(a,m):* *a* is an action in which amount *m* is paid out as a withdrawal
- *balance(b,p):* during pause *p* the balance is amount *b;*

At any time, the balance is equal to the sum of the amounts of all the previous deposits, minus the sum of the amounts of all the previous withdrawal payouts:

$$(\forall\, b,p \mid : balance(b,p) \equiv$$
$$(b = (+m \mid (\exists a \mid : deposit(a,m) \wedge earlier(a,p)) : m) - (+m \mid (\exists a \mid : withdrawal\text{-}payout(a,m) \wedge earlier(a,p)) : m))\ )$$

R

A withdrawal request leads to a withdrawal payout, if the requested amount is less then the current balance

$$(\forall\, a,m,p,b \mid withdrawal\text{-}request(a,m) \wedge ends(a,p) \wedge balance(b,p) \wedge b \geq m:$$
$$(\exists a' \mid : withdrawal\text{-}payout(a',m) \wedge earlier(a,a'))\ )$$

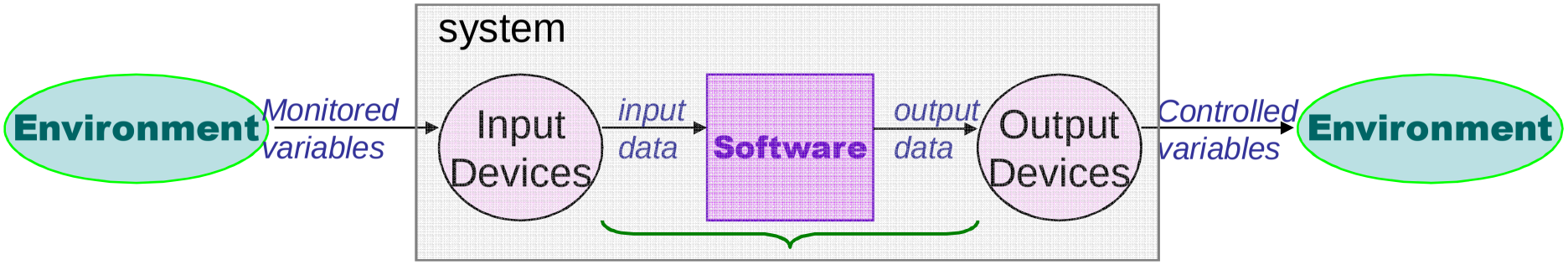
S – a requirement (**R**), which is implementable, hence a specification

$$(\forall\, a,m,p \mid withdrawal\text{-}request(a,m) \wedge ends(a,p) \wedge m \leq$$
$$((+m \mid (\exists a \mid : deposit(a,m) \wedge earlier(a,p)) : m) - (+m \mid (\exists a \mid : withdrawal\text{-}payout(a,m) \wedge earlier(a,p)) : m)) :$$
$$(\exists a' \mid : withdrawal\text{-}payout(a',m) \wedge earlier(a,a'))\ )$$

The 4-Variable Model

(the functional documentation model)

D.L. Parnas and J. Madey, "Functional Documentation for Computer Systems," Science of Computer Programming, Vol. 25, No. 1, Oct. 1995, pp. 41–61.



S - Specification of software in terms of inputs & outputs

(possibly large in number, and in very complex relationships)

NAT(m, c): describes nature without making any assumptions about the system;

REQ(m, c): describes the desired system behavior;

IN(m, i): relates the monitored real-world values to their corresponding internal representation;

OUT(o, c): relates the software-generated outputs to external system-controlled values; and

SOF(i, o): relates program inputs to program outputs.

Any issues?

Nat - the range of sounds detected or non-detected by the sensor and the possible range of values of the actuator controlling the buzzer.

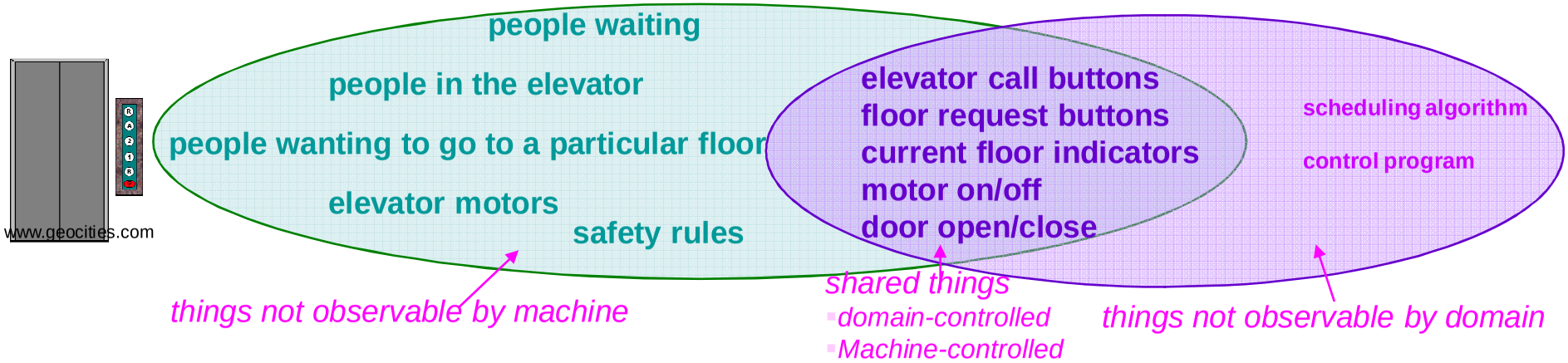
Req - A warning that notifies the nurse if the system detects heart stops beating. The document' formalization: if the sound being monitored falls below a certain threshold, then the system sound the buzzer.

In - The input registers holding the data read from the sensor monitoring the sounds of the heart beat.
Out - The output registers which are read by the actuator that can sound the buzzer.

SOF - If the input register doesn't show signs of a heart beat for more than some specified time then the output register indicates the alarm to ring.

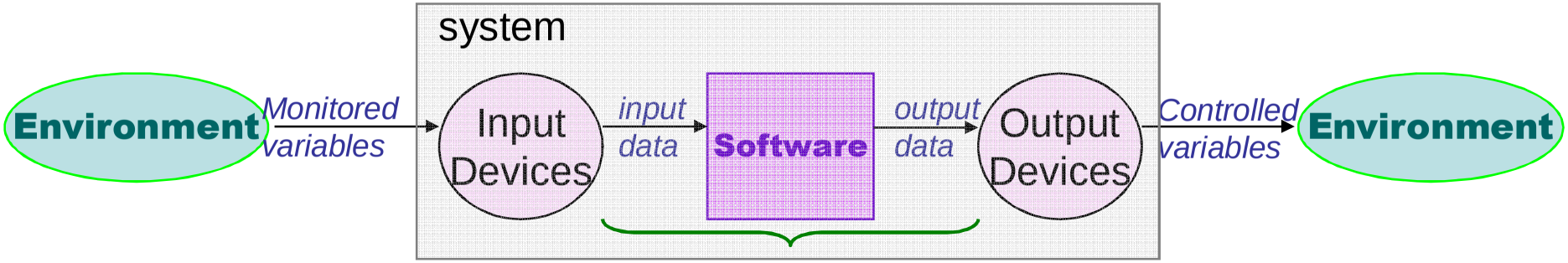
Boundaries are not fixed

Example 1: Elevator control system



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

Example 2: The 4-variable model



S - Specification of software in terms of inputs & outputs
(possibly large in number, and in very complex relationships)

Systems engineer decides - what application domain phenomena are shared
- the boundaries by designing the input/output devices
- I/O data as proxies for the monitored and controlled variables

Another Look at Completeness $S, K \vdash R$

[p. Zave and M. Jackson, Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology* 6(1) 1-30. ACM Press. 1997.

If the five following criteria are satisfied, then requirements engineering, in the strongest sense, is complete.

(1) There is a set R of requirements. Each member of R has been validated (checked informally) as acceptable to the customer, and R as a whole has been validated as expressing all the customer's desires with respect to the software-development project.

(2) There is a set K of statements of domain knowledge. Each member of K has been validated (checked informally) as true of the environment.

(3) There is a set S of specifications. The members of S do not constrain the environment, they are not stated in terms of any unshared actions or state components, and they do not refer to the future.

(4) A proof shows that:

$$S, K \vdash R$$

This proof ensures that an implementation of S will satisfy the requirements.

$$P, C \vdash S$$

If the machine is as described in C, then execution of the program P will ensure the behaviour S at the machine's interface with the world.

(5) There is a proof that S and K are consistent. This ensures that the specification is internally consistent and consistent with the environment. Note that the two proofs together imply that S, K, and R are consistent with each other.

… requirements should contain *nothing but* information about the environment. [p9]

The requirements are complete if they are sufficient to establish the goal they are refining

[K. Yue, "What Does It Mean to Say that a Specification is Complete?", Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design, Monterey, 1987.]

What is RE?

What are requirements?

"... Requirements definition is a careful assessment of the needs that a system is to fulfill.

It must say why a system is needed,

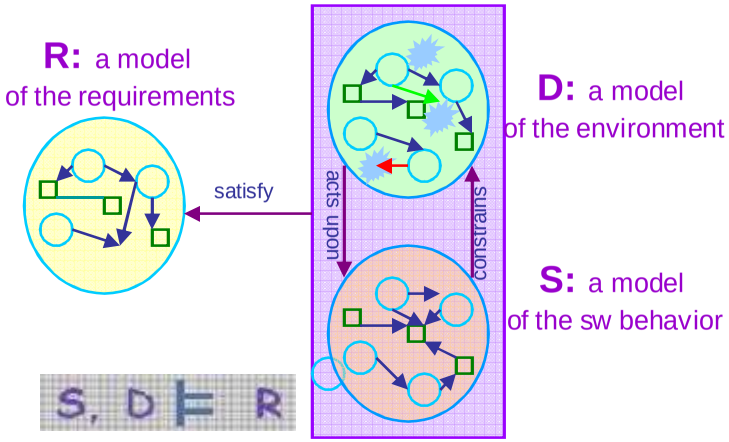
based on current and and foreseen conditions,
which may be internal operations or an external market

It must say what system features will serve and satisfy this context.

And it must say how the system is to be constructed ..."

[Ross77]

	Fn	NFn
E/D/B/W		
R		
S		



What is RE?

"... Requirements Engineering is the branch of Systems engineering concerned with

real-world goals for,
services provided by, and
constraints on

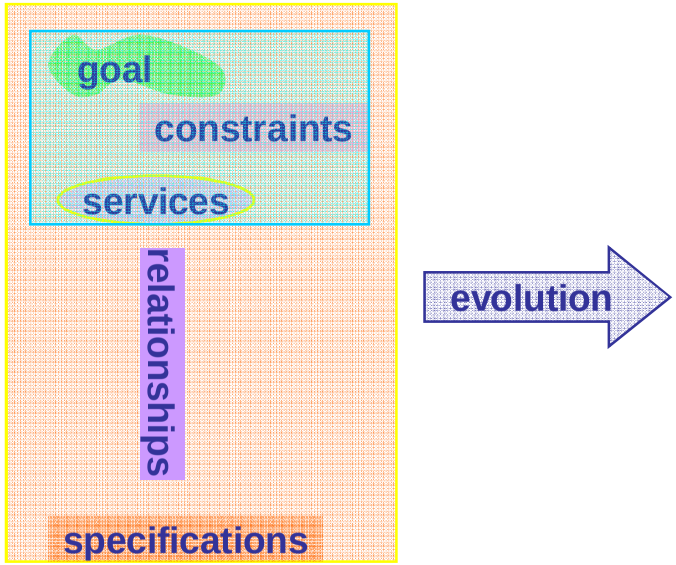
software systems

Requirements engineering is also concerned with

the relationships of these factors

to precise specifications of system behavior
and
to their evolution over time and across system families..."

[Zave94]



What is RE?

Not all RE projects are similar

- **Customer-driven projects**
  - involve a customer who needs a system that solves a particular problem
  - often one-shot

Customer	Project	Developer
Navy	Missile Tracking	Raytheon-E-Sys
Navy	Weapon Inventory	TRW
Sprint	MAN	Nortel
MCI	Accounting	Nortel

- **Market-driven projects**
  - involve a developer who needs to develop a system that is to be sold to the market
  - often hard to determine what the customer really wants

Customer	Project	Developer
Office workers?	Groupware	Microsoft
Casual users?	Multimedia	Fujitsu

* Examples are hypot

What is RE?

Not all RE projects are similar

- A Field Study involving 10 organizations *[Lubars93]*
  - **Customer-driven projects**
    - usually given large monolithic statements of requirements
    - despite their size, these are often sketchy, ill-defined
    - concept of "superdesigner" for interpretation, filling gaps

      => needs REeer who can deal with sketchy, ill-defined reqs

      => clarification, completeness through communication
  - **Market-driven projects**
    - often smaller requirements produced in-house
    - increasingly important (e.g., from military to commercial, from internal/external customer to open market)

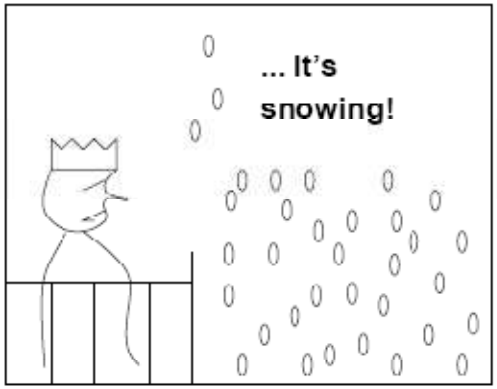
- Securing customer interaction always hard but critical
- About 1/3 of the projects did some sort of prototyping
- Organizations are changing much faster today, for economic and technological reasons

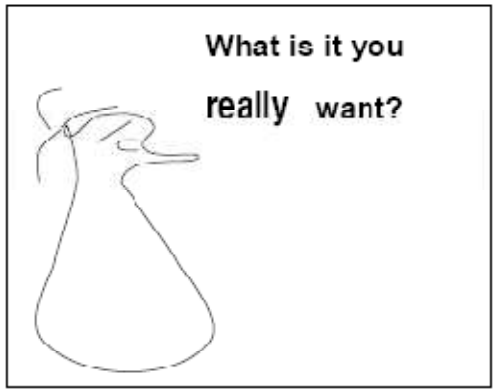
  => Requirements "evolution" a major concern

  => Requirements "traceability" a major concern

  => Requirements must deal with the environment (i.e., EM)
- **No such thing as the "problem"**

What is RE *Really* about?





What does the customer *really* want?

How to do RE?

Major themes of the course

Types of Errors

- **Jet Propulsion Laboratory (JPL) [Kelly92]**

  nearly 2/3 requirements defects are due to omission of key information

  => *techniques of completeness*

  => *setting the scope*

  => *identifying system operations*

  => *identifying goals/objectives*

- **The Naval Research Laboratory**

  Navy A-7E aricraft's operational flight program [Boehm, DeMarco, et al.]

  ongoing research since the mid-70's

  77% of requirements errors were nonclerical

  49% incorrect facts
  31% omission
  13% inconsistency
  5% ambiguity
  2% misplacement

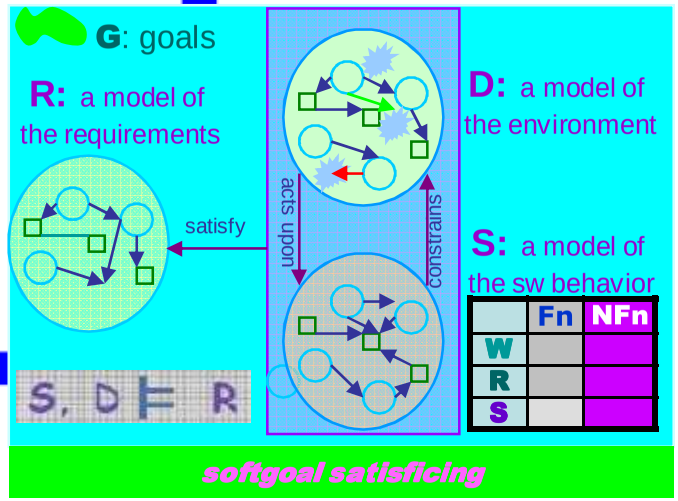
  33% of requirements errors were detected by manual review
  automated tools can detect a significant number of errors

- **Error detection and removal**

  in defining

- **Why, What & How**

Lawrence Chung



$M^G, Prog^G \models S^G; S^G, D^G \models R^G; R^G, D^G \models G; (G \models \neg P) \vee (G \models\sim \neg P)$

How to do RE?

Major themes of the course

- **Error detection and removal**

Conceptuality <		Formality >
Natural language	Semi-formal notations	Formalisms
customer/users		modellers/ specifiers
defects ↑		defects ↓

- **A requirements engineer should:**

  understand the problem
  express/translate the needs *through interaction*

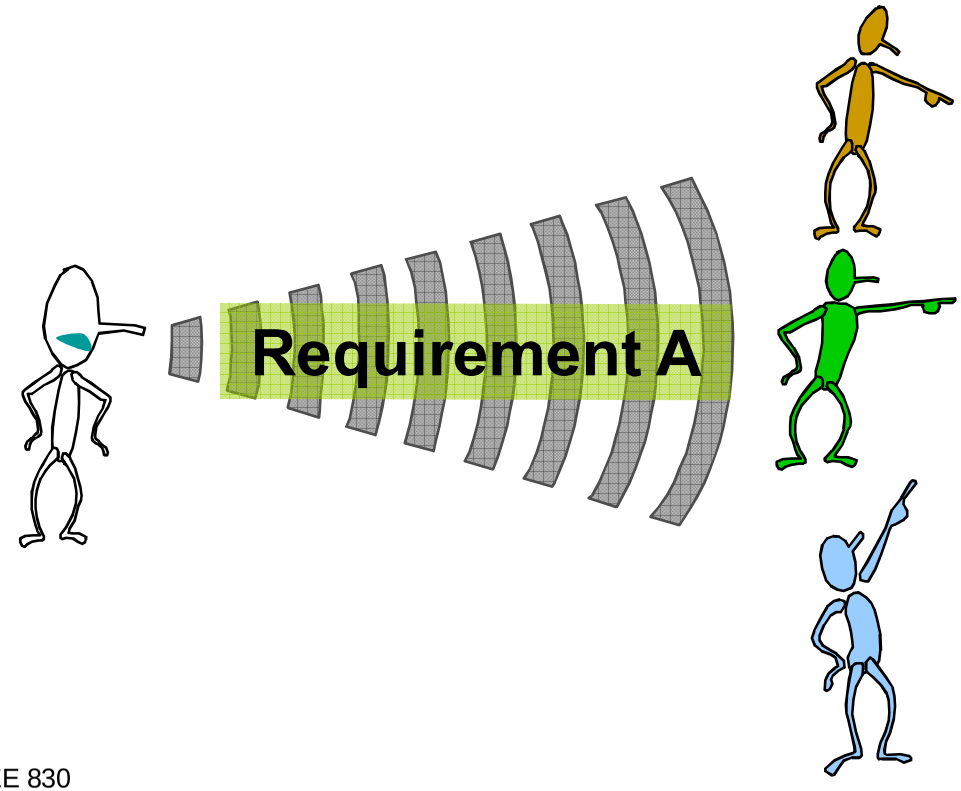
  be knowledgeable in relevant technologies toward possible solutions
  decide on a solution

  manage acquisition and evolution

A requirements engineer = SEeer, Customer Engineer, Req/Sys/Prog Analyst, Sys. Engineer, Sys. Engineer for Business, Req. Modeller, Assessor of Safety Control & Protection Systems, System Designer, RE Process Manager, Knowledge Engineer, Specifier, etc.

Qualities of a Requirement: Unambiguous

- A requirement is unambiguous if it has only one ***interpretation***.



"A shall do B to C"

"A shall do B to C"

"A shall do B to C"

ref - IEEE 830

Explore Ambiguity: Dictionary Definitions

Mary had a little lamb.

had - Past of have

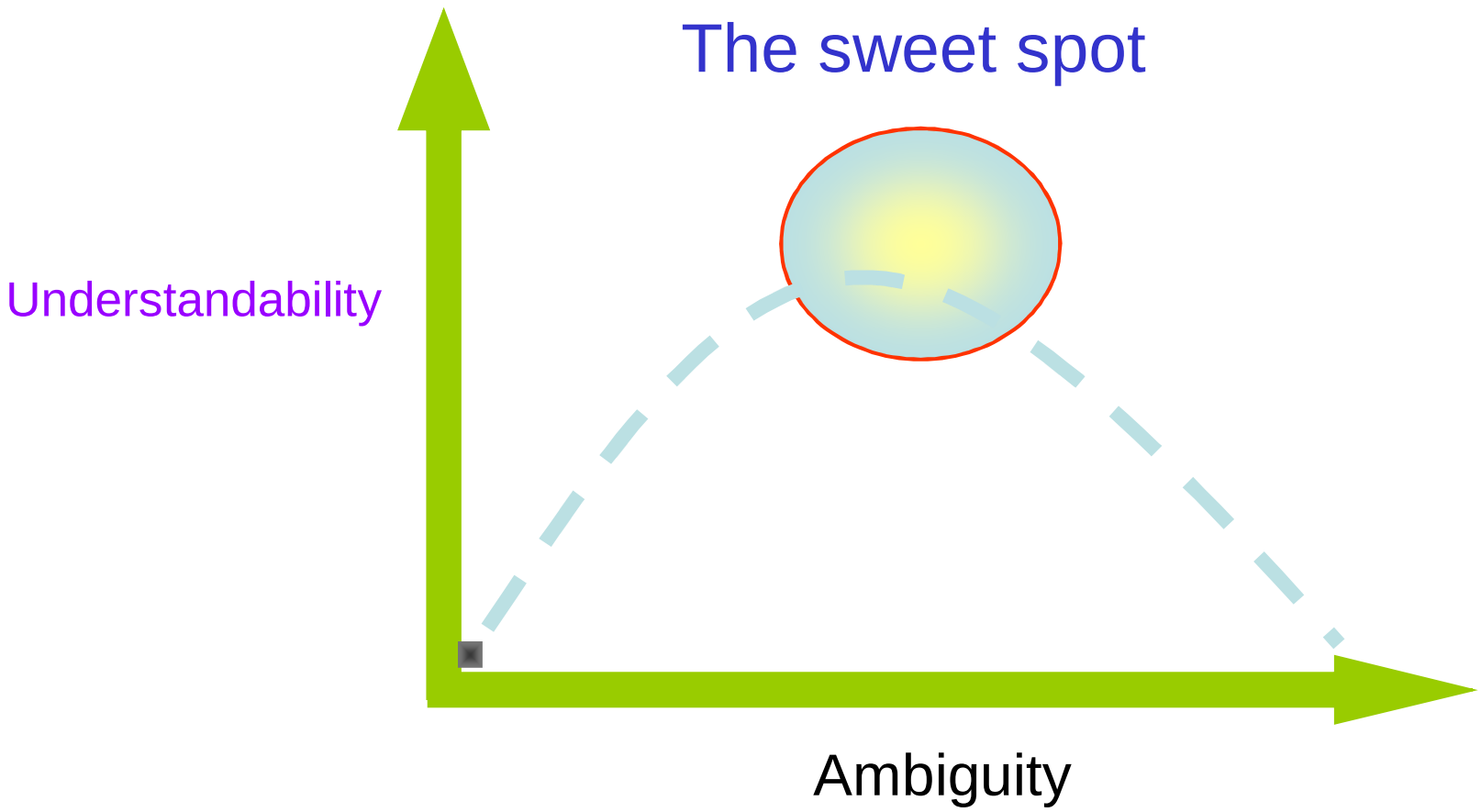
have - 1a: To hold in possession as property
4a: To acquire or get possession of: OBTAIN (best to be had)
4c: ACCEPT; to have in marriage
5a: To be marked or characterized by (have red hair)
10a: To hold in a position of disadvantage or certain defeat
10b: TRICK, FOOL (been had by a partner)
12: BEGET, BEAR (have a baby)
13: To partake of (have dinner)
14: BRIBE, SUBORN (can be had for a price)

lamb - 1a: A young sheep esp. less than one year old or without permanent teeth
1b: The young of various other animals (as smaller antelopes)
2a: A person as gentle or weak as a lamb
2b: DEAR, PET
2c: A person easily cheated or deceived especially in trading securities
3a: The flesh of lamb used as food

Explore Ambiguity: Analysis

Have	Lamb	Interpretation
1a	1a	Mary owned a little sheep under one year of age or without permanent teeth.
4a	1a	Mary acquired a little sheep under one year of age or without permanent teeth.
5a	1a	Mary is the person who owned a little sheep under one year of age or without permanent teeth.
10a	1a	Mary held a little sheep under one year of age or without permanent teeth in a position of disadvantage.
10b	1a	Mary tricked a little sheep under one year of age or without permanent teeth.
12	1b	Mary gave birth to a young antelope.
12	2a	Mary is (or was) the mother of a particular small, gentle person.
13	3a	Mary ate a little of the flesh of lamb.
14	2c	Mary bribed a small person trading in securities who was easily cheated.

Explore Ambiguity: An Observation



Qualities of Software Requirements

[Leffingwell & Widrig (1999). IEEE 830-1993, § 4.3.2, 1994]

- **Correct**
  - Is a true statement of something the system must do.
- **Complete**
  - Describes all significant requirements of concern to the user.
- **Consistent**
  - Does not conflict with other requirements.
- **Unambiguous**
  - Is subject to one and only one interpretation.
- **Verifiable**
  - Can be tested cost effectively.
- **Ranked for importance and stability**
  - Can be sorted based on customer importance and stability of the requirement itself.
- **Modifiable**
  - Changes do not affect the structure and style of the set.
- **Traceable**
  - The origin of each requirement can be found.
- **Understandable**
  - Comprehended by users and developers.

Qualities of a Requirement: Verifiable

IEEE 830-1993, § 4.3.2, 1994

- A requirement is verifiable if:
  - There exists some finite, cost-effective process with which a person or machine can check that the product meets the requirement.

- The system supports up to 1,000 simultaneous users.
- The system shall respond to an arbitrary query in 500 msec.
- The color shall be a pleasing shade of green.
- The system shall be available 24 x 7.
- The system shall export view data in comma-separated format, according to the IEEE specification.

Are these requirements verifiable? If not, what is a better way to state them?

Parts of Lecture Notes Come From

Some basic material

- Introduction to RE [Davis.Ch1; LK.Ch1]
- Requirements Engineering Processes [LK.Ch2]
  - RE evolutionary process
  - RE basic process
  - RE in software lifecycle
  - Process vs. product specifications
- Requirements Analysis, Modeling and Specification [LK.Sec4.1 -4.2]
- Requirements Elicitation: [LK.Ch3]
  - Scenario Analysis [Martin & Odell. Ch28]
- Enterprise Requirements: [LK.Sec4.3]
  - Modeling Techniques
    - Agent-oriented enterprise modeling
    - Business modeling with UML [Leffingwell and Eidrig, 2003]
    - Conventional enterprise modeling techniques}
  - AS-IS or TO-BE?
- Functional Requirements: Semi-formal Structural Models [LK.Sec4.3; Davis.Ch2]
  - Structured analysis
- Functional Requirements: Formal Structural Models
  - A Formal OO-RML/Telos
    - Deficiencies of SA
    - RML/Telos Essentials
    - A Formalization
    - A Brief Survey of FMs
  - Metamodeling
    - Models, Metaclasse, Metamodels
    - Metamodels for UML and other notations
- Functional Requirements: Behavioral Models [Davis.Ch4]
  - Decision-oriented
  - State-oriented
  - Function-oriented behavioral models
- Non-Functional Requirements [CNYM, 2000; LK.Ch5; Davis.Ch6]
  - Why NFRs
  - What – definitions and classifications
  - How – product- and process-oriented approaches

Another possible topic: Model Checking

Parts of Lecture Notes Come From

- Plus other references as in the syllabus
- Plus some selected articles (on the next slide)
- Plus articles and web resources as indicated in individual modules

Parts of Lecture Notes Come From

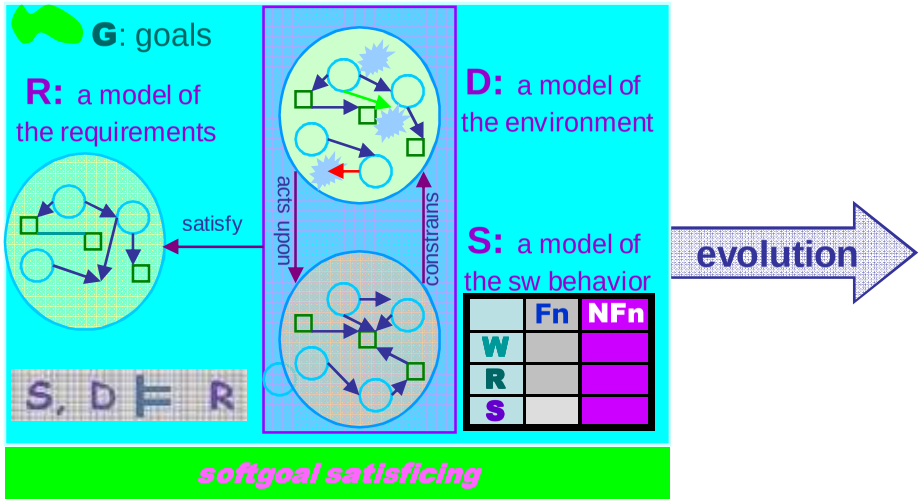
Some selected articles

- A. I. Anton and C. Potts, "Functional paleontology: system evolution as the user sees it," *Proc., 23rd IEEE Int. Conference on Software Engineering (ICSE'01),* Toronto, Canada, 12-19 May, 2001. pp. 421-430.
- B. Boehm H. In, "Identifying quality-requirement conflicts," *IEEE Software* 13 (2) 25-35. March 1996.
- M. S. Feather and S. L. Cornford, "Quantitative risk-based requirements reasoning," *Requirements Engineering*, Vol 8, pp. 248–265.
- R. G. Fichman and C. F. Kemerer, "Object-oriented and conventional analysis and design methodologies," *IEEE Computer*, 25 (10) 22 -39, Oct. 1992.
- X. Franch, "Systematic formulation of non-functional characteristics of software," *Proc., 3rd Int. Conference on Requirements Engineering, (ICRE'98).* 6-10 April 1998. pp.174-181. IEEE Computer Society Press.
- M. Glinz, "Problems and Deficiencies of UML as a Requirements Specification Language," *Proc. of the 10th Int. Workshop on Software Specification and Design (IWSSD-10),* 2000.
- J. Goguen and C. Linde, "Techniques for Requirements Elicitation," *Proc., 1st IEEE Int. Symposium on Requirements Engineering (RE'93)* San Diego, California, USA, pp. 152-164. IEEE Computer Society Press.
- O. C. Z. Gotel and A. C. W. Finkelstein, "Contribution Structures," *Proc. of the 2nd IEEE Int. Symposium on Requirements Engineering (RE'95),* York, UK, pp. 100-107, March 27-29 1995. IEEE Computer Society Press.
- S. Greenspan, J. Mylopoulos and A. Borgida, "On formal requirements modeling languages: RML revisited," *Proc., 16th Int. Conference on Software Engineering (ICSE-16)* pp135 -147. IEEE Computer Society Press.
- M. P. E. Heimdahl and N. G. Leveson, "Completeness and Consistency in Hierarchical State-Based Requirements," *IEEE Transactions on Software Engineering,* Vol 22 No 6, June 1996.
- C. L. Heitmeyer, R. D. Jeffords and B. G. Labaw, "Automated Consistency Checking of Requirements Specifications," *ACM Transactions on Software Engineering and Methodology,* 5(3), 231-261.
- A. M. Hickey and A. M. Davis, "Elicitation technique selection: how do experts do it?," *Proc., 11th IEEE Int. Requirements Engineering Conference (RE'03),* Monterey Bay, USA, 8-12th Sept. 2003, pp. 169-178. IEEE Computer Society Press.

Parts of Lecture Notes Come From

Some selected articles

- M. Jackson, “The Meaning of Requirements,” *Annals of Software Engineering*, Vol 3, pp5-21, Baltzer Science Publishers. 1997.
- A. van Lamsweerde, "Requirements engineering in the year 00: a research perspective", *Proc., the 22nd Int.Conference on Software Engineering (ICSE'00),* Limerick, Ireland, 5-9th June, 2000, pp5-19. IEEE Computer Society Press.
- A. van Lamsweerde, “Goal-Oriented Requirements Engineering: A Guided Tour. *Proc., 5th IEEE Int. Symposium on Requirements Engineering (RE'01),* Toronto, Aug., 2001, pp. 249-263. IEEE Computer Society Press.
- N. Maiden and S. Robertson, “Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System,” *Proc., 13th IEEE International Requirements Engineering Conference (RE'05),* Paris, France, Aug 29 - Sept 2, 2005.
- J. Mylopoulos, L. Chung and B. Nixon, “Representing and using nonfunctional requirements: a process-oriented approach,” *IEEE Transactions on Software Engineering*, Vol 18, Issue 6, June 1992, pp. 483 - 497.
- B. A. Nuseibeh and S. M. Easterbrook, "Requirements Engineering: A Roadmap", In A. C. W. Finkelstein (ed.) *The Future of Software Engineering*. (Companion volume to the proc. of ICSE'00). IEEE Computer Society Press.
- D. L. Parnas, “Formal Methods” Technology Transfer Will Fail,’ *Journal of Systems and Software*. Vol. 40, Issue: 3. March, 1998. pp. 195-198
- C. Potts and W. C. Newstetter, “Naturalistic inquiry and requirements engineering: reconciling their theoretical foundations,” *Proc., 3rd IEEE Int. Symposium on Requirements Engineering (RE'97),* Annapolis, USA, pp. 118 -127. IEEE Computer Society Press.
- B. Ramesh and M. Jarke, “Toward reference models for requirements traceability,” *IEEE Transactions on Software Engineering,* Volume: 27 1, January 2001, pp. 58 -93.
- A. Sutcliffe, “Scenario-based requirements engineering,” *Proc., 11th IEEE Int. Requirements Engineering Conference (RE'03),* Monterey Bay, USA, 8-12th Sept. 2003, Pages: 320- 329. IEEE Computer Society Press.
- J. Whittle and J. Schumann, “Generating statechart designs from scenarios,” *Proc., 22nd IEEE Int. Conference on Software Engineering (ICSE-00),* Limerick, Ireland, 4-11 June 2000. Pages: 314-323.
- W. M. Wilson, L. H. Rosenberg and L. E. Hyatt, “Automated Analysis of Requirement Specifications,” *Proc. of the 19th Int. Conference on Software Engineering (ICSE-97),* Boston, MA, May 17-23, pp.161 -171.
- E. S. K. Yu, “Towards modelling and reasoning support for early-phase requirements engineering,” *Proc., 3rd IEEE Int. Symposium on Requirements Engineering (RE'97),* Annapolis, USA, pp 226 -235. IEEE Computer Society Press.
- P. Zave and M. Jackson, “Four Dark Corners of Requirements Engineering,” *ACM Transactions on Software Engineering and Methodology* 6(1) 1-30. ACM Press. 1997.



$M^G, Prog^G \models S^G; S^G, D^G \models R^G; R^G, D^G \models G; (G \models \neg P) \vee (G \mid\sim \neg P)$