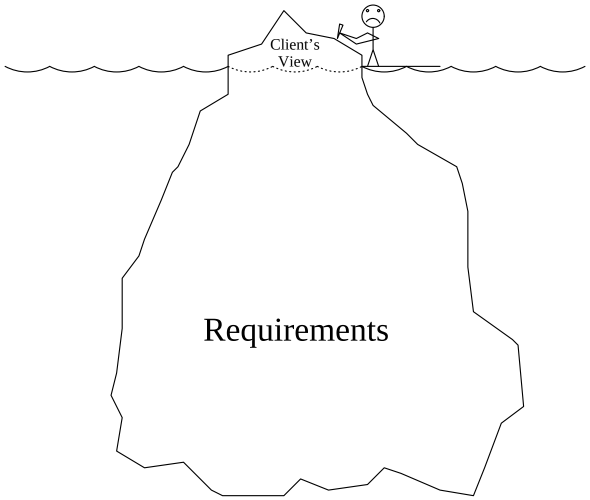
The Requirements Iceberg and Various Icepicks Chipping at it

Daniel M. Berry



Requirements Engineering (RE): the Problems and an Overview of Research

Outline

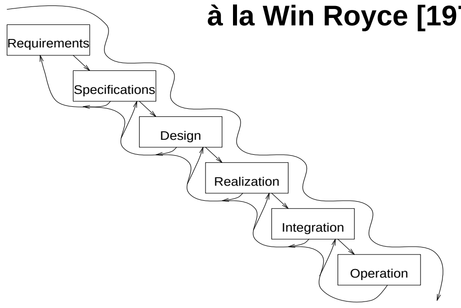
Lifecycle Models
Conceptual Problems
Cost
Myths and Realities
E-Type Systems
Where Do Requirements Come From?
Formal Methods?
Software and Mathematical Theories
Requirement Errors
Requirements and Other Engineering
Requirements Engineering Lifecycle

Overview of Research
- **Earlier and Later**
- **Elicitation**
- **Analysis**
- **Natural Language Processing**
- **Tools**

Future

Traditional Waterfall Lifecycle

à la Win Royce [1970]



Only one slight problem: It does not work!

Problems with Waterfall Model-1

The main problem, from the requirements point of view, of the waterfall model is the feeling it conveys of the sanctity, inviolability, and unchangeability of the requirements, as suggested by the following drawing by Barry Boehm [1988].

Problems with Waterfall Model-2

This view does not work because requirements *always* change:

- **partially from requirements creep (but good project management helps)**
- **partially from mistakes (but prototyping and systematic methods help)**
- **partially because it is inherent in software that is used (the concept of E-type systems is discussed later!)**

Fred Brooks about Waterfall

In ICSE '95 Keynote, Brooks [1995] says "The Waterfall Model is Wrong!"

- **The hardest part of design is deciding *what* to design.**
- **Good design takes upstream jumping at every cascade, sometimes back more than one step.**
- **Even the U.S. DoD finally knows this, to wit *Defense Science Board Study*, Kaminski Committee, June 1994.**

Fred Brooks also says:

"There's no silver bullet!" [Brooks 1987]

- **Accidents**
  - **process**
  - **implementation**

  i.e., details
- **Essence**
  - **Requirements**

"No Silver Bullet" (NSB)

- **The *essence* of building software is devising the conceptual construct itself.**
- **This is very hard.**
  - **arbitrary complexity**
  - **conformity to given world**
  - **changes and changeability**
  - **invisibility**

- Most productivity gain came from fixing *accidents*

  - really awkward assembly language
  - severe time and space constraints
  - long batch turnaround time
  - clerical tasks for which tools are helpful

- However, the essence has resisted attack!

  We have the same sense of being overwhelmed by the immensity of the problem and the seemingly endless details to take care of,

  and we produce the same brain-damaged software that makes the same stupid mistakes

  as 30 years ago!

Contracts

We all know how hard it is to get a contract just right ...

to cover all unanticipated situations before they are known.

Houses

We all know how hard it is to get a house plan just right before starting to build the house.

Homework Assignments

We all know how hard it is to get the specification of a programming homework assignment right, especially when the instructor must invent new ones for every run of the course.

Michael Jackson Says

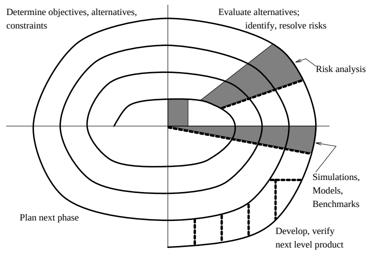
In a Requirements Engineering ’94 Keynote, Jackson [1994] says:

Two things are known about requirements:

1. They will change!
2. They will be misunderstood!

More Realistic Spiral Model

à la Barry Boehm [1988]

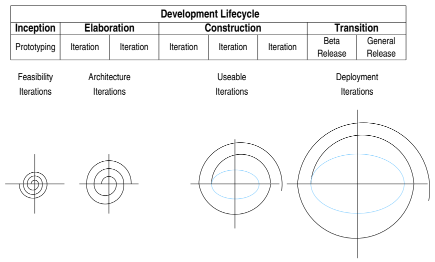


The waterfall can be considered one 360° sweep of the spiral.

New From a Royce

Walker, not Win

The Lifecycle Macroprocess



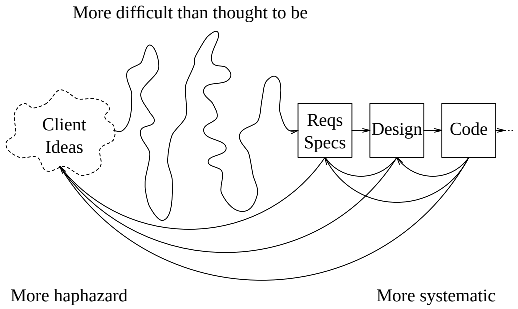
Walker Royce’s [1997] Waterfall Model, Part I

The Walkerfall Model

Relative Effort by Activity

Walker Royce's Waterfall Model, Part II

REAL Lifecycle for One Sweep

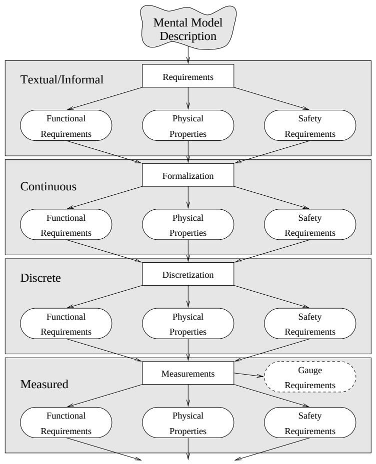


Agreed?

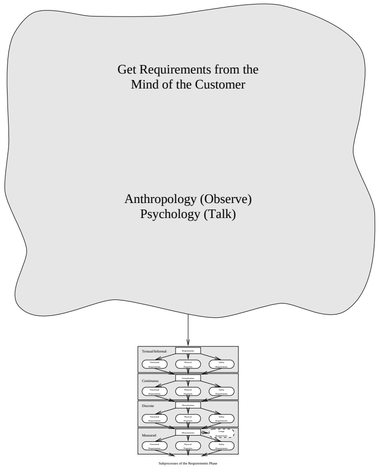
Montenegro's View

Sergio Montenegro described the reality by showing an older view of the requirements engineering process:

and then a newer view:

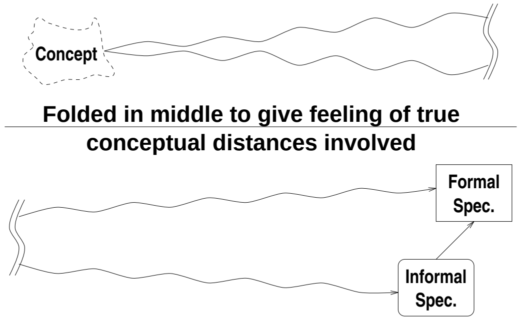


Subprocesses of the Requirements Phase



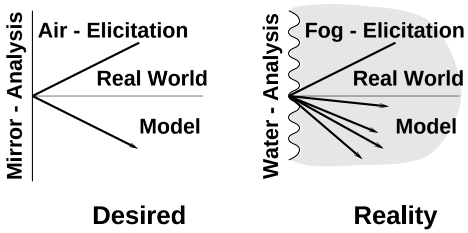
Relation between Getting and Formalizing Requirements

Distance: Concept $\rightarrow$ Specs



Modeling Difficulties

Brian Mathews [1998] has another view in terms of modeling



Brooks, Cont'd

Brooks adds, "The hardest single part of building a software system is deciding precisely what to build.... No other part of the work so cripples the resulting system if it is done wrong. No other part is more difficult to rectify later."

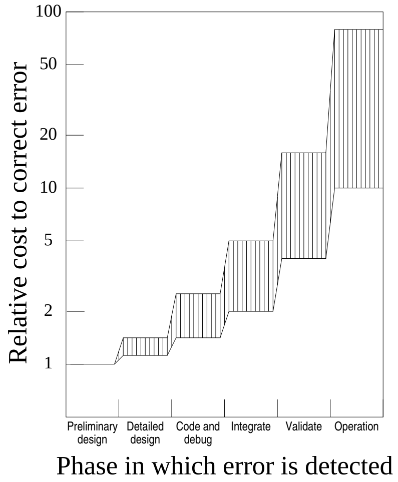
The BIG Question

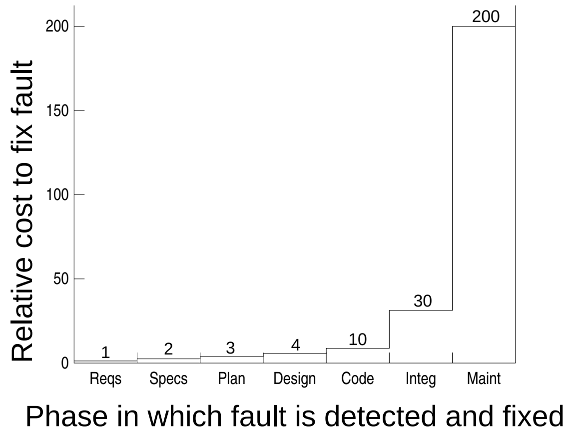
Why is it so important to get the requirements right early in the lifecycle? [Boehm 1981, Schach 1992]

We know that it is much cheaper to fix an error at requirements time than any time later in the lifecycle.

Cost to Fix Errors

Barry Boehm's (next slide) and Steve Schach's (slide after that) summaries of data over many application areas show that fixing an error after delivery costs two orders of magnitude more than fixing it it at RE time.





Conclusion...

Therefore, it pays to find errors during RE.

Requirements Production Myths-1

Several related myths:

> ***"You people start the coding while I go find out what the customer wants."***
>
> ***Requirements are easy to obtain.***
>
> ***The client/user knows what he/she wants.***

Myths-2

According to Ruth Ravenel, ...

> **The programmer who says the first line is suffering from the myth that the customer would be able to know what he or she wants and to say it just because the programmer asked.**

Myths-3

> **Most people (especially non-technically oriented) learn while doing; they've got to see some kind of prototype (even if it's only yellow stickies on a board) to *discover* what they want.**

First also expresses the nonsensical notion that somehow, coding can begin before it's known *what* to code.

May Not Even Have a Problem!

The client often says that he or she requires a specific solution of an unknown or nonexistent problem rather than any solution to a specific problem.

"*We gotta automate!*"

The problem with such a client is that there may not even be a problem that requires any solution, or if there is, other solutions, including non-computer, may be better!

Another Myth

After the requirements are frozen, ...

Ha!
Ha ha!
Ha ha ha!
Ha ha ha ha!

The only clients that are satisified and have stopped asking for changes are themselves frozen!

Why will requirements *never* be frozen?

First, Some More Reality

In a Requirements Engineering '94 Keynote, Michael Jackson says:

Two things are known about requirements:

1. ***They will change!***
2. **They will be misunderstood!**

Why will they *always* change?

E-Type Software

à la Meir Lehman [Lehman 1980]

A system that solves a problem or implements an application in some *real world* domain.

Once installed, an E-type system becomes inextricably part of the application domain, so that it ends up altering its own requirements.

E-Type Software-2

Example:

- **Consider a bank that exercises an *option* to automate its process and then discovers that it can handle more customers.**
- **It promotes and gets new customers, easily handled by the new system but beyond the capacity of the manual way.**
- **It cannot back out of automation.**
- **The requirements of the system have changed!**

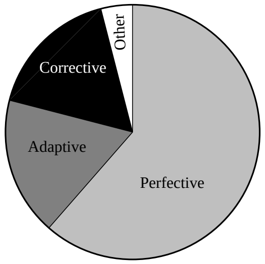
E-Type Software-3

Daily use of a system causes an irresistible ambition to improve it as users begin to suggest improvements.

Who is not familiar with that, from either end?

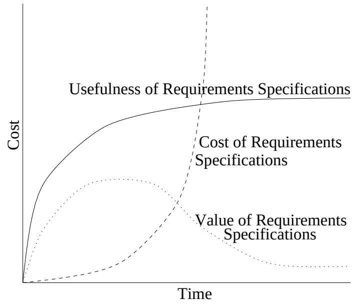
E-Type Software-4

In fact, data show that most maintenance is *not* corrective, but for dealing with E-type pressures!



Usefulness, Cost, Value

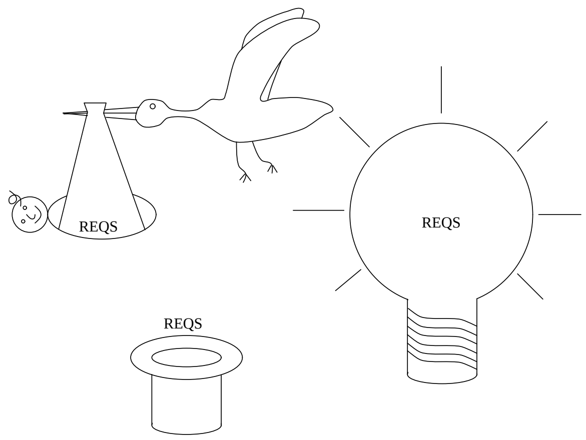
Janis Bubenko observed about requirements:



Whence Do Requirements Come?-1

Joe Goguen [1994] says, "It is not quite accurate to say that requirements are in the minds of clients; it would be more accurate to say that they are in the social system of the client organization. They have to be invented, not captured or elicited, and that invention has to be a cooperative venture involving the client, the users, and the developers. The difficulties are mainly social, political, and cultural, and not technical."

Whence Do Requirements Come-2



Whence Do Requirements Come-3

Interviewing does not really help because when asked what they do, most people will quote the official policy, and not what they actually do. Most of what they really do, which is not specified by the policy, is what they do in situations not covered by the policy.

We're not even talking about conscious, politically safe mouthing of the policy.

Whence Do Requirements Come-4

Many people simply do not remember the exceptions unless and until they actually come up. Their conscious model of what happens *is* the policy.

Therefore the requirements engineer has to *be* there when the exceptional situations come up in order to *see* what really happens.

Whence Do Requirements Come-5

Moreover, many people just do not know *why* they do something, saying only that it's done this way because the policy says so.

They very often do not even know why the policy is the way it is.

Whence Do Requirements Come-6

Moreover, many people just do not know *how* they do something, drawing a complete blank or saying only, "Watch me!".

For example, how do *you* ride a bicycle? Nu?

Whence Do Requirements Come-7

Don Gause and Jerry Weinberger tell the story of the woman who always cuts off ⅓ of a raw roast before cooking both pieces together.

She was asked "Why?" ...

Whence Do Requirements Come-8

In other words, the policy once made sense, but the person who formulated the policy, the reasons for it, and the understanding of the reasons are long since gone.

Whence Do Requirements Come-9

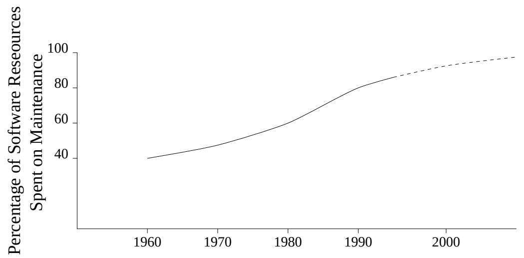
For example, many companies that have committed all data to a highly reliable data base continue to print out the summary in quintuplicate.

Why? At the time of automation, the five most senior members of the company, who long ago retired, refused to learn to use the computer to access the data directly!

Whence Do Requirements Come-10

Goguen further observes that most of the effort for a typical large system goes into maintenance.

In fact, Parnas [1994] has the data:

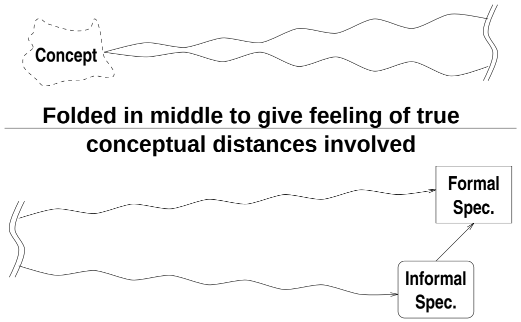


Growing Percentage of Maintenance Costs

Formal Methods Needed?-1

Some formal methodologists say that this is the fault of insufficient effort put into being precise in the early, specification stages of software development.

However, recall the conceptual distances involved:



Formal Methods Needed?-2

Goguen believes that "a deeper reason is that much more is going on during so-called maintenance than is generally realized. In particular, reassessment and re-doing of requirements, specification, and code, as well as documentation and validation, are very much part of maintenance...."

Formal Methods Needed?-3

Later, he adds, "it only becomes clear what the requirements really are when the system is successfully operating in its social and organisational context.... it is impossible to completely formalise requirements ... because they cannot be fully separated from their social context."

This is precisely the phenomenon of E-type systems.

Formal Methods Myths-1

Goguen has identified other myths about requirements, again based on the mistaken idea that the hard part about requirements are their specification.

Formal Methods Myths-2

If** only **you had written a formal specification of the system, you wouldn't be having these problems.

Mathematical precision in the derivation of software eliminates imprecision.

Formal Methods Myths-3

What is the reality?

Yes, formal specification are extremely useful in identifying inconsistencies in requirements specifications, especially if one carries out some minimal proofs of consistency and constraint or invariant preservation, ...

just as writing a program for the specification!

Formal Methods Myths-4

Don't get me wrong.

This formality is good, because it finds errors early, thus reducing the costs to fix them.

However, formal methods do *not* find all gaps in understanding!

Formal Methods Myths-5

As Eugene Strand and Warren Jones [1982] observe, “"Omissions of function are often difficult for the user to recognize in formal specifications”....

just as they are in programs!

Formal Methods Myths-6

von Neumann and Morgenstern (*Theory of Games*) say,

“There’s no point to using exact methods where there’s no clarity in the concepts and issues to which they are to be applied.”

Preservation of Difficulty

Indeed, Oded Sudarsky has pointed out the phenomenon of *preservation of difficulty*. Specifically, difficulties caused by lack of understanding of the real world situation are not eliminated by use of formal methods; instead the misunderstanding gets formalized into the specifications, and may even be harder to recognize simply because formal definitions are harder to read by the clients.

Bubbles in Wall Paper

Sudarsky adds that formal specification methods just shift the difficulty from the implementation phase to the specification phase. The “air-bubble-under-wallpaper” metaphor applies here; you press on the bubble in one place, and it pops up somewhere else.

One Saving Grace

Lest, you think I am totally against formal methods, they *do* have one positive effect, and it’s a BIG one:

Use of them increases the correctness of the specifications.

Therefore you find more bugs at specification time than without them, saving considerable money for each bug found earlier rather than later.

Analogy with Math Theory-1

Building new software is like building a new mathematical theory from the ground up:

- **Requirements gathering: deciding what is to be assumed, defined, and proved**
- **Development as a whole: assuming assumptions, defining the terms, and proving the theorems**

Analogy with Math Theory-2

- **Design: determining the sequence of assumptions, definitions, and theorems to build the theory**
- **Implementation: carrying out the designed theory and proving the theorems**

Analogy with Math Theory-3

Mathematical papers and books show only the results of the implementation,

This implementation is what is considered the mathematics, and not the requirements gathering and design that went into it.

But I know, from my own secret past life as a mathematician, that the hard, time consuming parts are the requirements gathering and design.

Math vs. SW Development-1

- **Software is usually developed under strict time constraints, and mathemtics is usually not.**
- **Mathematics development is subjected to error-eliminating social processes, and software development is subjected to a lot less.**

Math vs. SW Development-2

- **Mathematics is written for a human audience that is very forgiving of minor errors so long as it can see the main point; software is written for the computer that is very literal and unforgiving of minor errors**
  - **For people, UKWIM works, and they accept imprecision.**
  - **For computers, UKWIM and DWIM do not work, and they do not accept imprecision.**

Another Implication of Growing Maintenance Costs-1

For many programs, which are more and more often enhancements of legacy software, any original requirements specifications that may have existed are long gone.

The original programmers are long gone.

The old requirements have to be inferred from the software.

Another Implication of Growing Maintenance Costs-2

What is inferred may not capture all features.

Also the obvious requirement of not impacting existing functions in the enhancement is very easy to state, but, oh, so hard to satisfy.

Errors and Requirements

According to Barry Boehm [1981] and others, around 75-85% of all errors found in SW can be traced back to the requirements and design phases.

Errors and Requirements, Cont'd

Either

- **the erroneous behavior is required because the situation causing the error was not understood or expressed correctly, or**

- **the erroneous behavior happens because the requirements simply do not mention the situation causing the error, and something not planned and not appropriate happens.**

Paradox

Fred Brooks observed that a general purpose system is harder to design than a special purpose product.

RE is itself E-type

Janis Bubenko [1995] says:

Requirements Engineering deals with wicked problems,

and is itself wicked

and is thus modified by its own partical solution

Therefore, Requirements Engineering is E-type!

RE is Hard

Despite the clear benefits of getting requirements complete, right, and error-free early, they are the hardest part of the system development lifecycle to do right because:

- **we don't always understand everything about the real world that we need to know,**

- **we may understand a lot, but we cannot express everything that we know,**

RE is Hard, Cont'd

- we may think we understand a lot, but our understanding may be wrong,
- requirements change as client's needs change,

RE is Hard, Cont'd

- requirements change as clients and users think of new things they want, and
- requirements of a system change as a direct result of deploying the system, as pointed out by Meir Lehman.

Sources of RE Difficulties

- RE is where informal meets formal (says Michael Jackson).
- Many requirements are created, not found.
- Users, buyers, even developers may be unknown.
- Stakeholders have conflicting objectives.
- Multiple views exist.
- Inconsistency must be tolerated, for a while.
- Requirements evolve during and after development.

Study of Requirement Errors -1

à la Martin & Tsai [1988]

Experiment to identify lifecycle stages in which requirement errors are found

Polished 10-page requirements for centralized railroad traffic controller

Ten 4-person teams of software engineers

User believed that teams would find only 1 or 2 errors

Study of Requirement Errors -2

92 errors, some very serious, were found!

Average team found only 35.5 errors, i.e., it missed 56.5 to be found downstream!

Many errors found by *only* one team!

Errors of greatest severity found by fewest teams!

The TRUTH About Methodology Literature

(Body of slide intentionially left mostly blank)

Nu?

Lest we forget

Roberto Tom Price reminds us not to forget the requirement engineer's

- **imagination**
- **ideas**
- **suggestions**

like an architect for a new building, following input from the client

RE and Other Engineering-1

Speaking of architects and other engineers that get requirements from clients, ...

While software requirements gathering has much in common with requirements gathering for buildings, bridges, cars, etc., there are significant differences in:

- **the flexibility and malleability of the medium, and**
- **the degree to which basic assumptions are on the table, up for grabs.**

RE and Other Engineering-2

Michael Jackson [1995] considers the more traditional engineering disciplines.

Civil Engineering
Mechanical Engineering
Aeronautical Engineering
Electrical Engineering

Their engineers make machines by describing them and *then* building them.

RE and Other Engineering-3

Software engineers make machines merely by describing them.

Software is an intangible, infinitely malleable medium.

RE and Other Engineering-4

To build a new car from requirements the way software is built from requirements would be called *totally rethinking the automobile*. In fact, each new kind of car is really a minor perturbation of existing kinds of cars.

RE and Other Engineering-5

Perhaps, we should do much more minor perturbation of existing software, i.e., practice reuse, but in many cases we cannot simply because we are developing software for an entirely new application.

Bottom Line

The notions that

- **one can derive requirements**
  or
- **even interview a few people to get requirements**

are patent nonsense.

Ample Evidence That

- **The later an error is detected, the more it costs to correct it.**
- **Many errors are made in requirements elicitation and definition.**
- **Many requirements errors can be detected early in the lifecycle.**
- **Typical errors include incorrect facts, omissions, inconsistencies, and ambiguities.**
- **Requirements problems are industry's biggest concern.**

Reliability, Safety, Security, & Survivability

We know that we cannot program reliability, safety, security, and survivability into the code of a system only at implementation time. They must be required from the beginning so that consideration of their properties permeate the entire system development.

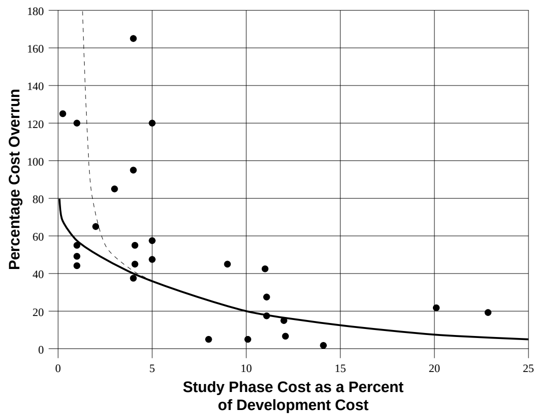
The wrong requirements can preclude coding them at implementation time.

RE & Project Costs

The next slide shows the benefits of spending a significant percentage of development costs on studying the requirements.

It is a graph from by Kevin Forsberg and Harold Mooz relating percentage cost overrun to study phase cost as a percentage of development cost in 25 NASA projects.

Project Costs, Cont'd



Project Costs, Cont'd

The study, performed by W. Gruhl at NASA HQ includes such projects as

- **Hubble Space Telescope**
- **TDRSS**
- **Gamma Ray Obs 1978**
- **Gamma Ray Obs 1982**
- **SeaSat**
- **Pioneer Venus**
- **Voyager**

OK, OK, You're Convinced!!!

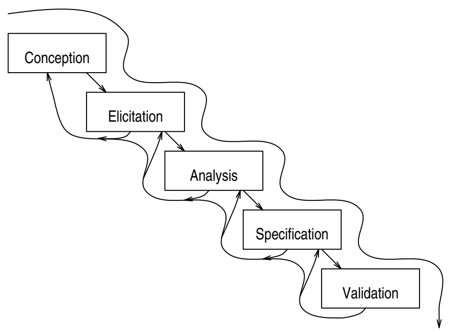
So what do we DO about it?

What research is being done to solve the problems?

First, recognize that the problem is HARD!

Second, recognize that requirements engineering has its own lifecycle

Requirements Engineering (Sub)Lifecycle-1



(Sub)Lifecycle-2

This, of course, is an idealization, just as much as the original waterfall.

Reality is that there is a spiral, with each sweep going through this entire subwaterfall, and all the steps in the sweep happening concurrently.

(Sub)Lifecycle-3

And most importantly, ...

There are no real solutions yet!

It is very much an art form.

Research Topics

Earlier Work, prior to mid-80s

Later Work, mostly after mid-80s

Some temporal overlap, because as we will see, the work is classified by nature, and that nature has changed slowly.

Earlier Work -1

Languages and Tools:

PSL/PSA [Teichroew 1977]
SADT [Ross and Ross & Schoman 1977]
RSL [Alford 1977]
RDL [Winchester 1982]
PAISley [Zave 1982]
RML [Borgida, Greenspan & Mylopolous 1985]
IORL [Salton & McGill 1983]

Earlier Work -2

Alan Davis wrote the book! *Requirements Analysis and Specification* [1990]

Focus was on analysis and specification, *not* on elicitation

Later Work

More consideration of elicitation

Recognition of importance of sociology and psychology

I apologize in advance if I have left out things about which I am not aware.

- **Elicitation**
- **Analysis**
- **Natural Language Processing**
- **Tools and Environments**

Global View-1

***Requirements Engineering* is:**

How to squeeze requirements out of the client's mind without damaging the client!

***Elicitation* is:**

How to squeeze information out of the client's mind without damaging the client!

Global View-2

***Analysis* is:**

How to squeeze as much additional information as possible out of what has been obtained by squeezing the client!

***Natural Language Processing* (NLP) is concerned with:**

How to automate as much of the analysis squeezing as possible

Global View-3

Tools and Environments **deal with:**

How to automate the storage of information before and after analytic squeezing as well as all kinds of squeezing!

Use of Exemplars in RE Research

Many areas of SE use published exemplars, eg KWIC Index System, for research case studies.

Since the problem is *how* to get the information for requirements, published exemplars are too polished and too late.

What is normally done to prepare exemplars for publication *is* the subject of requirements engineering.

Elicitation Overview-1

The gist of the specific work is:

- **Interviewing does not get all the information that is needed.**
- **The requirements engineer must get into the client's work place, blend into the woodwork or among the employees, and observe, learn, and question, in order to overcome ignorance of the problem domain.**

Elicitation Overview-2

- **The requirements engineer must become an employee of the client to learn the ropes well enough to understand the underlying rationale behind the way things are done.**
- **The requirements engineer should parlay his or her ignorance into on-the-spot questions that expose tacit assumptions and special cases.**
- **Don't tell them what you mean; show them! This works in both directions!**

Elicitation Specifics-1

- Ignorance hiding in elicitation and analysis [Berry 1980, 1983]
- Concept of abstract user and prototype elicitation management tool [Burstin 1984]
- Brainstorming [Gause & Weinberg 1989]
- Joint Application Development[Hill 1990 and others]
- Contextual Inquiry, an anthropological approach to understanding client [Holtzblatt & Jones 1990]

Elicitation Specifics-2

- Study of discourses in elicitation techniques, e.g., interviews [Goguen & Linde 1993]
- Storyboarding & paper mockups [Zahniser 1993]
- Importance of ignorance in elicitation [Berry 1995]

Analysis Overview-1

- Basic idea of analysis is to
  - derive implications of,
  - resolve inconsistencies in, and
  - determine what is missing from

  the information that has been gathered so far so that follow up questions may be asked.
- The requirements engineer must constantly attempt to validate his or her understanding with the client or user.

Analysis Overview-2

- Client’s and Users’ validation responses to prototypes are far more credible than to long written specifications.
- It is essential to be able to trace the history of a requirement from its conception through its specification and on to its implementation.

Analysis Overview-3

- **Error checking, handling, and prevention will not happen in a program *unless* they are explicitly required of the program; this is particularly so in safety-critical software for which the dangers are not readily apparent.**
- **While being specified, requirements *are* logically inconsistent.**

Analysis Specifics-1

- **Software safety fault isolation techniques [Leveson 1986]**
- **Brainstorming [Gause & Weinberg 1989]**
- **Domain modeling [many, surveys: Kang *et al* 1990, Lubars *et al* 1993]**
- **Prototyping for requirements discovery and validation [Wasserman et al 1984 & 1986, Bowers & Pycock 1993, Luqi 1992]**
- **Object-orientation as a natural view [many, including: Coad & Yourdon 1991, Zucconi 1993]**

Analysis Specifics-2

- **Viewpoint Resolution [Leite 1987, Easterbrook 1993]**
- **System bounding [Drake & Tsai 1994]**
- **Issue-based information system (IBIS) [Burgess-Yakemovic & Conklin 1990]**
- **Requirements traceability [Gotel & Finkelstein 1994]**
- **Living with logical inconsistencey of specs [Easterbrook & Nuseibeh 1995, 1996, Hunter & Nuseibeh 1997]**

NLP Overview-1

The total amount of information to deal with for any real problem is HUGE and repetitititive.

We desire assistance in extracting useful information from this mass of information.

NLP Overview-2

We would like the extracted information to be

- **summarizing,**
- **meaningful, and**
- **covering.**

From 500 pages, we want 5 pages containing *all* and *only* the meaningful information in the 500 pages.

NLP Overview-3

We prefer less summarization and occasional meaningless stuff than to lose some meaningful stuff, because in any case, a human will have to read the output and at that time can filter out the meaningless stuff.

Stupidity is preferred to intelligence if the latter can lose information as a result of it not ever being perfect.

NLP Specifics-1

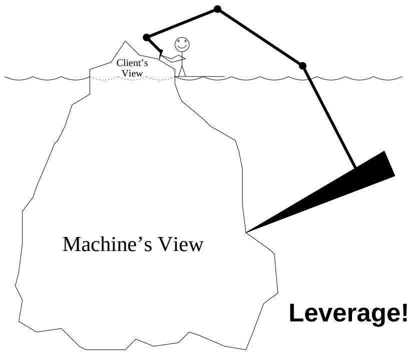
- **Restricted natural language processing of requirements ideas to get specifications [Saeki, Horai, *et al* 1987]**
- **Natural language abstraction identification with lexical affinities [Maarek 1989]**
- **Application domain lexicon building and tools [Leite & Franco 1993]**

NLP Specifics-2

- **AI-based natural language processing [Ryan 1993]**
- **Nonintelligent, fully covering natural language abstraction identification using signal processing techniques [Goldin 1994]**

Martin Feather's View of RE Tools

The Requirements Iceberg and Various Machine-Assisted Icepicks Chipping at It



Tools & Environments Overview-1

Even with summarizing tools,
the amount of information that the requirements engineer must deal with is HUGE,

the number of relations between the individual items of information is HUGE^2, and

the number of relations between the individual relations is HUGE^4...

Tools & Environments Overview-2

So we want an environment filled with useful tools that help manage all the information needed to produce a requirements specification from the first conceptions, and then to be useable for the rest of the lifecycle.

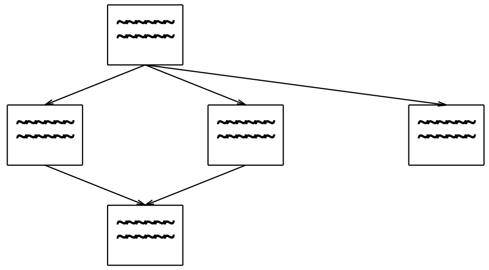
Tools & Environments Specifics-1

- **Graphical Issue-Based Information System (gIBIS) [Burgess-Yakemovic & Conklin 1990]**
- **Hypermedium as requirements engineering environments [Potts & Takahashi 1993, Kaindle 1993]**

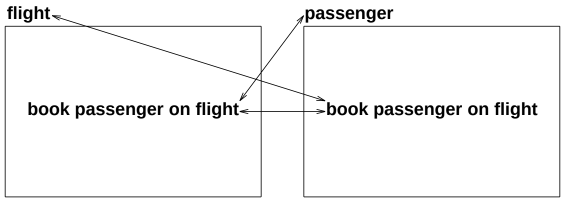
Tools & Environments Specifics-2

- **Full spectrum, including traceability analysis, requirements engineering tool, READS [Smith 1993]**
- **Multimedia hypermedium as requirements engineering environments [Wood, Christel & Stevens 1994]**

Some Views of Hypermedia Tools

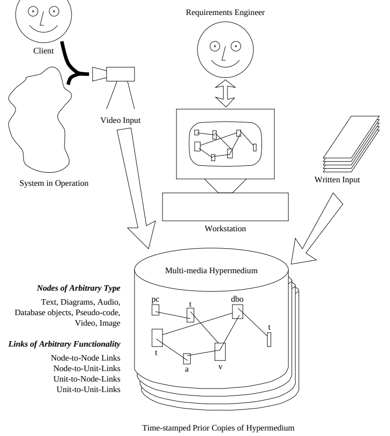


Network of Nodes



Links

Multimedia Hypermedium



Future

Lots more work is needed

Please join in!!

Conferences and Workshops

- **International Workshop on Software Systems Design 1–9**
- **Requirements Engineering ’93, ’95 & ’97**
- **International Conference on Requirements Engineering ’94, ’96 & ’98**
- **IFIP WG 2.9 Software Requirements Engineering ’95, ’96 & ’97**

Journals

- ***Software Practice and Experience***
- ***Journal of Systems and Software***
- ***IEEE Software***
- ***Journal of Automated Software Engineering***
- ***Requirements Engineering Journal***
- ***ACM Interactions***

Research Networks

- **RENOIR, Requirements Engineering Network of Excellence, sponsored by European Union**

Web Pages

- **Requirements Engineering Newsletter (these are the files named renl*: ftp://ftp.cs.city.ac.uk/pub/requirements/**
- **Requirements Engineering Bibliography: http://www.inf.puc-rio.br/˜bdbib/**
- **RENOIR: http://web.cs.ucl.ac.uk/research/renoir/**