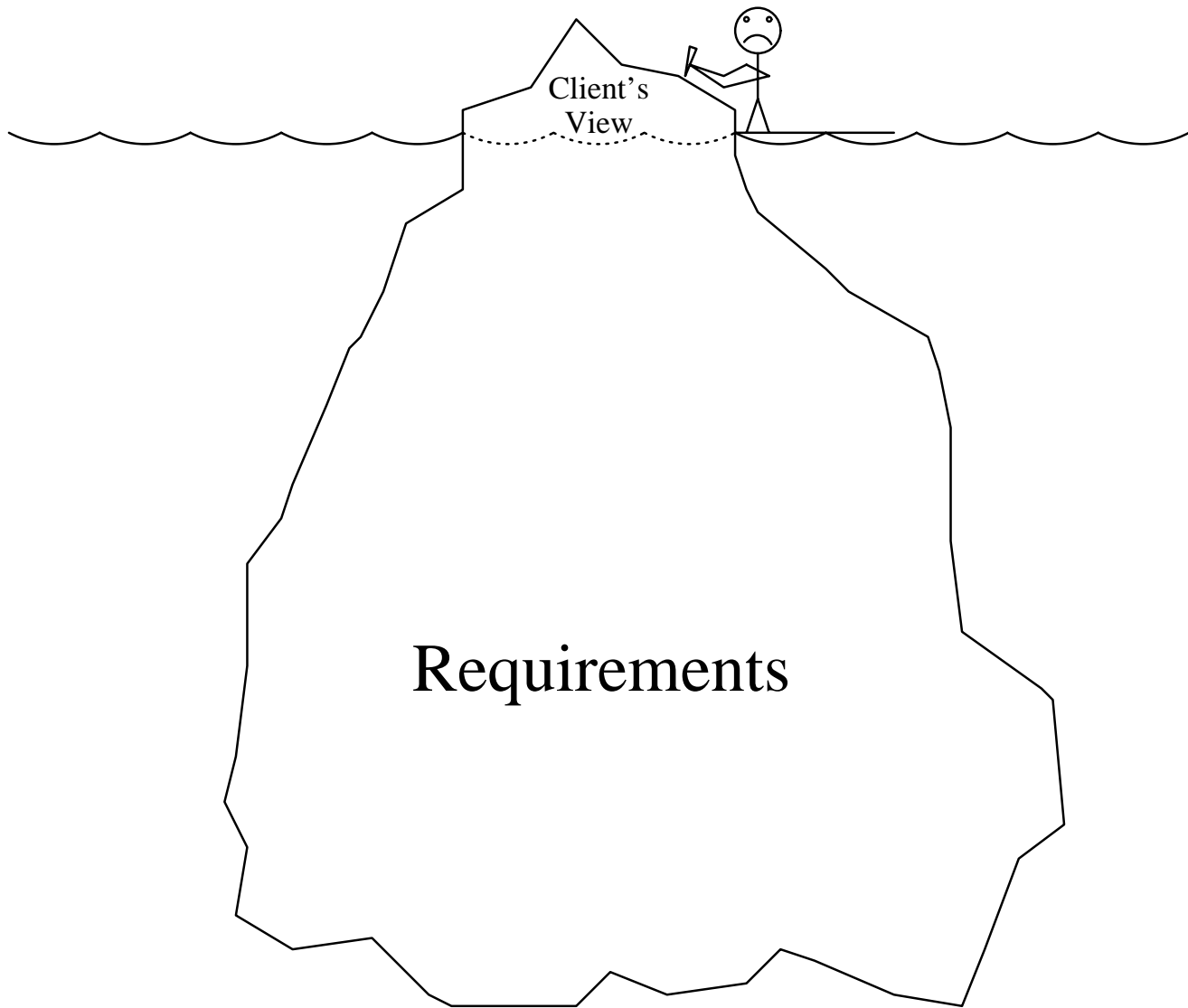


The Requirements Iceberg and Various Icepicks Chipping at It

Daniel M. Berry



The Boring Title

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

Outline

Lifecycle Models

RE is Hard

Why Important to Do RE Early

Myths and Realities

Where Do Requirements Come From?

Formal Methods Needed?

Requirements and Other Engineering

Bottom Line

RE Lifecycle

Outline, Cont'd

Overview of Research

Earlier and Later

Elicitation

Analysis

Natural Language Processing

Tools

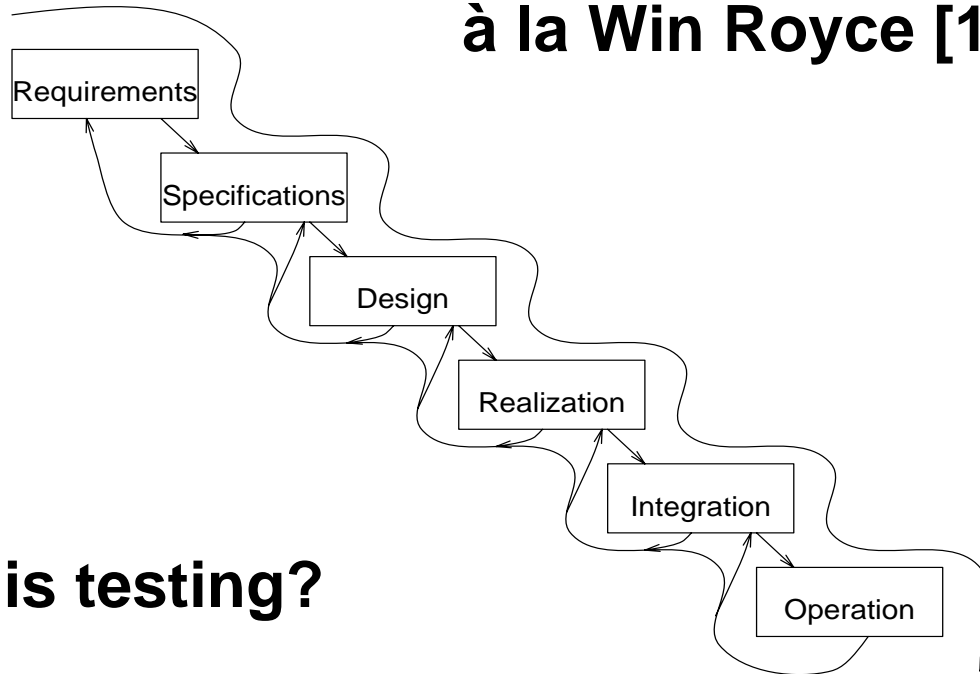
Changes

Empirical Studies

Future

Traditional Waterfall Lifecycle

à la Win Royce [1970]



Where is testing?

Only one slight problem: It does not work!

Problems with Waterfall Model

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



Problems with Waterfall, Cont'd

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 [1995a] 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.**

ICSE '95 was in Seattle, Washington!

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**

NSB, Cont'd

- **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**

NSB, Cont'd

- **However, the essence has resisted attack!**

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

and we produce the same kind of poorly designed software that makes the same kind of stupid mistakes

as 40 years ago!

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.”

Real Life

We see similar requirement problems in real-life situations not at all related to software.

Contracts

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

to cover all possible unanticipated situations.

Houses

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

Contractors even *plan* on this; they underbid on the basic plan, expecting to be able to overcharge on the inevitable changes the client thinks of later [Berry 1998].

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.

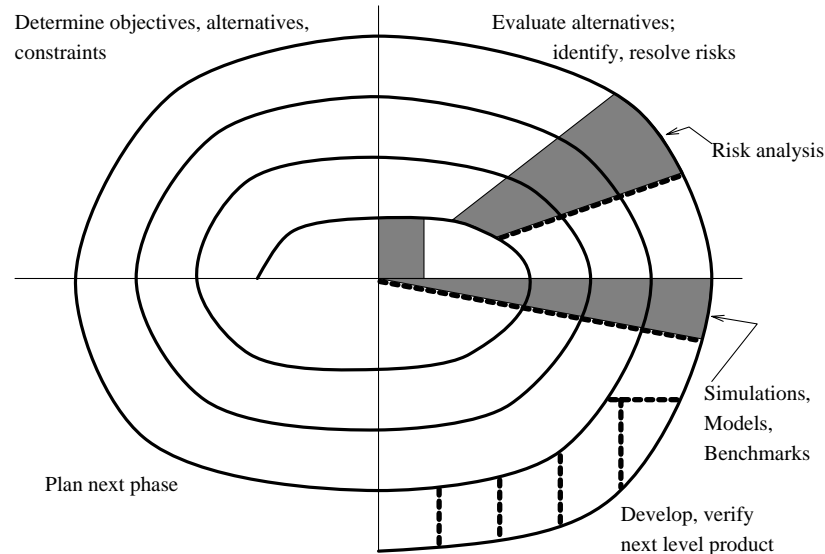
There is a continual stream of updates to the assignment.

SE Lifecycle and Reqs Changes

Thus, the SE lifecycle must be prepared to deal with ever-changing requirements.

More Realistic Lifecycle Model

Spiral Model à la Barry Boehm [1988b]



One may even follow the waterfall in each 360° sweep of the spiral.

Spiral Model

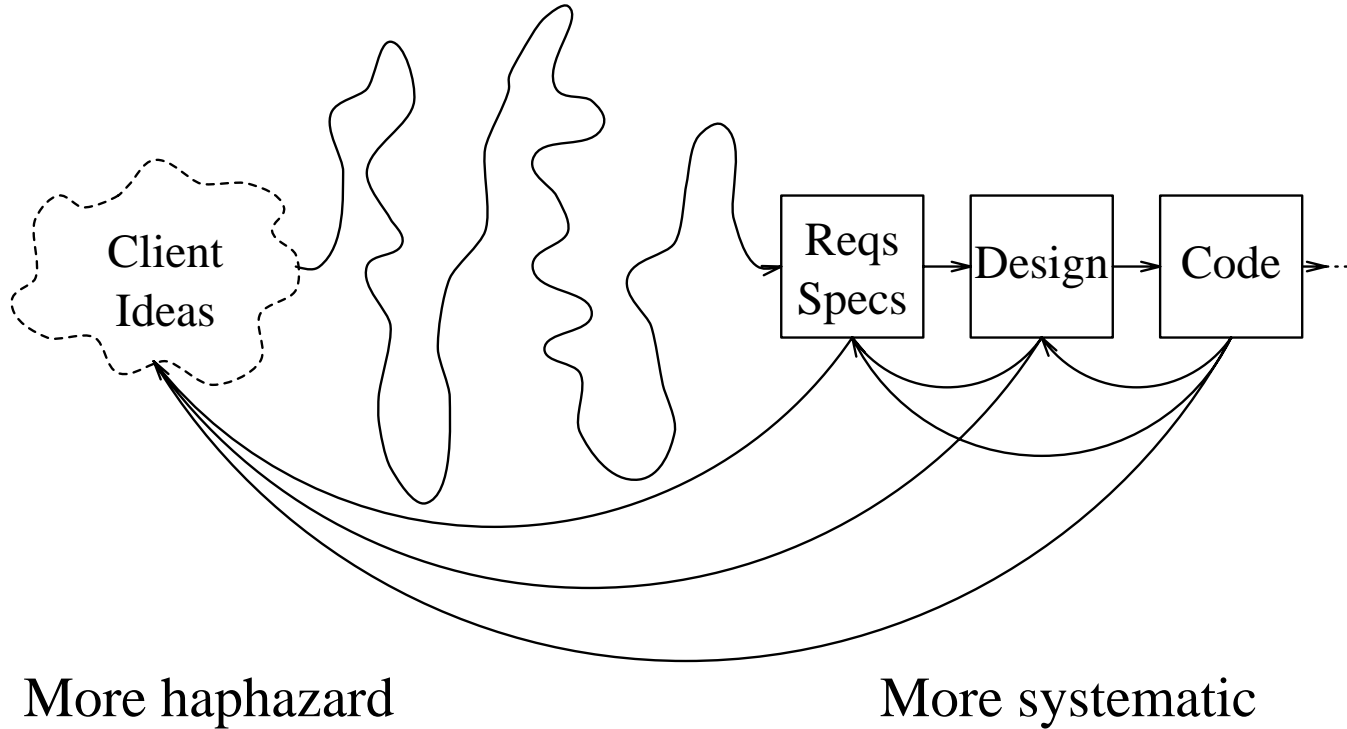
That is, the requirements and the implementation are developed incrementally.

That requirements are changing is planned.

But still, where is testing?

REAL Lifecycle for One Sweep

More difficult than thought to be



Requirements Engineering

That wavy line between Client Ideas and Requirements Specifications is RE.

Stakeholders

A stakeholder is a person who is directly or indirectly affected by the system under construction, .e.g.,

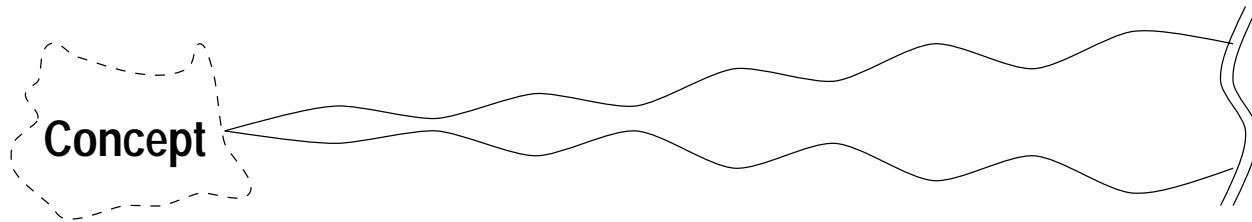
- **customers and users**
- **marketing and sales personnel**
- **developers and testers**
- **maintainers**
- **managers**

RE is Hard

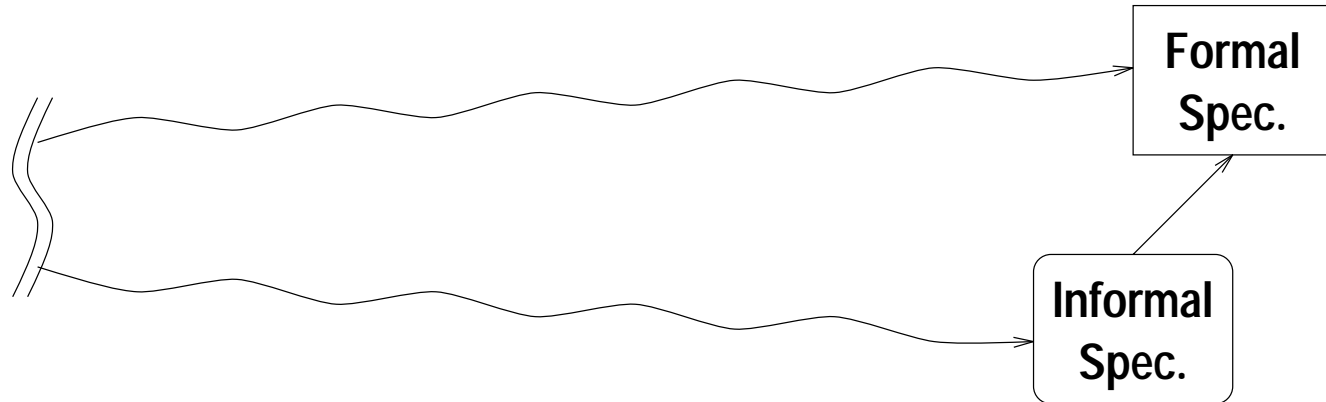
How hard?

Like fighting a platoon of angry, slightly inebriated Klingons.

Distance: Concept \rightarrow Specs



**Folded in middle to give feeling of true
conceptual distances involved**



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

Ken Jackson in a 2003 Tutorial on Requirements Management and Modeling with UML2, cites data from a year 2000 survey of 500 major projects' maintenance costs concluding that 70–85% of total project costs are rework due to requirements errors and new requirements.

In the table, the *d lines include requirements issues and add to 84%, but not all their instances are requirements related.

Flip Side

Those data say that we are doing a pretty good job of implementing of what we *think* we want.

But, we are doing a lousy job of knowing what we want.

Source of Errors

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.**

Even NASA Doesn't Always Fix

Robyn Lutz and Inés Carmen Mikulski [2003] discuss how NASA sometimes discovers requirements errors and new requirements during system operations.

If a mission is already in progress, sometimes the response to this discovery is to change the procedures for the human operators at mission control rather than to try to modify the embedded system on board a space craft.

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!***

Tacit Assumptions, Cont'd

Those tacit assumptions of the problem are reasonable, right?

Unfortunately, the source of most disasters, such as at nuclear power plants, is perfectly reasonable, possibly explicit but usually tacit, assumptions that did not hold in some special circumstances that nobody thought about.

More Timely Tacit Assumptions

Once in Tel Aviv, I read a Hebrew no-parking sign saying “8:00–17:00” as saying that there is no parking from 5:00pm until 8:00am the next morning to reserve parking places for the people who live on the street, who would have special exemption certificates.

While numerals are read from left to right, the flow of the sentence is from right to left and the “–” is *not* part of each numeral.

Timely Tacit Assumptions, Cont'd

According to the city of Tel Aviv, the sign means that there is no parking from 8:00am until 5:00pm to keep the street traversable during the business day.

According to Hebrew reading rules, I am correct and the city is wrong.

I could not get the city to see their error and cancel the ticket!

Requirements Always Change

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]

An E-type system 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, Cont'd

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!**

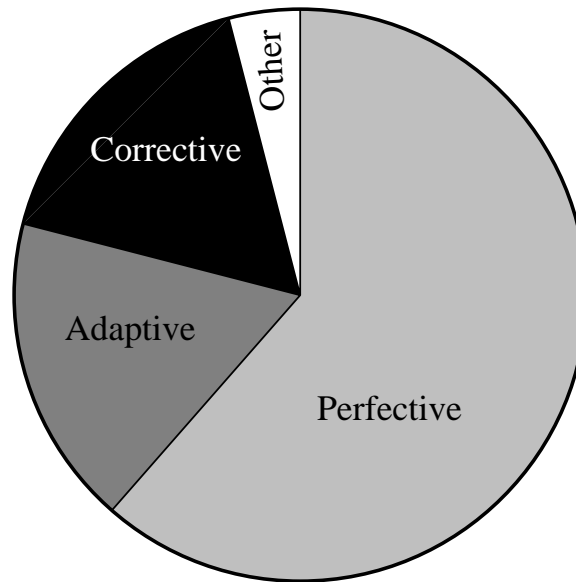
E-Type Software, Cont'd

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, Cont'd

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



Why Important to Do RE Early

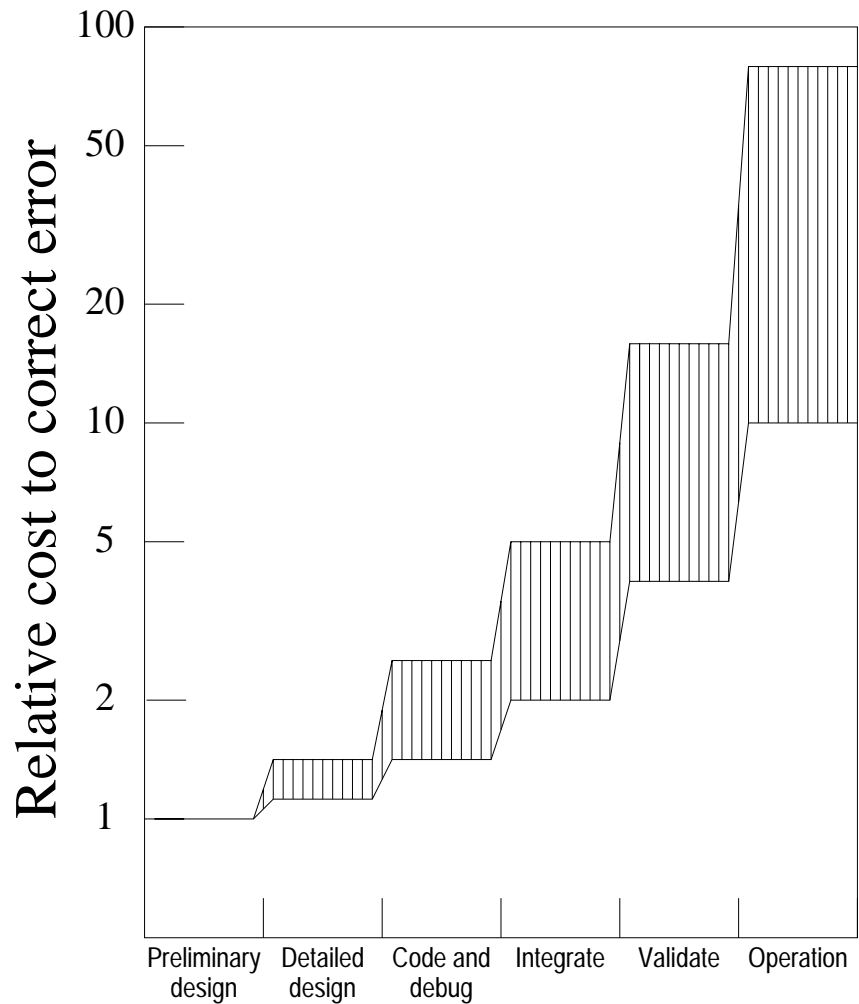
The BIG Question:

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

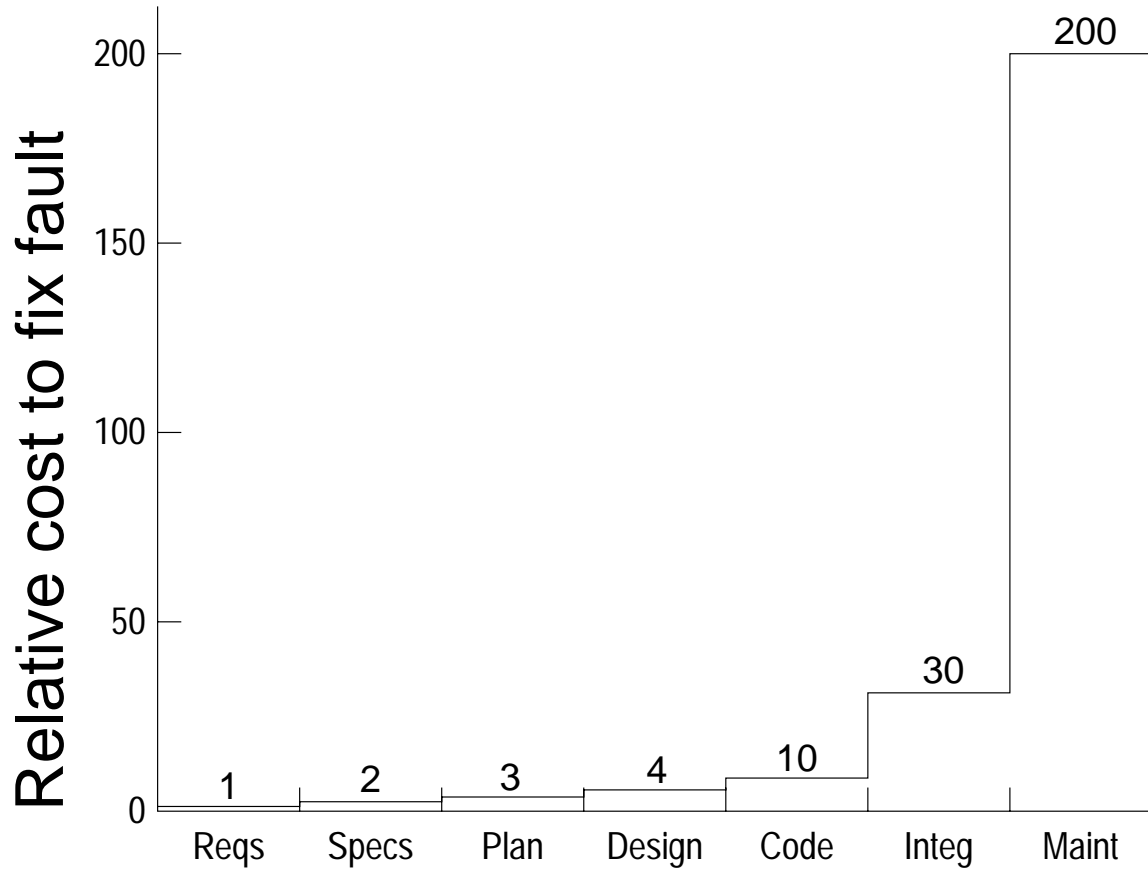
We know that it is much cheaper to fix an error at requirements time than any time later in the lifecycle.

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 at RE time.



Phase in which error is detected



Phase in which fault is detected and fixed

Conclusion...

Therefore, it pays to find errors during RE.

Also, it pays to spend a *lot* of time getting the requirements specification error-free, to avoid later high-cost error repair, and to speed up implementation—even 90% of the lifecycle!

RE & Project Costs

The next slides show the benefits of spending a significant percentage of development costs on studying the requirements.

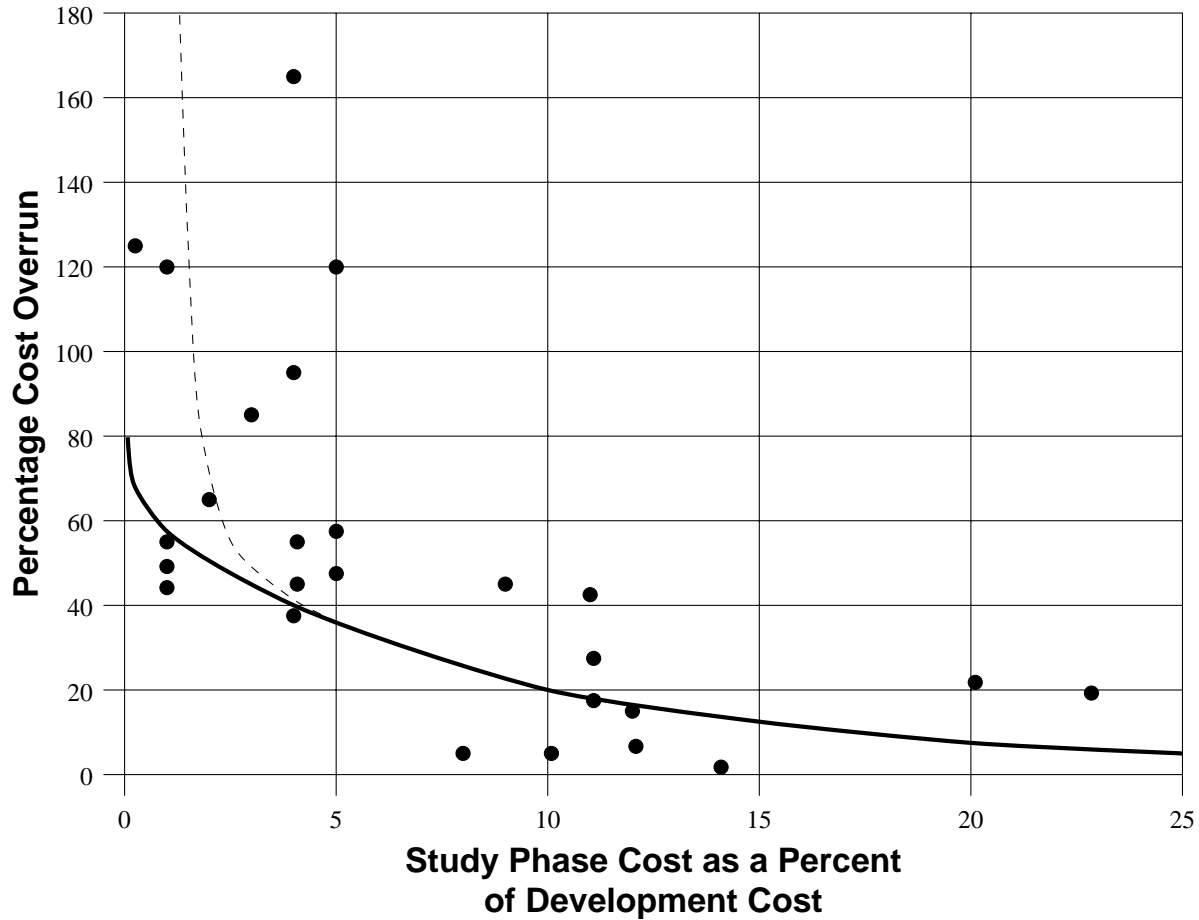
They contain a graph by Kevin Forsberg and Harold Mooz [1997] relating percentage cost overrun to study phase cost as a percentage of development cost in 25 NASA projects.

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**

Project Costs, Cont'd



Project Costs, Cont'd

There are three interpretations of the data:

The more you study the problem, ...

- 1. the lower the costs,**
- 2. the fewer the surprises that cause debugging and rework, and,**
- 3. the more accurate the cost estimates are.**

It's probably a mixture of these.

A Case Study of Serious RE

A Master's student of mine, Lihua Ou, did a case study of writing requirements specification in the form of a user's manual [Berry *et al* (Ou) 2004].

It was *very* successful in that I got a piece of software that I wanted, it was implemented well, it does what I want it to do, and there is a well-written manual that describes the software's behavior completely.

A Case Study, Cont'd

Along the way, it ended up being also a case study in just having a serious requirements process, in which implementation did not begin, and was in fact *delayed*, until the requirements were completely worked out and specified satisfactorily.

The Software

The software was a WYSIWYG, direct manipulation picture drawing program, WD-PIC, based on the batch picture drawing language PIC, a TROFF preprocessor.

Lihua Ou's assignment was to produce a first production-quality version of WD-PIC as her master's thesis project.

Ou's Professional Background

Prior to coming to graduate school, Ou had built other systems in industrial jobs, mainly in commerce.

She had followed the traditional waterfall model, with its traditional heavy weight SRS.

She had made effective use of libraries to simplify development of applications.

Ou's Input

Ou was to look at all previous prototypes and UMs as specifications.

She was to filter these and scope them to first release of a production quality version of WD-PIC running on Sun UNIX systems.

Ou's Assignment

Ou was to write a specification of WD-PIC in the form of a UM.

This UM was

- 1. to describe all features as desired by the customer, and**
- 2. to be accepted as complete by the customer,**

before beginning design or implementation.

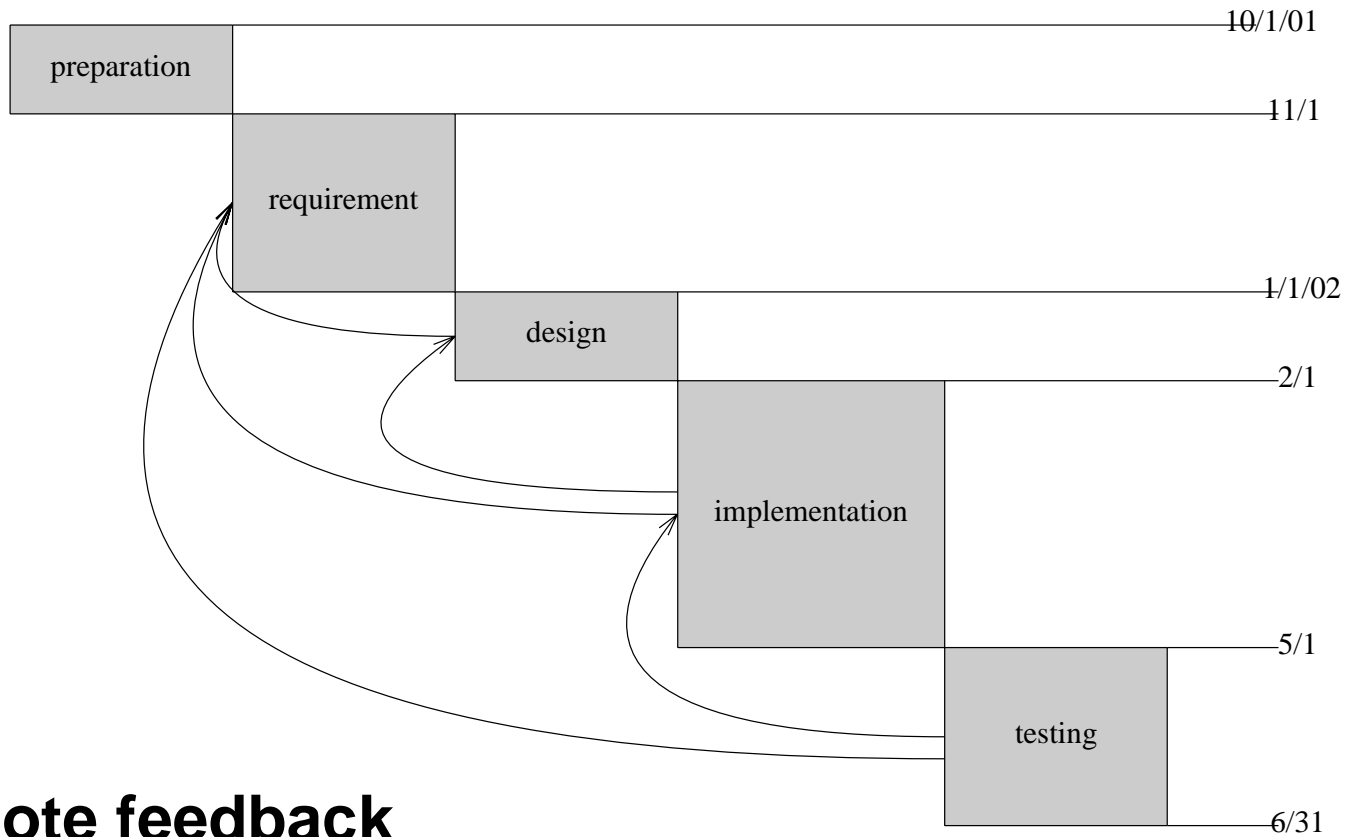
Ou's Assignment, Cont'd

Once implementation started, whenever new requirements were discovered, the UM had to be modified to capture new requirements.

In the end, the UM was to describe the program as delivered.

Project Plan

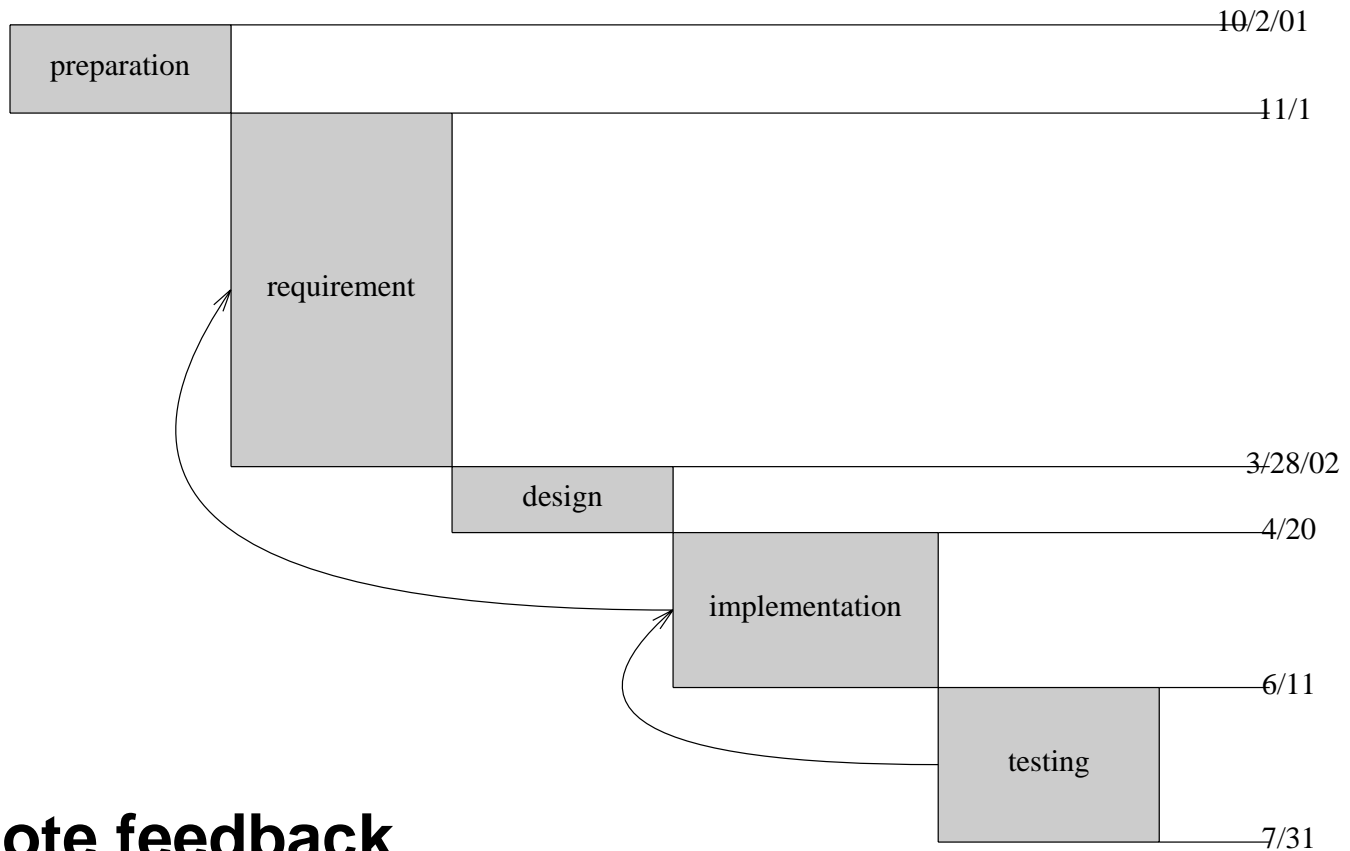
Duration in months	Step
1	Preparation
2	Requirements specification
4	Implementation
2	Testing
1	Buffer (probably more implementation and testing)
10	Total planned



Note feedback

Actual Schedule

Duration in months	Step
1	Preparation
4.9	Writing of user's manual = reqs spec, 11 versions
.7	Design including planning for maximum reuse of PIC code and JAVA library
1.7	Implementation including module testing and 3 manual revisions
1.7	Integration testing including 1 manual revision and implementation changes
10	Total actual



Note feedback

What Happened?

While detailed plan was not followed, total project time was as planned.

Also, Ou produced two implementations for the price of one, for:

- **(planned) Sun with UNIX and**
- **(unplanned) PC with Windows 2000**

Surprise

Ou was more surprised than Berry that she finished on time.

Berry had a lot of faith in the power of good RE to reduce implementation effort.

Adding to Ou's surprise was that the requirements phase took nearly 5 months instead of 2 months; the schedule had slipped 3 months out of 10, what appeared to be way beyond recovery.

Then and ...

Ou's long projected implementation and testing times and the 1 month buffer indicate that she expected implementation to be slowed by discovery of new requirements that necessitate major rewriting and restructuring.

Then and Now

This time, only minor rewriting and no restructuring.

Thus instead of 2 months specifying and 7 months implementing and testing,

she spent 5 months specifying and only 4 months implementing and testing.

Why?

By spending 3 additional months writing a specification that satisfied a particularly hard-nosed customer who insisted that the manual convince him that the product already existed,

Our produced a specification that

- **had very few errors and**
- **that was very straightforwardly implemented.**

The Errors

Almost all errors found by testing were relatively minor, easy-to-fix implementation errors.

The two requirement errors were relatively low level and detailed.

They involved subfeatures in a way that required only very local changes to both the UM and the code.

What Helped?

All exceptional and variant cases had been worked out and described in the UM.

Thus, very little of the traditional

- **implementation-time fleshing out of exceptional and variant cases and**
- **implementation-time subconscious RE.**

Test Cases

The manual's scenarios, including exceptions and variants turned out to be a complete set of black box test cases.

Tests were so effective that, to our surprise, ... scenarios not described in the UM, but which were logical extensions and combinations of those of the UM worked the first time!

The features composed orthogonally without a hitch!

Satisfied Customer

Berry found Ou's implementation to be production quality and is happily using it in his own work.

We Don't Have Time for RE

“I know that it is important to get requirements, but we don't have time for it; we have to get to coding to meet our deadline!”

We Don't Have Time, Cont'd

I will prove that we, in fact, always do write requirements during the normal commercial software lifecycle.

Therefore, since we always do it, we must have had enough time!

Never Needed RE Before

“We’ve never written a requirements document and we’re still successful!”

So says the manager of a project that delivered tested software with a user’s manual or an on-line help.

So says the manager justifying a decision to plunge into development without first determining and specifying requirements.

Sorry to Disappoint You!

**Kamsties, Hörmann, and Schlich [1998]
observe:**

Any project that does testing has to determine the requirements in order to determine covering test cases and their expected outputs. The test plan ends up being a requirements specification.

Sorry, Cont'd

Any project that produces user documentation has to determine the requirements in order that the documentation describe all of the features well. The documentation ends up being a requirements specification.

Even Microsoft does both.

So there is no escaping determination and specification of requirements (unless you don't do testing and user documentation!).

Sorry, Cont'd

There is no avoiding the time required to determine and specify the requirements.

However, if you produce requirements only as a side effect of testing and documentation, you lose the key benefit of *early* requirements determination and specification, namely finding errors at the least cost.