Writing code during requirements determination: A good head start or a costly bad bet?

Daniel M. Berry
dberry@uwaterloo.ca

At the start of many SW developments:

I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING !
SHELTON

Boss's Fateful Order

"You people start the coding while I go find out what the customer wants."

Hereinafter, called "the boss's order"

How many of you have participated in such a boss's order?

Have Participated

You have participated in such a boss’s order!

I am not surprised.

It’s a very common occurrence, arising from the mistaken assumption that the sooner coding starts, the sooner it ends.

Did it work as planned — giving your team a head start on the coding?

If not, …

If not, then you will learn why!

If so, …

If so, then:

Did your team have to change *any* of the already-written code after the boss learned what the customer really wanted?

If not, …

If not, then you were *very, very, very* lucky,

and you will see why.

We’ll all see what normally happens.

If so, …

If so, what percentage of the already-written code had to be changed in the end, including all ripple effects?

10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%, 110%?

Thanks, we’ll use these data as we answer the title question!

Implicit in Boss’s Order

First, there are some assumptions implicit in the boss’s order.

First Assumption

1. Requirements are easy to learn.

How many of you have found this to be true?

No

The fact that this is *not* true in most people's experience is probably the reason they prefer to skip this step and move right to coding.

Yes

Your software was a toy, not operating in the real world, perhaps solving a purely mathematical problem.

Second Assumption

2. The customer knows what E wants!

How many of you have found this to be true?

No

A very common complaint among programmers is that the idiot customers just do not know what they want and are *always* changing their minds!

Yes

Wow! you were *very, very, very* lucky.

Perhaps, your customer was a professional requirements engineer, who had done er homework.

Third Assumption

3. One can write code before knowing knowing its requirements, i.e., what it is supposed to do.

Since writing any line of code requires knowing what the line of code is supposed to do, how can you write SW without knowing its requirements?

Ah, but …

A programmer who knows the SW's domain can make very educated guesses about what most of the SW is going to have to do.

And while the code will not be 100% correct, it won't be too far off.

In any case, we can fix the imperfections later.

But we'll see …

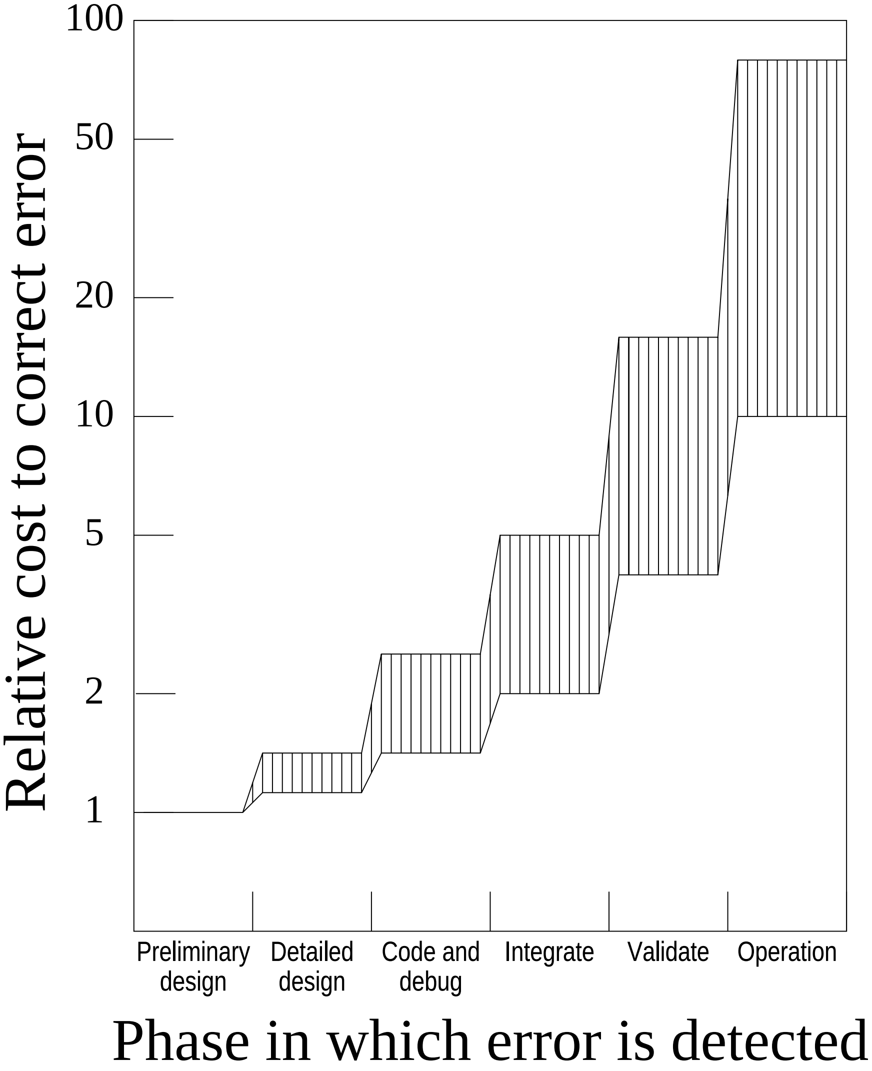
We'll see later how much that imperfection will cost.

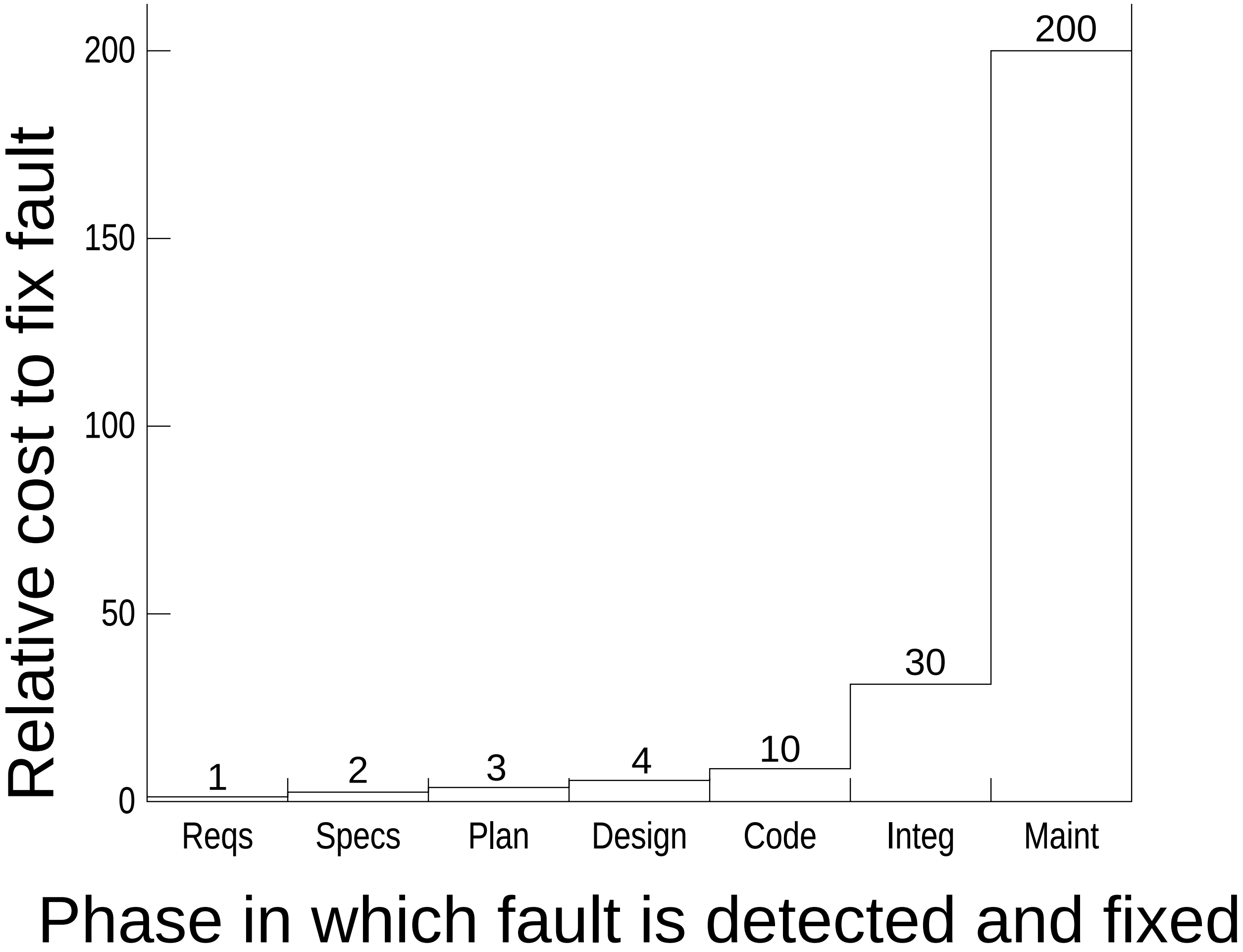
Some Facts

Here are some facts that we need to analyze if writing code during requirements determination is a good head start or a costly bad bet:

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 requirements engineering (RE) time.


100
50
20
10
5
2
1
Relative cost to correct error
Preliminary design
Detailed design
Code and debug
Integrate
Validate
Operation
Phase in which error is detected

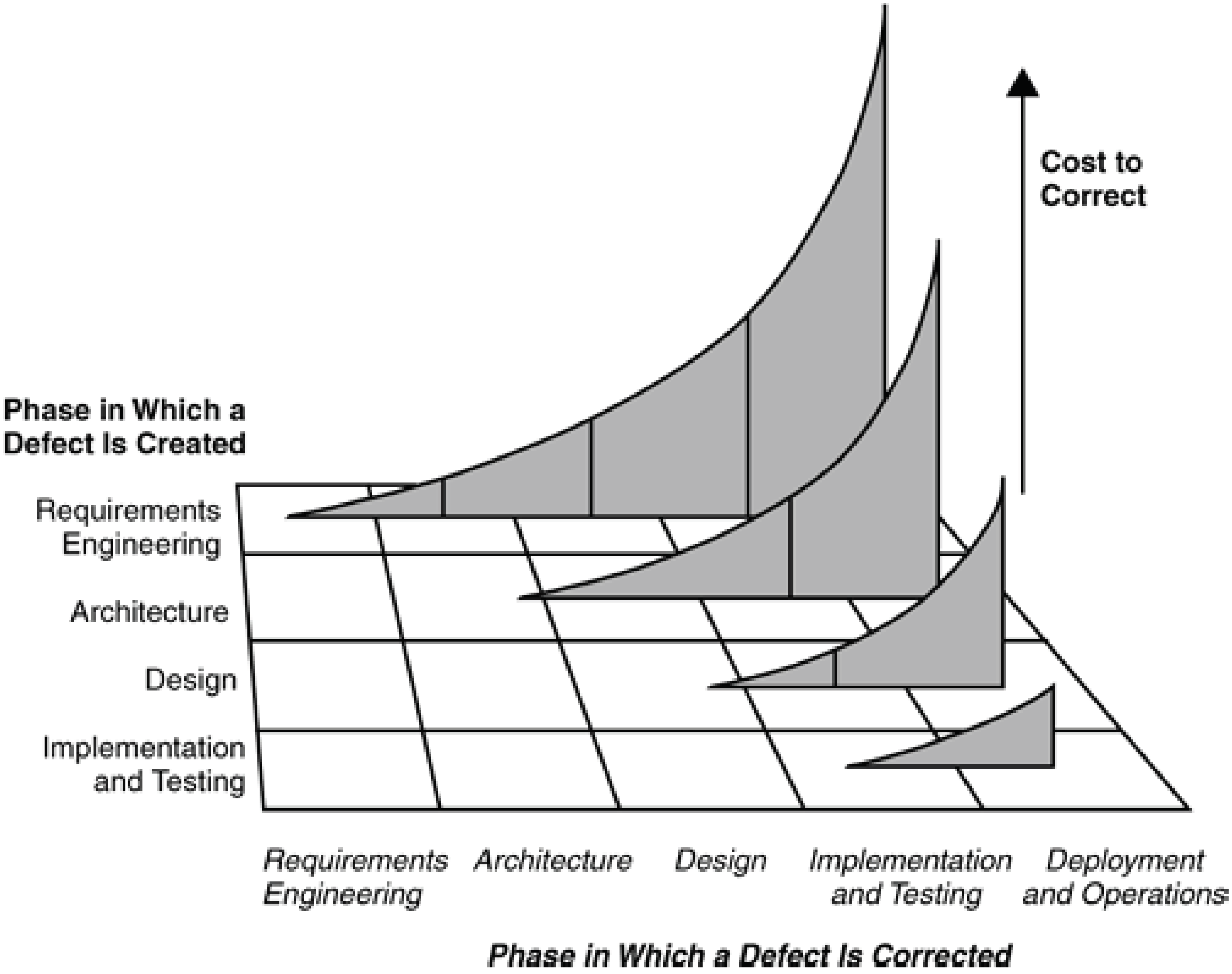


Relative cost to fix fault
200
150
100
50
0
1
2
3
4
10
30
200
Reqs
Specs
Plan
Design
Code
Integ
Maint
Phase in which fault is detected and fixed

Cost to Fix Errors, Cont'd

More specifically,

- **requirement defects are harder to fix than architectural defects,**
- **which are harder to fix than design defects,**
- **which are harder to fix than implementation defects [Allen et al 2008].**


Cost to Correct
Phase in Which a Defect Is Created
Requirements Engineering
Architecture
Design
Implementation and Testing
Requirements Engineering
Architecture
Design
Implementation and Testing
Deployment and Operations
Phase in Which a Defect Is Corrected

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 70% of the lifecycle!

The 70% is not a prescription, but a prediction of what will happen.

Back to the Boss’s Order

If as little as 10% of the code written in advance of knowing the full requirements has to be changed after the full requirements are known, …

the cost of writing the code has doubled:

Bad Bet

If C is the cost of writing the advance version, the cost of fixing the advance version when as little as 10% of it has to be changed is $(10 \times 0.1 \times C)$, and the total cost of writing the code is

$$C + (10 \times 0.1 \times C) = 2 \times C$$

Oy!

And it gets worse if more than 10% has to be changed.

It Can Get Much Worse!

Data show that 50–85% of all lifetime defects in deployed SW can be traced back to requirement errors:

> **missing,**
> **wrong, and**
> **extraneous**

requirements

Better Bet

So what’s a better use of the programmers who would become idle if they are not put to work starting the coding while the boss goes to find out what the customer wants?

Better Bet

So what’s a better use of the programmers?

Have them join the RE team

- **to provide more brain power to the RE effort and**
- **to help the RE team know when the requirements specification is complete enough that it can be programmed without the programmers’ having to ask questions.**

So, obeying the boss’s order amounts to a *very* bad bet!

It’s practically guaranteed to end up at least doubling the cost of writing the code and developing the system.

Other Implication

This cost says something about how bug fixes and maintenance should be done.

Start All Over

It’s cheaper in the long run to throw out the buggy code, redo and finish requirements analysis, and start coding from scratch.

But, no one is willing to throw out er investment in written code, even if it’s clearly buggy, …

***even though* the data are clear!**

Nature of Most Bugs

About 70% of the bugs arise from missing easily identified requirements that deal with exceptions that are inherent in the basic functions of the SW, e.g.,

> **ill-formed input, and**
> **denominator of 0,**

and *not* creative new requirements that no one thought of.

Nature of Most Bugs, Cont’d

So, a little bit of careful thinking about exceptions, …

enough to track them all down, …

pays off BIG!