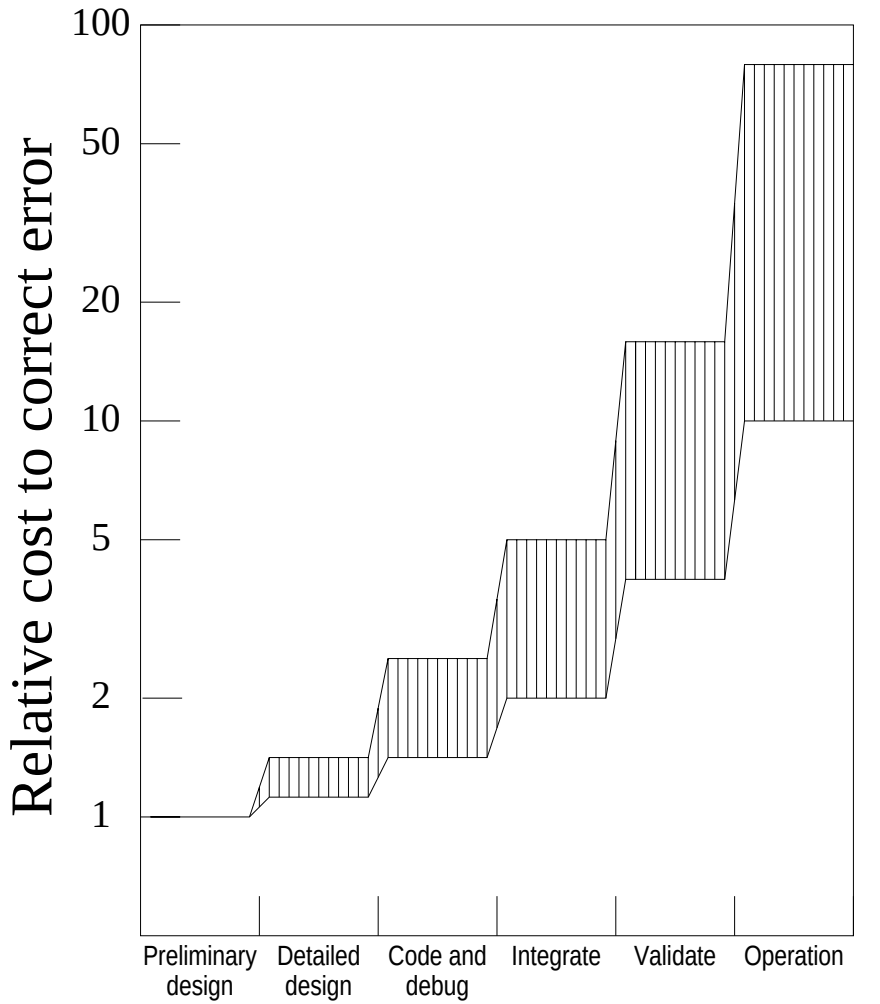
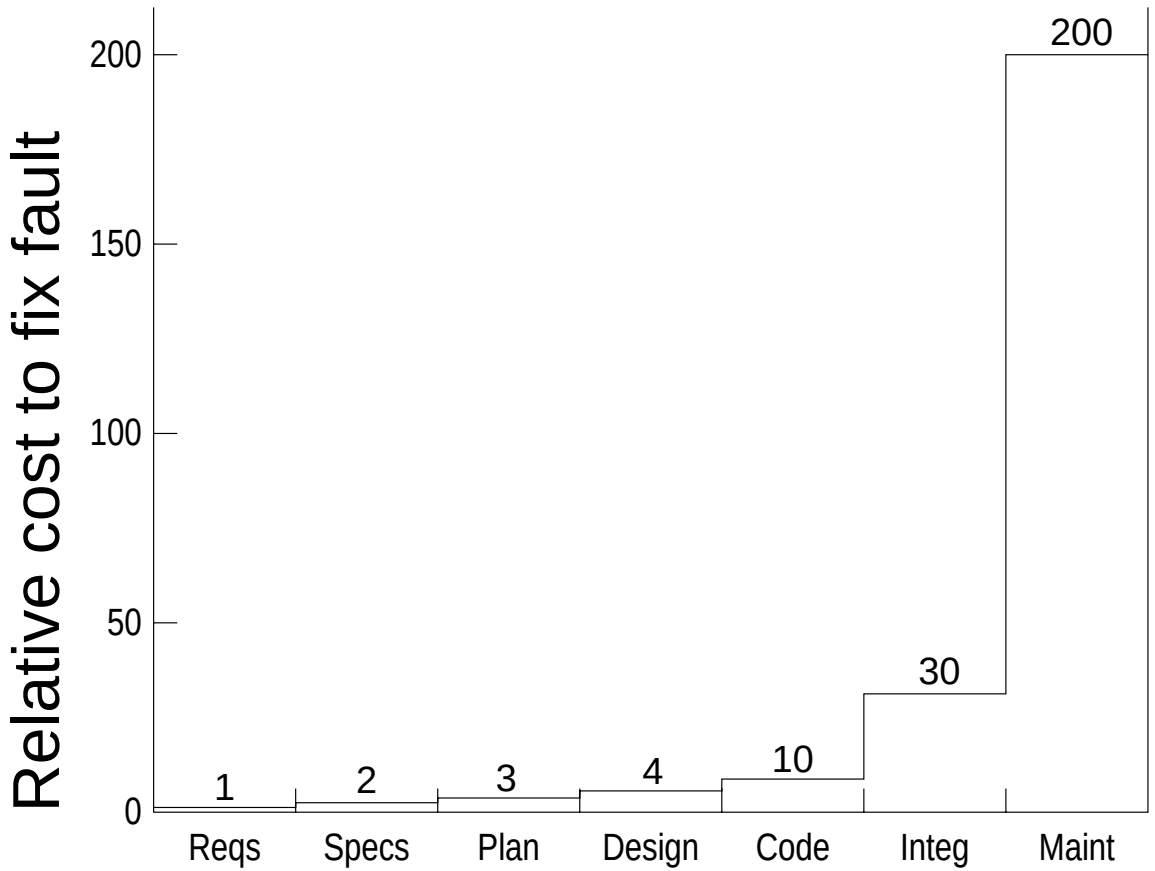
Cost to Fix Errors

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.


Relative cost to correct error
100
50
20
10
5
2
1
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

Source	**Phase Requirements Issue Found**			
	Require-ments	Design	Code	Test
[Boehn, 1981]	1	5	10	50
[Hoffman, 2001]	1	3	5	37
[Cigital, 2003]	1	3	7	51
[Rothman, 2000]		5	33	75
[Rothman, 2000] Case B			10	40
[Rothman, 2000] Case C			10	40
[Rothman, 2002]	1	20	45	250
[Pavlina, 2003]	1	10	100	1000
[McGibbon, 2003]		5		50
Mean	**1**	**7.3**	**25.6**	**177**
Median	**1**	**5**	**10**	**50.5**

[NASA, 2010 System Cost Factors]				
Method 1	1	8	16	21
Method 2	1	3–4	13–16	61–78
Method 3	1	4	7	157–186

[Langenfeld, 2016]	1	1.6	4.9	6.7
[Hamill, 2017] Mean	1		5.1	24
[Hamill, 2017] Median	1		3	27

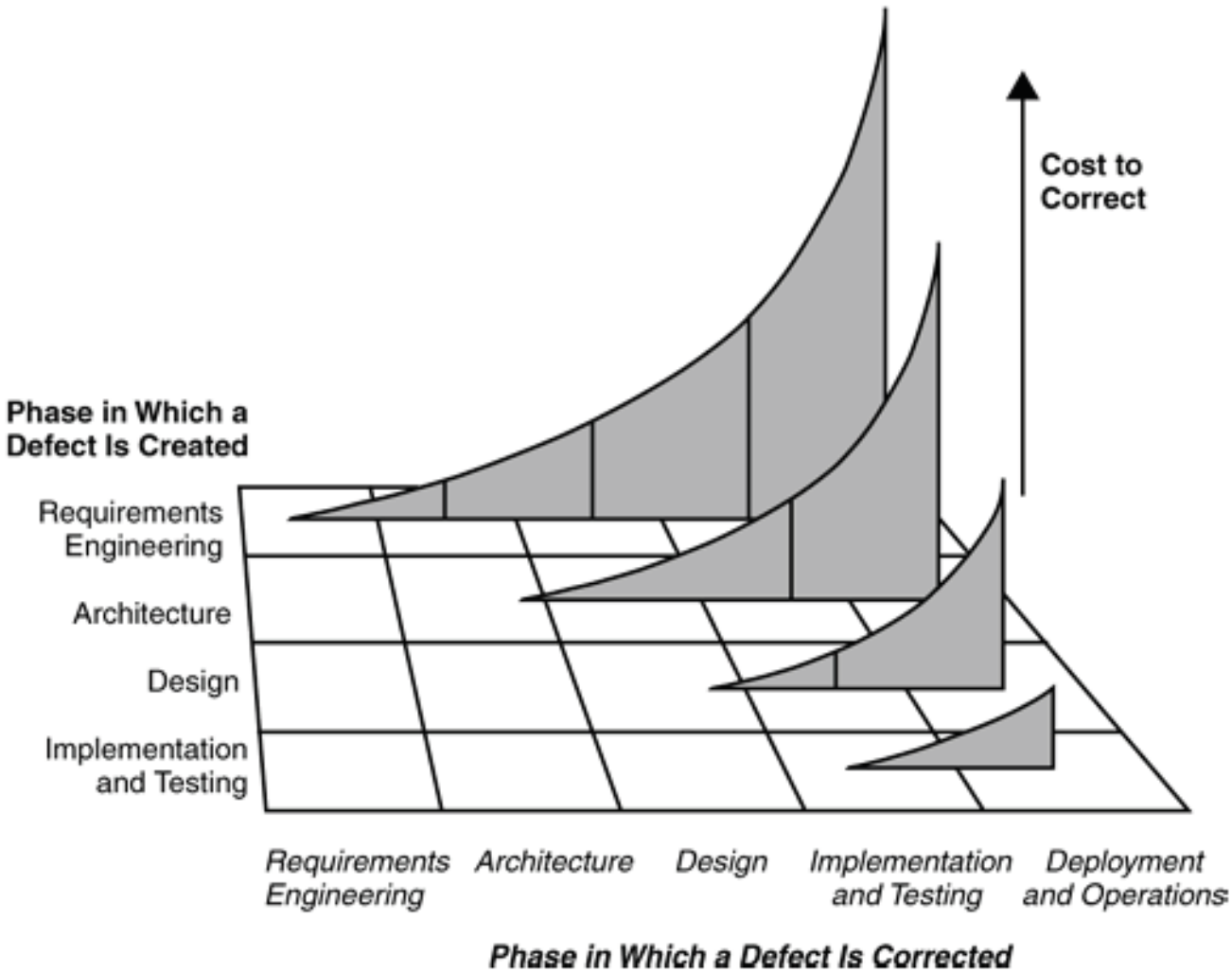
[IBM SSI, Pressman]		1	6.5	15
[Extrapolated & Normalized IBM SSI, Pressman]	1	5	32.5	75

Normalized Cost-to-Fix Estimates

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.

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:

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!**