G and D Requirements Overview

Daniel M. Berry
Márcia Lucena
Victoria Sakhnini

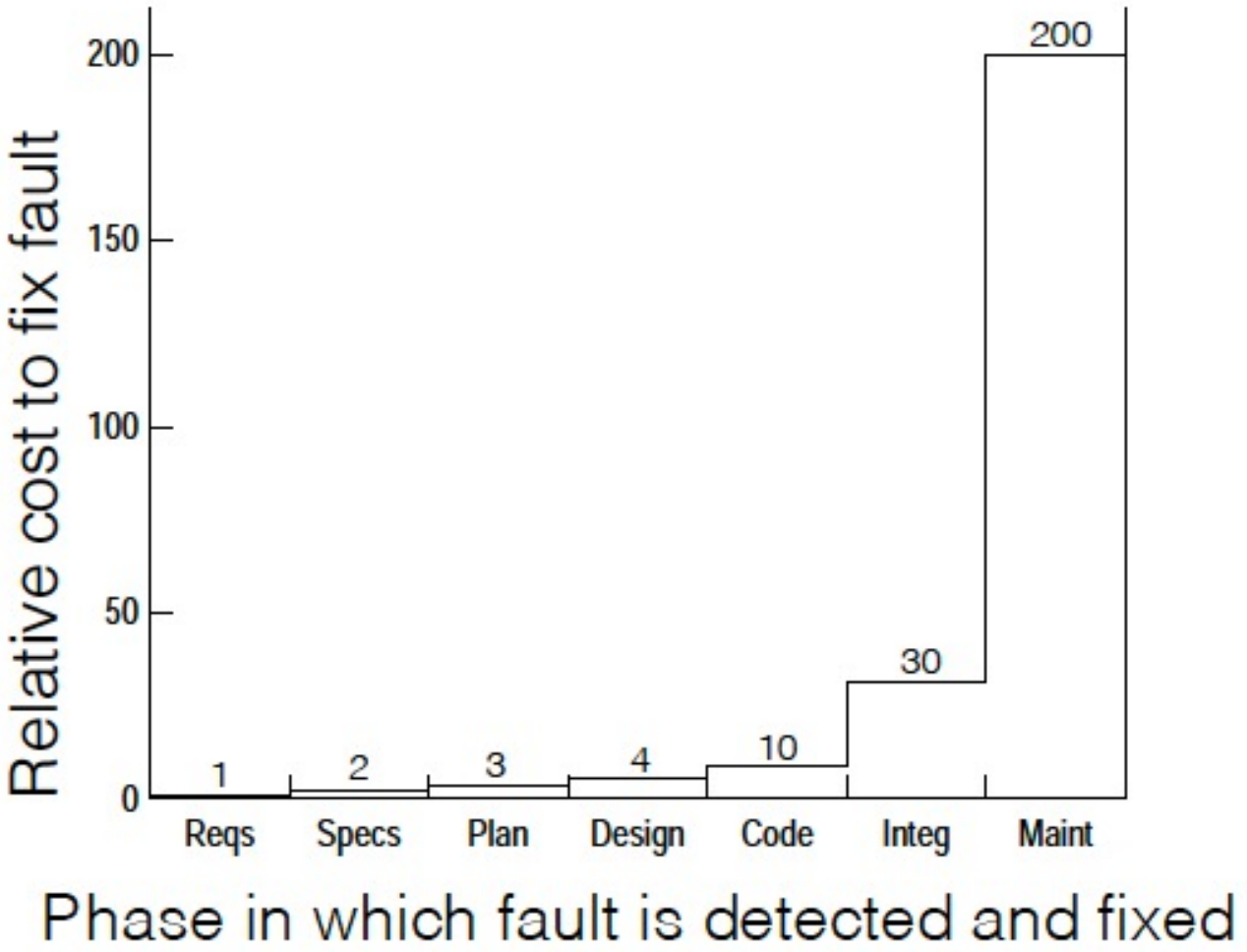
Realities About Software Development Projects

- Everyone says
- “We *know* that we should work out all the requirements before we start to code,
- but we *don’t* have time!
- We gotta get started coding; otherwise we will not finish in time!”

Wrong!

The problem is that if you start coding before you work out all the requirements, then the cost of correcting the code when a missing requirement is finally discovered is *10-200 times* — depending on when the defect is found — the cost of writing the code with that requirement already specified.

The Data Show



Start Coding Later, Save Time!

A: Starting coding before all the requirements are worked out and specified completely means that you finish coding much later than if you had delayed the starting of the coding until after all the requirements are worked out and specified completely!

Start Coding Later, Save Time!

A: In other words:

- start coding later, finish earlier
- start coding earlier, finish later

This truth goes against every manager's guts, so *no on*e delays coding until after the requirements are completely specified (even though the data are clear!).

But But But…

B: But but but.. requirements keep coming with no end in sight. Users think of new requirements *all the time*. So what difference does it make? We're going to have to deal with new requirements after the coding is done anyway?

That's absolutely right. In fact, both A and B are right! So, now what?

Two Different Kinds of Requirements

You see, each of A and B is talking about a different set of requirements!

- *Scope DeterminiG Requirements (G requirements)* that keep coming and phenomenon B
- *Scope DetermineD Requirements (D requirements)* that are expensive to fix and phenomenon A

Pocket Calculator Example

- Pocket calculator: with +, -, *, and /
- G requirements: +, -, *, /, and **
- D requirement: NZD: “in /, the denominator cannot be 0”

If You Start Coding Too Soon

So if you start coding the G requirement /, and you are not aware of its D requirement, NZD, you will write code that will break if ever / is presented with a 0 denominator.

At that point fixing the code will cost 10-200 times what it would have cost to have just specified NZD upfront so that coding takes it into account from the beginning.

The G Requirements *Are* Different

- Yes, if you now add a new G requirement, particularly one that is not anticipated, there is a chance that it will clash with the existing architecture, and you'll have to do an expensive restructuring.
- But that's unavoidable. And that's the sort of thing iterative and agile methods are designed to deal with.

The G Requirements *Are* Different

- And, if you have to restructure, it will cost 10-200 times more than it would have cost if you had included the G requirement from the beginning.
- There is evidence that throwing out the code and starting all over with all the requirements is much cheaper.
- But no manager's guts permits doing that!

Inescapable Fact Affecting D Requirements

The basic fact is that there is *no* way that you can write any code without knowing what its requirements are, i.e., what it is supposed to do, *even* if you have to decide what the requirements are **as you are coding**.

It's inevitable, like death and taxes.

So the nature of D requirements is:

Once you have picked a scope for your next sprint or iteration, i.e., a particular set of G requirements, the D requirements associated with the chosen G requirements are there *even if you have not written them down*.

The Nature of D Requirements:

If you start coding with them missing from the specification, and you discover their existence during coding, you will have to specify the missing D requirements before you can finish the coding, at 10 times the cost of having determined them before coding.

So the nature of D requirements is:

This is a stupidly expensive way to discover and specify D requirements, because they were already apparent when specifying them was much cheaper.

Worse Comes to Worst

If worse comes to worst, and as very typically, you deliver the code *before* a D requirement is discovered, then a *user* — the best defect finder in the universe — will eventually discover it, …

and it will cost 100 times more to fix it than having written it down up front.

More Detailed Example

In a system for processing payments and taxes from and for a national insurance plan (e.g., social security in the US),

- any place in the requirements a person's national insurance number (NIN) is used, the requirement will be assuming that each person has a unique NIN.

Exceptions → D Requirements

There are a number of exceptions that have to be guarded against:

- the NIN is not unique
- the NIN is invalid in some way
- (perhaps you can think of more)

Exceptions → D Requirements, Cont'd

- the NIN is not unique:
  - a person has more than one NIN
  - a person has no NIN
  - the NIN is shared by at least two people

Exceptions → D Requirements, Cont'd

- the NIN is invalid in some way:
  - the NIN has never been issued
  - the NIN is not syntactically correct (has a character that cannot be in a number)
  - the NIN fails validity check (check sums are wrong)

Exceptions → D Requirements, Cont'd

For each of these exceptions, *E*, one D requirement is that *E* has not happened.

Examples of G Requirements for this System

- Requirements arising from using the NIN as an income tax payer identification number
- Requirements arising from using the NIN as a vaccination certificate identification number

See how these requirements are independent of the G requirements of the original system.

D and G Requirements Partition Requirements

Every requirement of a system should be either a D or a G requirement, …

Because every requirement is dependent or independent of the requirements that form the scope of the system.

Classifying Defect Tickets for the Development of a System *S*

Each defect ticket should be about either

- a defective implementation of some requirements
- missing requirements, each of which is either a D or G requirement

Classifying Defect Tickets for *S*, Cont'd

Examples of defects from an incorrect implementation of some requirements:

- checking that y != letter O instead of number 0
- used the wrong variable in an expression

Classifying Defect Tickets for *S*, Cont'd

Examples of defects from a missing D requirement

- system crashes when user enters a nonexistent NIN
- system crashes when user enters a short NIN
- user that enters a wrong NIN gets into another person's account

Classifying Defect Tickets for *S*, Cont'd

Examples of defects from a missing G requirement

- system crashes when user tries to request a tax refund from er income tax account
- system crashes when user tries to add a new jab to er vaccination record

Alternative Names for D and G Requirements

D Requirement	G Requirement
Use Case: Variation/Exception	Use Case: New/Independent
Internal	External
Non-E-Type	E-Type
Req Needed to Build the System Right	Req Needed to Build the Right System
Dependent/Implied/Interacting	Independent/Axiom/Orthogonal
Update	New Release
Revision (Vx.Rn → Vx.Rn+1)	New Version (Vx.Rn → Vx+1.R1)
Maintain Consistency	Add New Feature
System Req	Environment/World Req
White Box Req	Black Box Req

Questions?