Scope Determined (D) Versus Scope Determining (G) Requirements: A New Significant Categorization of Functional Requirements

Daniel M. Berry
Márcia Lucena
Victoria Sakhnini
Abhishek Dhakla

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!”

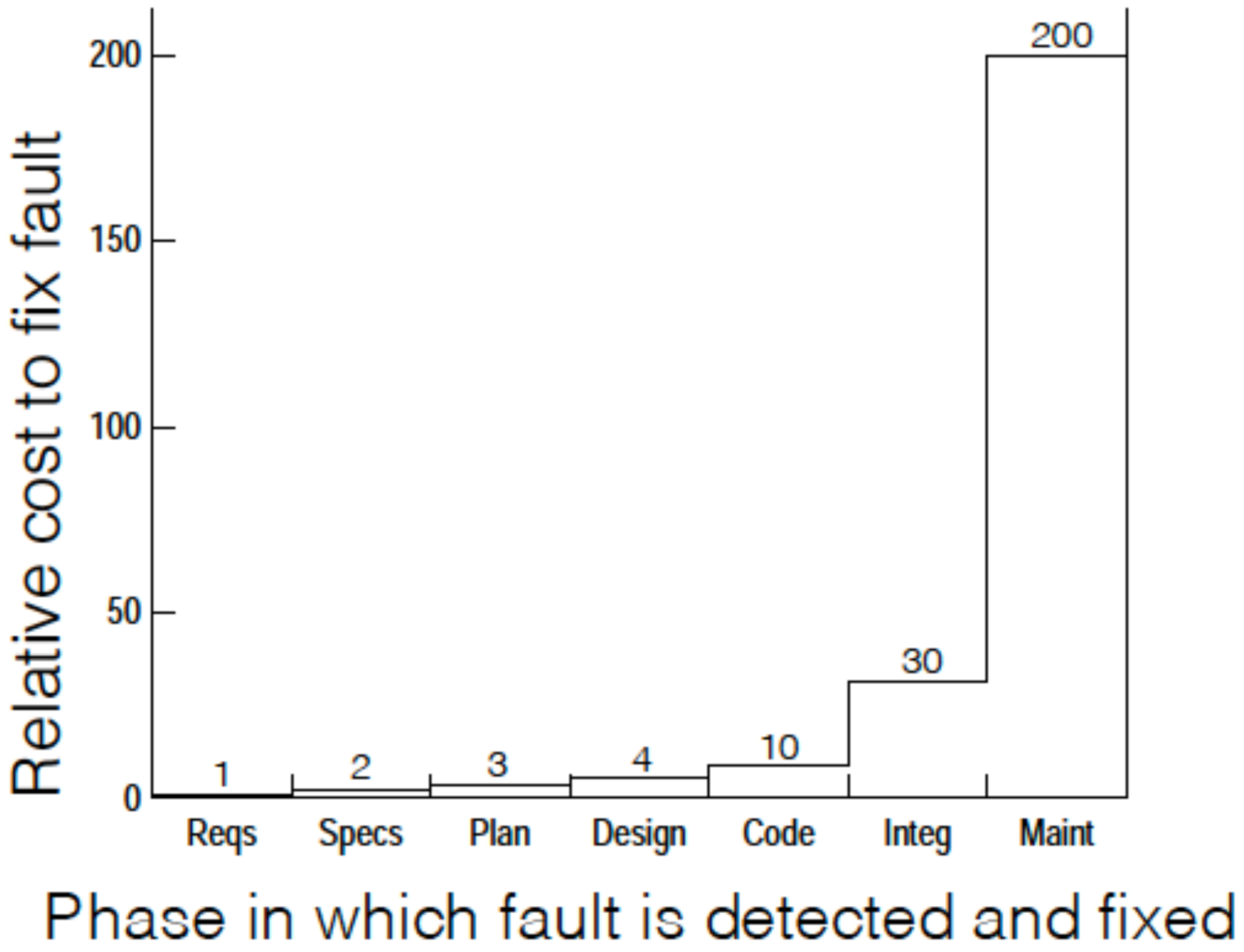
Wrong!

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 defect 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



Phenomenon A: Start Coding Earlier, Finish Later!

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 were worked out and specified completely!

Phenomenon A: Start Coding Later, Finish Earlier!

In other words:

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

This truth goes against every manager's guts;

so *no sane manager* delays coding until after the requirements are completely specified (even though the data are clear!), for fear of losing job if the project with a new-fangled method fails.

Phenomenon B: But But But…

“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!

Phenomena A and B

That’s absolutely right!

In fact, empirical studies go both ways or are inconclusive.

In fact, *both* A *and* B are right! So, now what?

Two Different Kinds of Requirements

You see, the Phenomena A and B are talking about entirely different sets of requirements!

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

Pocket Calculator Example

Pocket calculator (PC): with scope +, -, *, and /
This is the *scope* of the PC.

- G requirements: **, *log*
  Adding them would determine a new scope for the PC.
- D requirement: ...

Pocket Calculator Example

Pocket calculator (PC): with scope +, -, *, and /

- D requirement: NZD: “in /, the denominator cannot be 0”
  Its presence determined by the presence of / in the PC’s scope.

  In a sense, NZD is already in the PC’s scope.

Completion of a Scope

Thus, there is a notion of
the *completion of a scope*
to contain all its D requirements.

If You Start Coding Too Soon

So if you start coding the 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, …

If You Start Coding Too Soon

At that point,

depending on when the discovery is made,

fixing the code will cost 10–200 times what it would have cost to have specified NZD upfront so that coding took it into account from the beginning.

Sometimes, fixing a missing D requirement requires restructuring.

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 w.r.t. the empty scope, the D requirements associated with the chosen scope are there *even if you have not written them down*.

I.e., every requirement in a scope’s completion is there, even if you have not written it 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.

This is major technical debt from postponing full RE!

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 200 times more to fix it than having written it down up front.

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

Research Questions

How relevant is the G vs. D categorization of requirements?

How prevalent are

- missing requirements,
- missing D requirements, and
- missing G requirements

in computer-based system (CBS) development projects?

Preliminary Results: Summary

We reexamined past case studies,

done with no notion of D and G requirements,

with a D vs. G requirements lens:

- Challenged and Failed Projects
- Successful Project

Berry was one of the authors in each.

The studies are described in tech report cited by proceedings short paper.

Challenged and Failed Projects : Summary

In all the challenged and failed projects, it appears that many if not a majority of the defects were …

from missing or wrong D requirements, ...

and not from missing G requirements
and not from implementation defects.

Successful Project : Summary

The one project that focused its RE on finding all D requirements of its scope, letting RE continue until it had found the completion of the project’s scope …

delivered two versions of its software on time, matching its specification, with no known defects.

Preliminary Results from Work in Progress

We are studying defect tickets from a company that has about the same number of Waterfall & Agile projects.

Unfortunately, we have no access to domain experts in the company. So we have to guess about the nature of each defect.

Each author classified separately,

but Fleiss's κ among them is 0.056 ☹.

Preliminary Results, Cont’d

Among all projects, using only tickets classified unanimously:

- 100% of the defects are from missing requirements & none of the defects are from implementation errors,
- of the 100%, 92% are from missing D requirements & 8% are from missing G requirements.

Preliminary Results, Cont’d

Among the Waterfall projects , using only tickets classified unanimously :

- 100% of the defects are from missing requirements & none of the defects are from implementation errors,
- of the 100%, 87% are from missing D requirements & 13% are from missing G requirements.

Preliminary Results, Cont’d

Among the Agile projects:

- 100% of the defects are from missing requirements & none of the defects are from implementation errors,
- of the 100%, 95% are from missing D requirements & 5% are from missing G requirements.

Preliminary Results, Cont’d

In these projects, …

each missing D requirement could end up costing up to 10 times what it would have cost to have identified them during RE.

Ouch! ☹

This is MAJOR technical debt, from postponing full RE, done intentionally or not

Preliminary Results, Cont’d

None of the projects is doing a good job of finding D requirements in its scope, probably because none is spending enough time doing RE, if any at all.

However, Waterfall projects are doing a slightly better job than are Agile projects.

Plan for the Rest of the Talk

Reexamine past case studies,

done with no notion of D and G requirements,

with a D vs. G requirements lens:

- Challenged and Failed Projects
- Successful Project

Berry was one of the authors in each.

The original slides from each are used, with current remarks in red.

Challenged and Failed Projects

In all the challenged and failed projects, it appears that many if not a majority of the defects were from missing or wrong D requirements, ...

and not from missing G requirements and not from implementation defects.

Successful Project

The one project that focused its RE on finding all D requirements of its scope, letting RE continue until it had found the completion of the project’s scope delivered two versions of its software on time, matching its specification, with no known defects.

Challenged and Failed Projects

- Experiences of Requirements Engineering for Two Consecutive Versions of a Product at VLSC (2006)
- Requirements Determination is Unstoppable: An Experience Report (2010)
- Developers Want Requirements, but Their Project Manager Doesn’t … (2011)

Experiences of Requirements Engineering for Two Consecutive Versions of a Product at VLSC (2006)

https://student.cs.uwaterloo.ca/~se463/Slides/SoBerry.pdf

See Pages 4—7.
See Pages 20—45.

Because the version being developed is a refactoring (with no behavioral change) of the previous version, it’s clear that most of the defects were missing D requirements.

Requirements Determination is Unstoppable: An Experience Report (2010)

https://student.cs.uwaterloo.ca/~se463/Slides/RDisUnstoppable.pdf

See Pages 10—18.
See Page 35.
See Pages 39—41.

It’s clear from the nature of the so-called requirements *creep*, that most of the defects were missing D requirements.

Developers Want Requirements, but Their Project Manager Doesn’t … (2011)

https://student.cs.uwaterloo.ca/~se463/Slides/IsaacsBerryFullSlides.pdf

First, see Page 27 to see scope of PX.
See Pages 21—29.
See Pages 72—77.
See Pages 89—90.

Because PX is supposed to be identical in behavior to an *implemented* PY, the 37 missing requirements are clearly D requirements..

Successful Project

- WD-pic, ... its Development as a Case Study of the Use of its User’s Manual as its Specification (2002)

WD-pic, ... its Development as a Case Study of the Use of its User’s Manual as its Specification (2002)

https://student.cs.uwaterloo.ca/~se463/Slides/users.man.pdf

See Page 1.

https://student.cs.uwaterloo.ca/~se463/Slides/IcebergSlides.pdf

See Pages 113—132.

Serious upfront RE, planned for 2 months, took 5 months, but project planned for 10 months still finished on time, with 2, not just 1 planned versions, and very few, easy defects to correct.

WD-pic, … (2002), Cont’d

https://student.cs.uwaterloo.ca/~se463/Slides/OuData/Chaps3and4LihuaOuThesis.pdf

Via the D and G requirements lens:

Berry forced Ou to not start implementing until RE was done to his satisfaction (incomplete RE, no degree!).

Ou had focused her RE on trying to find all D requirements for Berry’s scope, and …

she nearly succeeded.

WD-pic, ... (2002), Cont’d

After RE took 5 months instead of 2, …

implementation went *much faster* than Ou expected, and she made up all the slipped time.

There was *almost no* backtracking and correction of the code and manual.

The little she had to do were to fix *wrong*, not missing, D requirements, whose effect were very localized.

For the Future

We need *more*

and *targeted*

empirical studies to confirm these preliminary findings.

For the Future

If these findings hold up, a modified agile lifecycle model is suggested:

- globally, agile iteration on backlog list to identify scope for each sprint.
- within each sprint, full upfront RE to identify all D requirements of the scope, to specify the completion of the sprint’s scope, and to build a complete test data set for the scope, testing also all exceptions.

Questions?