

SE463

Software Requirements: Specification & Analysis

Overview and Admin Notes

Fall 2026

Daniel Berry

with Input from Ahmed ElShatshat

SKIP TO EXCEPTIONS

Realities About Software Development Projects

Everyone says,

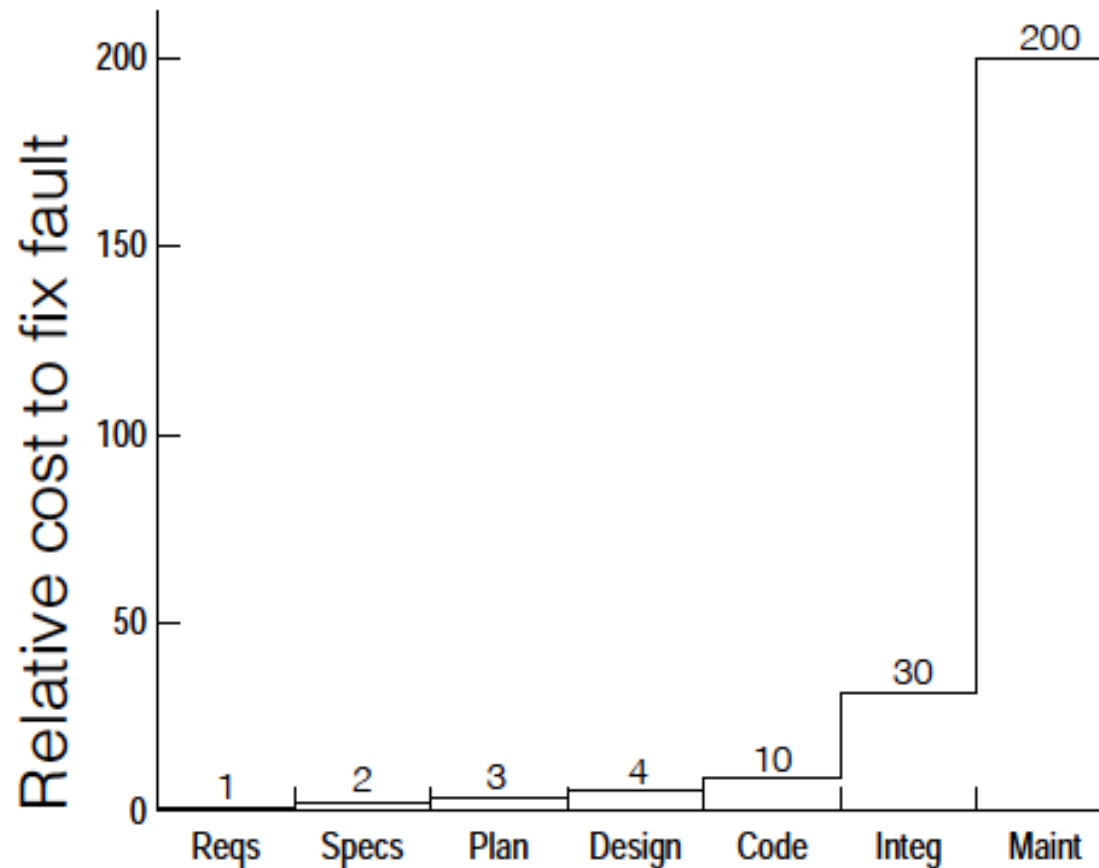
“We *know* that we should work out all the requirements before we start to code, but we *don't* have time!

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 1980s Data Show



Phase in which fault is detected and fixed

Recent data show larger numbers, e.g., 10 → 50

The 1980s Data Show

More recent data show larger multipliers, e.g.,
10 → 25.

These multipliers assume that you (plural) are fixing code that you (plural) wrote.

If one of the authors of the code is an AI, the multiplier is even higher!

Because before fixing anything you gotta spend time examining the AI-generated code to understand WTF the code does. 🙄

Start Coding Earlier, Finish Later!

Starting coding before all the requirements are worked out and specified completely means that

...

you have more defects arising from missing requirements, and ...

each costs to fix 10 times what doing it right the first time does, and ...

Start Coding Earlier, Finish Later!

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!

This leads to ...

the conclusion that maybe ...

Start Coding *Later*, Finish *Earlier*!

In other words:

- full *upfront* requirements analysis (Waterfall!) that is let to run its course until it all stakeholders say that it's done,
- then writing a requirements specification (RS) that is used to drive the downstream development.

Start Coding Later, Finish Earlier!

In other, other words, you gotta delay starting implementation until RE has run its course.

This truth goes against every middle-through-upper manager's guts.

Start Coding Later, Finish Earlier!

This truth goes against every middle-through-upper 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 er job if the project with a new-fangled method fails.

If the system fails, but the manager uses the traditional method, it's not er fault; that's life!

Reality of Your Capstone Project

So for your Capstone projects, you have been likely postponing working out the details of all requirements, because you don't have enough time.

You have probably picked a small viable set of requirements (a.k.a. features) as the scope of your prototype and are heading into design and coding without having fleshed out all the requirements' exceptions.

You don't have the time!

The Most Common Kind of Bug

We have discovered that about 94% of the defects in both waterfall and agile system developments are the result of missing requirements to handle exceptions to features that are already in the system.

I.e., the developers had started implementing each system before they had identified all the bad inputs to the features and had determined how to handle them.

The Most Common Kind of Bug

I.e., the developers had started implementing each system before they had identified all the bad inputs to the features and had determined how to handle them...

and *almost all* of the bugs that showed up later were because of this lapse.

Let's look at this situation in more detail!

Example of an Exception

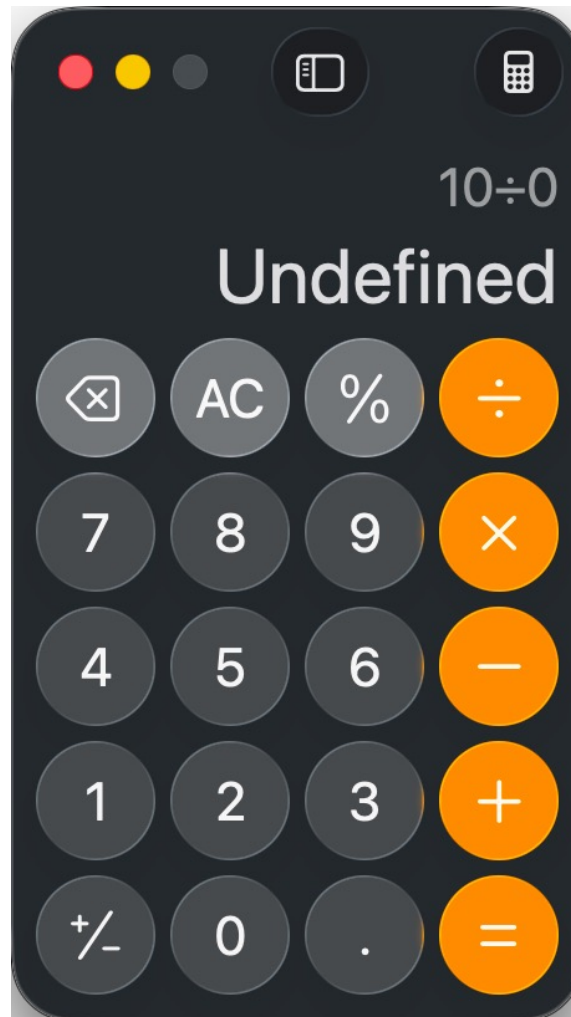
Consider a pocket calculator (PC): with requirements: +, -, *, and /
This is the *scope* of the PC.

An exception for the PC's "/" requirement occurs when the denominator is "0".

The requirement for detecting that exception is "in /, the denominator cannot be 0".

This requirement specification needs to specify the response to this exception, e.g., ...

Example of an Exception



Another Example of an Exception

Consider a program MS that inputs two ascendingly sorted (AS) files of records of varying and unbounded lengths and outputs a sorted file that is the merge of the input files.

An exception for MS occurs when an input file is not AS. In this case, MS's output file will not be AS.

The requirement for detecting that exception is “no input file can be not AS”.

This requirement specification needs to specify the response to this exception.

If You Start Coding Too Soon

So, if you start coding the PC, and you are not aware of “/”’s exception, you will write code that will break if ever / is presented with a 0 denominator.

At that point, ...

If You Start Coding Too Soon

So, if you start coding MS, and you are not aware of its exception, you will write code that will break if ever one of its input files is not AS.

At that point, ...

If You Start Coding Too Soon

At that point,
depending on when the discovery is made,
fixing the code could cost 10–200 times what it
would have cost to have specified the exception
upfront so that coding took it into account from the
beginning.

Sometimes, fixing a missing exception-handling
requirement requires restructuring , e.g., as in MS,
in which more of its unbounded-length input will
have to be kept for later comparisons.

Inescapable Fact Affecting Exceptions

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 Exceptions is:

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

The Nature of Exceptions

If you start coding with exception-handling requirements missing from the specification, and you discover their existence during coding, you will have to specify the missing exception-handling 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!

The Nature of Exceptions

This is a stupidly expensive way to discover and specify exception-handling 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* an exception-handling requirement is discovered, then a *user* — the best defect finder in the universe — will eventually discover it, ...

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

BACK TO OTHER SLIDES

SE463

Software Requirements: Specification & Analysis

Overview and Admin Notes

Fall 2026

Daniel Berry

with Input from Ahmed ElShatshat