SE463
Software Requirements: Specification & Analysis

Overview and Admin Notes

Fall 2025

Daniel Berry
with Input from Ahmed ElShatshat

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.

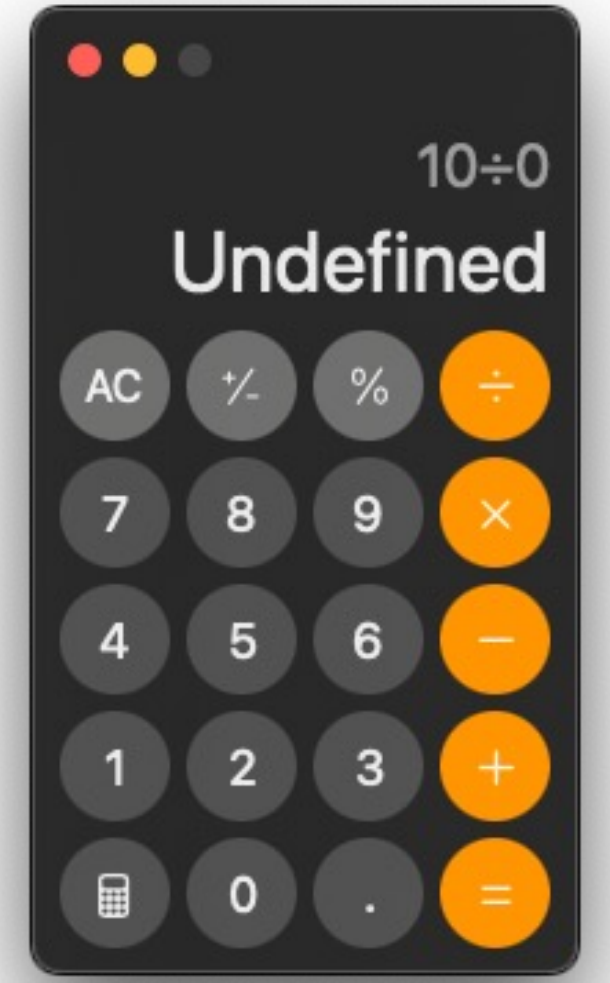
You have probably picked a small viable set of requirements (a.k.a. features) as the scope of your prototype and are heading into design and coding without having fleshed out all the requirements' exceptions.

You don't have the time!

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

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

Daniel Berry[1], Anzira Rahman[2], Victoria Sakhnini[1],
Abhishek Dhakla[1], and Márcia Lucena[3]

[1] Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1, Canada
{dberry,vsakhnini,adhakla}@uwaterloo.ca

[2] School of Information Technology
York University
Toronto, ON M3J 1P3, Canada
anzira.rahman@gmail.com

[3] Department of Computer Science and Applied Mathematics
Universidade Federal do Rio Grande do Norte
Natal, RN 59078-900, Brazil
marciaj@dimap.ufrn.br

Abstract. Some believe that Requirements Engineering (RE) for a computer-based system (CBS) should be done upfront, producing a complete requirements specification before any of the CBS's software is written. A common complaint is that (1) new requirements *never* stop coming; so upfront RE goes on forever with an ever growing scope. However, data show that (2) the cost to modify written software to include a new requirement is at least 10 times the cost of writing the software with the requirement included from the start; so upfront RE saves development costs, particularly if the new requirement is one that was needed to prevent a failure of the implementation of a requirement already included in the scope. The scope of a CBS is the set of requirements that drive the CBS's implementation.

We believe that both (1) and (2) are correct, but each is about a different category of requirements, (1) scope determininG (G) or (2) scope determineD (D), respectively.

Reexamination of the reported data of some past case studies through the lens of these categories indicates that when a project fails, a large majority of its defects were due to missing D requirements, and when a project succeeds, the project focused its RE on finding all of its D requirements.

The hypothesis that waterfall methods (WMs), with their upfront RE, do a better job of avoiding missing D requirements in developing CBSs than do agile methods (AMs) was not supported by the data from 8 WM projects and 8 similar AM projects in one company. In fact, the null hypothesis, that there is no difference between WMs and AMs in avoiding missing D requirements in developing CBSs, seems to be true for WMs and AMs, as they are typically practiced.

It appears that intimate knowledge of the domain and development of a CBS is necessary to be able to classify the CBS's defects arising from missing requirements as D or G with respect to the CBS's scope.

Finally, software and requirement engineers are able to learn from a half-hour lecture about D and G requirements to correctly classify any requirement of a familiar CBS as D or G with respect to the CBS's scope.

Keywords: Agile methods, Cost to repair defects, Defect tickets, Empirical studies, Exceptions and variations, Missing requirement, Requirements specification, Scope, Scope-determined requirement, Scope-determining requirement, Software development lifecycle, Sprint, Upfront requirements engineering, Waterfall methods

1 Introduction

A current great debate [4, 11, 19, 22, 30, 32, 38, 39, 49, 50, 58, 61] in Requirements Engineering (RE) is whether requirements for a computer-based system (CBS)

1. should be identified upfront before design and coding begin, as in the waterfall lifecycle [51], or

2. should be identified incrementally, interleaved with design and coding of requirements identified so far, as in the spiral or agile lifecycles [2, 14].

Here, "identifying requirements for a CBS upfront" means "identifying requirements for the CBS in their entirety".

The argument for identifying requirements upfront is that catching and repairing a requirement defect, i.e., a missing or incorrect requirement, during coding costs about 10 times the cost of catching and repairing it during upfront RE [13, 23, 25, 27, 33, 43, 45, 52]. Thus, developing a CBS using a waterfall method (WM), with requirements determined for the entire CBS upfront before beginning any coding, leads to the shortest overall development time [6, 9, 13, 17, 29, 44, 55].

The arguments for identifying requirements incrementally are that

- requirements never stop coming; if design and coding do not start until *all* requirements are identified, design and coding will *never* start! and
- many requirements change as more and more of a CBS is developed and as the world changes as a result of the CBS's being used [36, 37]; many requirements that were identified before will be thrown out; and the time spent identifying these thrown-out requirements is wasted!

Thus, we should develop CBSs using an agile method (AM), with requirements determined for each sprint of coding only at the beginning of the sprint.

Attempts to settle the debate with empirical data have failed. Empirical studies go both ways and are overall inconclusive [4, 19, 32, 39, 49, 58]. Consequently, the choice of CBS development lifecycle, upfront RE or agile, to use in a CBS development project is made on the basis of gut feelings informed by experience and a recognition that if a project does something different from what is established practice, and the project fails, the heads of the project's decision makers will roll.

The reason that data have not decided the debate is that each side is right! There are two phenomena, A1 and A2, happening:

A1. Requirements *do* never stop coming; and many requirements *do* change, resulting in wasted effort.
A2. There *are* a lot of requirements defects that *can* be found and repaired early if one is spending enough time doing RE, and a complete requirements specification (RS) for a CBS *dramatically reduces* the incidence of expensive-to-repair requirement defects that appear in the code for the CBS.

We believe that the two competing phenomena, A1 and A2, are talking about two different kinds of requirements, respectively:

K1. One kind of requirement often cannot be identified until users are trying some version of the CBS and notice its necessity. Such a requirement is best handled incrementally, so that when it is finally identified, it is less likely to be later discarded as unneeded [2].
K2. The other kind can be identified before design and coding if enough time is devoted to RE. It is wasteful to leave such a requirement to be found and repaired only later in the lifecycle when it is more expensive to repair [9].

We believe that the empirical studies are inconclusive because none of them distinguishes the different kinds of requirements. In any study, both phenomena, A1 and A2, are happening. Because the two kinds of requirements are not treated differently, A1 and A2 are not being treated differently.

We have identified a new binary categorization of new requirements being considered for addition to a CBS:

C1. The first category of requirement is a *scope determininG (G)* requirement, and
C2. the second category of requirement is a *scope determineD (D)* requirement.

Here, the *scope* of a CBS is the set of requirements — a.k.a. use cases or features — the CBS implements. This categorization has been identified in the past under different names. For example, among use cases, a variation or exception of another use case is a D requirement, but a new, independent use case is a G requirement. New are the names of the categories, which are more suggestive of

- how the categorization of a requirement can be done and
- how knowledge of the categorizations of candidate requirements for a CBS can be used during RE for the CBS and during its subsequent development.

Go to Page 23!

We would gain from Rahman's familiarity with data to be able to obtain a correct categorization of any requirements identified as the source of a defect ticket, to get as close as possible to a correct categorization to serve as our ground truth or gold set. This ground truth would give us the opportunity to answer the RQ of Section 6 plus a number of research questions (RQs) that had emerged during the research to produce Sections 1 through 5 of this article:

1. Is the categorizing of requirements as D or G requirements operational, i.e., it can be done easily by any competent software developer, perhaps after a little bit of training?
2. Does one need to be a member of the team developing a CBS in order to correctly categorize the CBS's requirements?
3. How correctly do experienced software engineers that are ignorant of the domain of a CBS categorize the CBS's requirements?
4. How well do experienced software engineers that are ignorant of the domain of a CBS agree on their categorizations of the CBS's requirements?
5. Is the D-vs-G categorization of requirements useful in identifying what requirements are worth focusing on in an upfront RE in each sprint in an AM development?
6. In each of upfront RE and AMs, what are the frequencies of D and of G requirements among the missing requirements that are the causes of defects discovered during the development and deployment and thereafter?

7.1 Frequencies of Missing D and G Requirements in Defect Tickets

We[10] began a study to compare the frequencies of missing D requirements and missing G requirements in WM and AM CBS development projects. As described in Section 6 of this article, A WM project is characterized by upfront RE of all kinds of requirements until all the stakeholders agree that the RE is done and production of a requirements specification (RS) for the whole CBS before coding begins [9,51]. An AM project is characterized by the CBS's being developed in a sequence of sprints, which is driven by the requirements in a BLL. At any time, even during a sprint, any new, discovered, or missing requirement is added to the BLL, and in each sprint, a small group of requirements is taken from the BLL, a US and test cases for the set are written, and the sprint is coded. [2, 15]. Rahman *et al* provide in their Section 1.1 more details on the methods and their differences [48].

7.2 Hypotheses for Study

We went into the first study with the belief that

1. practitioners of WMs took their upfront RE seriously, and
2. practitioners of AMs avoided anything resembling upfront RE.

We, thus, believed that the data of Rahman's study would fall into the upper right cell of Table **??** at WMs: Full Upfront RE $\times$ AMs: Only User Story / Sprint. This belief was bolstered by Rahman and Cysneiron's conclusion that their results suggest that use of AMs "may lead to higher number of defects and in consequence, to higher cost to maintain the software".

So, we hypothesized the prediction of the upper right cell of Table **??**:

WM $<$ AM:
A WM CBS development project has fewer missing D requirements than does an AM CBS development project of the same domain and of similar size and complexity.

WM $<_r$ AM:
A WM CBS development project repairs fewer missing D requirements than does an AM CBS development project of the same domain and of similar size and complexity.

WM $<_t$ AM:
A WM CBS development project spends less time repairing missing D requirements than does an AM CBS development project of the same domain and of similar size and complexity.

[10] Until further notice, "we" means "Berry, Lucena, Rahman, and Sakhnini".

It soon became clear that we could not get access to any additional data about the defects, in particular about the time and costs to repair the defects. So we could test only the main hypothesis and not either of the subhypotheses.

To test Hypothesis **WM** $<$ **AM**, we decided to reuse the defect tickets that Rahman *et al* studied, of 8 WM CBS development projects and 8 AM development projects in one company, all in the same commercial application domain, which is the company's niche [48]. As described in Section 5.8 of this article, Rahman had selected for her study a set of 8 WM projects and a set of 8 AM projects, such that the sets are well matched in project sizes and complexity. In fact, in addition, the 8 WM projects had 66 defect tickets, about 8.25 per project, and the 8 AM projects had 68 defect tickets, about 8.5 per project. The difference between the numbers of defects in the two sets of projects is not significant, boding badly for support of the hypothesis.

It must be said that there was no control over how well each project followed its method, WM or AM. However, each WM project had an RS with unknown completeness, and each sprint of each AM project had a US, but few test cases. It appeared in retrospect that each RS or US was complete in the sense of defining a scope for the project or the sprint, but incomplete in the sense of providing also the scope's D requirements. This observation is consistent with the numbers of missing D and G requirements we found in the defect tickets. It is fortunate that the RSs and USs were available, because measuring the projects' sizes and complexity for selection of the projects for the study depended on having these artifacts to measure.

The defect tickets were post deployment, and thus would be about defects that would be visible to users. Rahman and Cysneiros determined that *none* of the defect tickets were about implementation errors. Thus, each defect ticket is about a missing requirement of some kind.

Remember that the defect tickets were from a company at which Rahman worked before, during, and immediately after doing her master's thesis. Thus, Rahman was an expert in the domain of the projects, and could be expected to understand the defect tickets. We tasked Rahman with the job to classify each defect ticket as one of (1) an implementation defect, of which there should be none, (2) a missing D requirement, and (3) a missing G requirement. The difficulty for Rahman is that she last looked at the tickets in 2014, about 10 years earlier. However, among the authors, she is quite young!

Berry gave to Rahman an alive version of the talk found in the directory that is pointed to by the text "click this sentence to be taken to a folder with a READMEfirst, the video, and the slides" at the page pointed to by the URL https://tinyurl.com/sp26j2u7. Then, Rahman filled the questionnaire[11] that is pointed to by the text "To answer the questionnaire, click this sentence." in the same page. This questionnaire tests a participant's understanding of the concepts of D and G requirements by giving a specification that carefully describes the scope of a CBS that is similar to one in common use and then asking to classify each of 14 specific requirements as a D or as a G requirement. Rahman got only 8 of the 14, 57%, of the answers correct. Berry discussed the wrong answers with her to improve her understanding. Rahman felt that this discussion had clarified the meanings of the two classifications and felt confident that she could classify the missing requirements in the defect tickets correctly.

Rahman made a first pass in classifying the defect tickets. She determined in this first pass that none of the defect tickets were of implementation defects, confirming what she and Cysneiros had arranged more than 10 years earlier, when they removed from the defect tickets of the 16 projects the very few that were of implementation defects [48]. At the end of this first pass, there were about 2 dozen classifications that Rahman was not certain of. She and Berry discussed these classifications. Berry knew that he had to avoid influencing Rahman's answers in a way that would cause her to classify incorrectly. He knew nothing of the domain, making it difficult to influence her as an expert could. He did understand the natures of D and G requirements, but any influence arising from this understanding is desired. So, for each discussed defect ticket, Berry explained

- circumstances in which it would be a missing D requirement and
- circumstances in which it would be a missing G requirement.

Then Rahman would make her *own* decision. Because Rahman made her own decisions for each defect ticket, and she is a domain expert, we are confident that Rahman's classifications are right. We declared her classifications to be the ground truth, i.e., the gold set, for the later work!

Rahman already knew that all the defect tickets were from missing requirements. However, she was visibly shocked that most of the defect tickets, 96.3% , were the result of missing D requirements. Only 3.7% were the result of missing

[11] This is the questionnaire described in Section 10.

G requirements! So, it looks like most servicing of defects was for puting out fires initiated by overlooked exceptions and not for enhancing the system.

Among the WM projects, 97% of the defect tickets were the result of missing D requirements, and only 3% were the result of missing G requirements! Among the AM projects, 96% of the defect tickets were the result of missing D requirements, and only 4% were the result of missing G requirements!

In these projects, regardless of development method, each missing D requirement could end up costing up to 10 times what it would have cost to have identified them during RE. This cost is *major* technical debt, from postponing full RE, whether intentionally or not. 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. Each is spending all of its time putting out fires.

There is no significant difference between WM projects and AM projects. Consequently none of the hypotheses, **WM** $<$ **AM**, **WM** $<_r$ **AM**, and **WM** $<_t$ **AM**, can be supported. See Section 8 for a discussion of the implications of this conclusion.

7.3 Can Non-Experts Classify Defects Correctly?

Independently of having Rahman classify the defect tickets to test the hypotheses, we[12] investigated how much domain expertise is needed for correct classification of requirements.

Before authors Dhakla and Rahman had become authors of this article, in order to debug the instructional materials to go with any classification experiment, we had authors Berry and Lucena do a pilot classification of the defect database. We decided to leave author Sakhnini out of this pilot to allow a later pilot with debugged instructional materials to be done by someone not contaminated in any way by the first pilot. Each of Berry and Lucena did er own classification after discussing the instructional slides. They then met to discuss what they had done, to try to come to a consensus, and to identify improvements to the instructions. Then, we had all three of Berry, Lucena, and Sakhnini classify the defect tickets. Note that each of Berry and Lucena is mature enough in years that E really could not recall er previous classification in detail. The result was that each actually differed from er previous self almost as much as from Sakhnini. As is shown in Tables 8 through 10, there is very poor agreement among the classifications. Given that Berry, Lucena, and Sakhnini are totally ignorant of the domain of the CBSs involved in the defect tickets, this poor agreement among the classifications says that domain knowledge or even domain expertise is essential for correct classification.

Then, we got the idea of recruiting from students of and recent graduates of our undergraduate BSE (Bachelor of SE) program someone who had expertise in the relevant domain, perhaps from a cooperative education job with a company in the domain.

Author Dhakla responded to the recruitment ad and convinced us that he knows the domain. We gave him the final version of the lecture, and he classified the defects and wrote down reasons for the difficult cases. He, Berry, and Sakhnini discussed his answers and corrected some misunderstandings about the meanings of "D" and "G" and how to interpret some fact situations about the defects. He did the classifications again, and we declared Dhakla's second classification to be his classification.

So with each author other than Rahman having classified the defect tickets, we [13] were able to determine how much domain expertise is needed for correct classification of requirements,

None of the authors other than Rahman had ever worked at the company that provided the defect tickets. Only Dhakla has any domain knowledge from having worked in the same industry during his cooperative terms as a student at the University of Waterloo [60]. So, Berry, Lucena, and Sakhnini would be considered completely domain ignorant, Dhakla would be considered domain knowledgeable, and Rahman would be considered a domain expert. Berry, Lucena, Sakhnini, and Dhakla are collectively domain nonexperts, to distinguish them from Rahman.

Out of the 134 defect tickets, all of missing requirements, Rahman classified only 5 as arising from missing G requirements. Of these same 134 defect tickets, Berry classified 9, Lucena classified 41, Sakhnini classified 48, and Dhakla classified 12 as arising from missing G requirements. On average, the domain nonexperts classified 27.5 defect tickets as arising from missing G requirements. The nonexperts were quite generous in assuming that the projects were actively enhancing their CBSs instead of constantly putting out fires caused by their lack of sufficient RE.

[12] Until further notice, "we" means "Berry, Lucena, and Sakhnini".

[13] For the rest of the article, "we" means "Berry, Dhakla, Lucena, Rahman, and Sakhnini".