SE463
Software Requirements: Specification & Analysis

Overview and Admin Notes

Fall 2025

Daniel Berry
with Input from Ahmed ElShatshat

Table of Contents

INTRODUCTION

Gender-Nonspecific Third-Person *Singular* Pronouns

“E”, “em”, and “er” are gender-nonspecific third-person singular pronouns in subjective, objective, and possessive forms, respectively.

Example:

Someone was hungry.

E gave to Joe er last dollar.

Joe gave to em his lunch.

Gender-Nonspecific Third-Person *Plural* Pronouns

The more common “they”, “them”, and “their” are more ambiguous, and ...

ambiguity matters in this course about software system requirements!

What does

The teacher gave the students their lunch.

mean?

Gender-Nonspecific Third-Person *Plural* Pronouns

The teacher gave the students their lunch.

Does their refer to teacher or to students?

And how many lunches?

consider:

The teacher gave the students er lunch.

The teacher gave the students their lunches.

Less ambiguous.

Welcome

- … to Software Requirements: Specification and Analysis
- This course is known as:
  - ECE451
  - CS445
  - CS645
  - SE463
  - SE 1 (not an official course, just for discussing all the courses together)

Welcome

- It is one course of a three-course set on software engineering:
  - ECE452/CS446/SE464 (SE 2): Software Design and Architecture.
  - ECE453/CS447/SE465 (SE 3): Software Testing, Quality Assurance, and Maintenance.

A BRIEF HISTORY LESSON

Changes

- Previously, the courses could be taken only in order, as they shared an incremental project
  - SE1 ➔ SE2 ➔ SE3.
- In Fall 2008, the three courses were de-coupled, so they can be taken (in theory) in any order.

More Changes

- In the Summer of 2018, SE463 was changed so that you wrote a requirements spec for your capstone project (FYDP) that you were working on in SE490 in the same term.
- In the Summer of 2020, the course was changed so that the TA you have in SE463 and in SE490 are the same person; so, you have only one TA to deal with on what is really one project.

More Changes, Cont'd

- In the Summer of 2021, the course brought in
  - some more about Agile development
  - requirements for AI and ML systems
- In the Fall of 2024, the course brought in considerations of the cost of AI assistance to system development

ADMINISTRATIVE DETAILS

Dan

Dan

Prof. Daniel Berry

- **Office hours**: by appointment made by e-mail, but feel free to knock on his room door, if the door is closed.
- Email: dberry ATT uwaterloo DOTT ca
- Web: http://cs.uwaterloo.ca/~dberry
- Appointments are normally in person, but an appointment can be via Zoom. (It has to be Zoom and not Teams so that Berry can read lips.)

More about Communicating with Dan

The reason I have no telephone is that I am nearly deaf. I do not sign, but I do read lips. So I cannot use a voice-only telephone. I can use a video communication medium *if the bandwidth of the connection is high enough that the image gets updated at the frequency of television or movies and thus, lip movement is smooth enough to be decipherable.*

Still More about Communicating with Dan

This means that *you will need to show your face in any virtual meeting, if you want me to understand what you are saying*.

Dan outside of the classroom

- I have been programming since 1965 when I became 17.
- I learned FORTRAN at an NSF (USA) Summer Science Training Program at Illinois Institute of Technology.
- In 1966, I programmed a matchmaking program to match each of a group of high school students with a date for a youth-group sponsored dance.
- I have been writing software ever since ☺️.

Dan outside of the classroom

- I'm a researcher in the field of software engineering (SE), particularly requirements engineering (RE).
- I specialize in:
  - Requirements Elicitation,
  - Ambiguity in Natural Language Requirements Descriptions,
  - Creativity in Requirements Elicitation, and
  - RE for AI and ML systems.
- I dabble also in Electronic Publishing: formatting, typography, etc.

Dan outside of the university

- I swim, skate (both kinds), and ski (downhill & water).
- I am considered a good cook.
- I am even semi-professional as a cook, having catered two weddings, one not my own!
- I am a “Star Trek” (since 1966!) , “Big Bang Theory”, & “Young Sheldon” fan.
- I write scientific satire.
- I write Biblical commentary.
- I have 3 grown children & 4 college-aged grandchildren.
- I love programming.

[Now, do I seem human enough? ☺]

The Course TAs

- Ashu Adhikari, a8adhika ATT uwaterloo DOTT ca
- Mehrad Haghshenas, m3haghsh ATT uwaterloo DOTT ca
- Ende Jin, e5jin ATT uwaterloo DOTT ca
- Michael Lapshin, mlapshin ATT uwaterloo DOTT ca
- Gengyi Sun, g25sun ATT uwaterloo DOTT ca

They will be your mentors, one per group.

Grading Scheme

Project	40 %
Assignments	10 %
Final exam	50 %
Total	**100 %**

Course Web pages

https://outline.uwaterloo.ca/

- This page is the stable official, university-sanctioned-and-mandated course outline that describes all the things about the course that are not expected to change during the term, such as policies, due dates, grading schemes, etc.
- It has a link to the dynamic course page at http://www.student.cs.uwaterloo.ca/~se463.

Course Web pages

http://www.student.cs.uwaterloo.ca/~se463

- This is the dynamic course Web page that has all the things that change during the term.
- It is updated typically before and after each lecture, based on what is planned for and what happened in the lecture.
- Watch for the details about what each deliverable requires on its due date.
- It has a link to the stable course Web page at https://outline.uwaterloo.ca.

Course Web pages

https://outline.uwaterloo.ca/

http://www.student.cs.uwaterloo.ca/~se463

These two sites take the place of a textbook, which the course does not have.

Course email

se463 ATT uwaterloo DOTT ca

- Please send most questions here
  - You may send to Dan (dberry ATT uwaterloo DOTT ca) questions that relate to course administration or are personal in nature directly.

Communication

I will communicate with the class by e-mail, using your watlam, uwaterloo.ca e-mail addresses, and the dynamic course Web page.

Not reading either and the stable course Web page in a timely fashion is not an excuse for any lapses on your part.

Learn & Piazza

Heretofore, I didn't use *Learn* and *Piazza* because they do not provide an order of magnitude improvement over what I am already using using *vi*, *html*, the shell, and regular e-mail.

There are things that I do using *vi*, *html*, the shell, and regular e-mail that they don't do.

See https://doi.org/10.1109/MC.2004.247

Learn

By popular demand of past students:

I have set up a Learn account for managing the registration of the SE463 groups of students working together.

If at least one of you is willing to help me learn Learn, I will change deliverable submission from the e-mail-based way it is now to using Learn's submission dropboxes.

SE463 Groups and SE490 Teams

If you're also in SE490 this term, then your SE463 group is the same as your SE490 team, and the SE463 group's number is the same as the SE490 team's number.

If you are not also in SE490, e.g., you are in a different engineering, then your SE463 group has no SE490 counterpart team, i.e., there is no SE490 team whose number is your group's number.

SE463 Groups and SE490 Teams

Because there are two streams active now, there may be other configurations that we'll have to wing on the fly.

E.g., if you took SE490 last term with some or no members of your SE490 team, and this term, other members of that team are taking SE490, but not SE463, then your SE463 group's number will be the same as that SE490 team's number, because it's the same capstone project, and should have one TA.

Term Project

If you're in SE490:

- The term project for SE463 is to write a requirements specification for your planned prototype of your SE490 capstone project.
- Your group's SE463 TA is also your team's SE490 TA.
- That is, your TA will be familiar with all aspects of your capstone project.

Term Project

If you're in another Engineering, you will either:

1. join another group as one of its *smart ignoramuses* that thinks out of the group's box, or
2. form a group of one, or maybe, two with a smart ignoramus, to write a requirements specification of a major part of your non-SE capstone project.

The preference is #2 above.

Term Project

If you're in CS, you will either:

1. form a group with other CSers for a project of your *mutual* choosing,
2. join another group as one of its *smart ignoramuses* that thinks out of the group's box, or
3. form a group of one, or maybe, two with a smart ignoramus, to write a spec. of a major part of your CS capstone project, if you have one.

Term Project

- Your group/team will be assigned a TA, who will serve as your mentor and will grade all of your deliverables.
  - Thus, you'll get some consistency in marking.
  - E will initially know nothing about the project and will be learning along with you.
  - E will give you feedback on your interaction with your customer and on your deliverables.
  - E will meet with your group/team frequently.

Term Project

- Your job:
  - to create detailed models of the various entities and processes,
  - to decide what features should be there,
  - to decide the correct functionality of these features,
  - to work out *all exceptions and variations* of these features,
  - eventually, to use these models and decisions to create a specification describing your prototype.

Final Deliverable

The final deliverable is a specification of your prototype in the form of a user's manual (UM), an SRS, a complete set of scenarios, a complete set of UML models, velc. (discuss it with me)

("vel" = "exclusive or" in Latin; so "velc." is to "exclusive or" as "etc." is to "and")

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!"

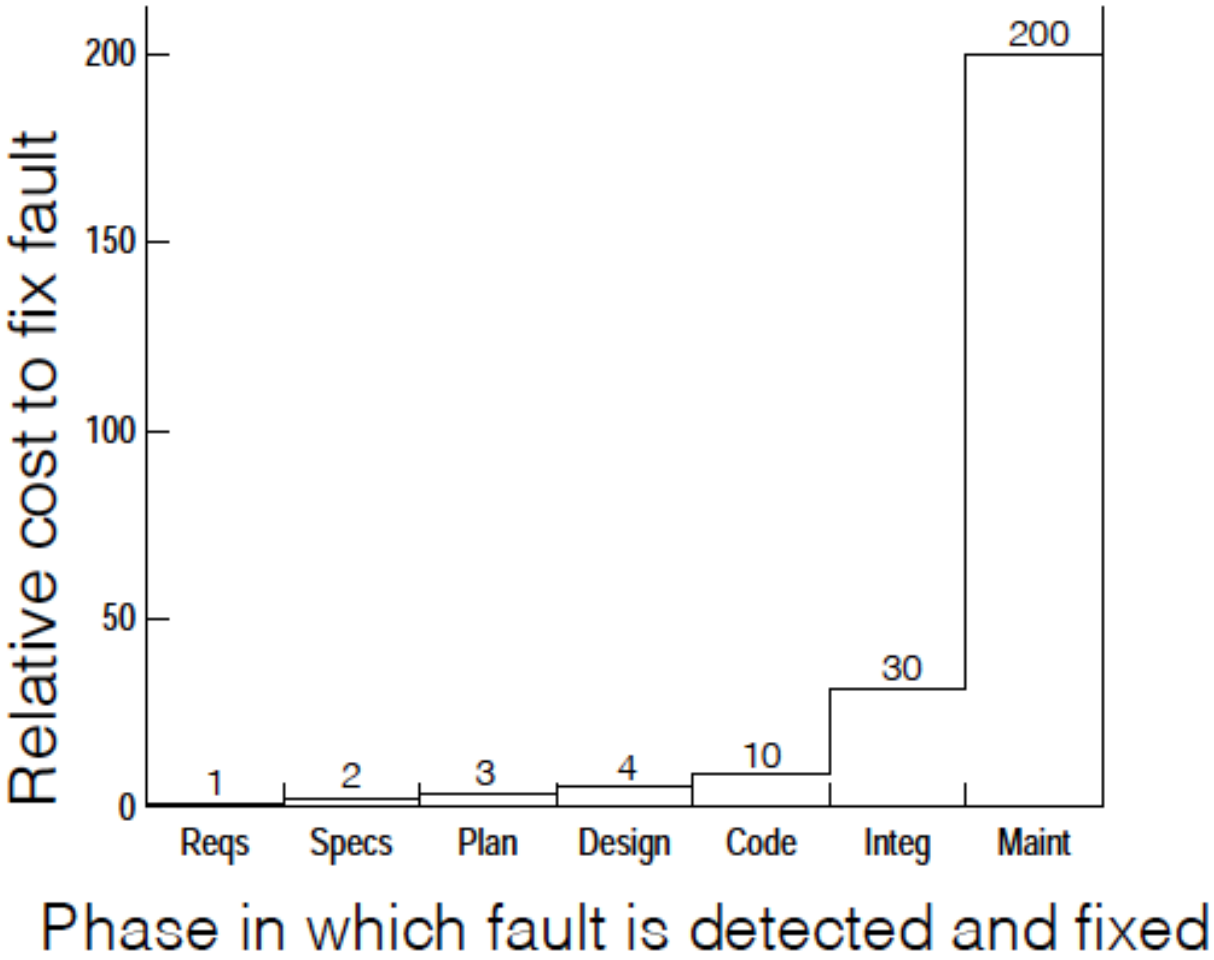
Wrong!

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



"fault" is often called "defect" or "bug"

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 ...

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.

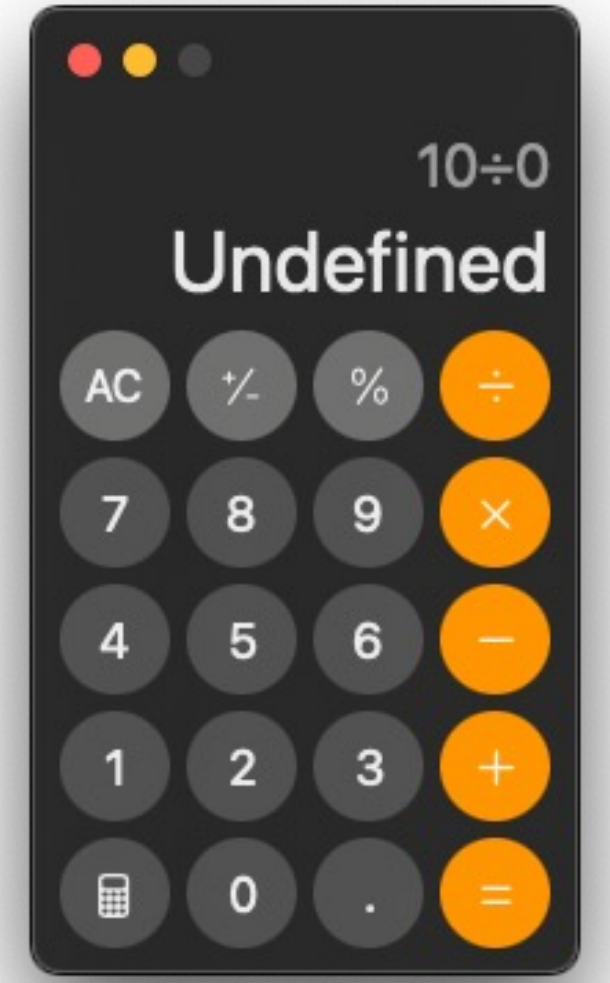
You have probably picked a small viable set of requirements (a.k.a. features) as the scope of your prototype and are heading into design and coding without having fleshed out all the requirements' exceptions.

You don't have the time!

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.

The Gift of SE463!

OK.. So, what SE463 is going to do is give you the gift of time, time that you *must* spend to pass SE463, to work out *all* the requirements for your chosen scope your capstone project, including for handling *all* their exceptions!

So thank me! ☺️

No applause yet? ☺️

How The Gift Will Be Given

And the way I am going to give you this time is to take advantage of my dictatorial powers of not giving you a passing grade in SE463 unless you satisfy the course requirements, which includes writing a specification of the selected set of requirements, the selected scope, of your capstone prototype, in which each of the exceptions of the scope's requirements has been identified and a response for it is specified.

To Be Clear

- I accept that you may think that what I am forcing you to do is a colossal waste of time!
- No one writes these specifications any more, not in real life where everything is Agile, and not even full Agile with writing of a full set of test cases upfront for each story.
- “Hey man! get with the times!” ☺️

Double Whammy

- This writing of a specification for SE463 has always been a colossal waste of time, especially when I concoct a system for you to specify; the system is totally useless and boring to you.
- Once the term is over, you'll never touch it again in your life!
- Double whammy: useless work for a useless system.

But Now!

- But now, the system is your own capstone!!!
- I hope that it’s not boring to you!
- So at most, the specification is useless!
- You’ll at least have learned something about the capstone project that you did not know before, even it it’s only some potential bugs that you did not think of before!

But Now!

- However, it *could* be different from in the past.
- The specification *just could* end up being interesting and useful.
- It *could* end up improving your capstone system immensely!!!
- Wow, a useful class project!!!
- Will miracles never cease!!!

Maybe Not!

- If you believe that what you produced is indeed useless to your capstone, then at the end of the term, *just ignore the specification and continue as before*!
- Regardless, if you do a good job on the stupid inane requirements spec that I force you to write, you'll get a good grade in this course!
- No different from when the spec is of *my* system!

But It's Not Agile!

But but but …

- *Nobody* does it this way any more!
- We're Agile!
- No upfront requirements engineering (RE)!
- It's in the Agile Manifesto!!!
- Get with the times!

Agility Works!

Yes.. Agility works, but it's a hideously expensive way to discover all the exceptions and implement their detection and responses.

In Agile developments, an exception to a requirement is typically discovered only during the sprint that implements the requirement, when implementation costs 10 times what it would have cost if it were discovered earlier…

Costs

The cost to *fix* the code to handle the discovered exception is

at least 10 times

the cost to write the code with the handling of the discovered exception already specified, …

so that nothing needs to be *fixed* later.

Data Are Clear

The data are clear.

Spend more time figuring out reqs upfront

→

- implementation is faster and
- the resulting code has fewer defects.

But, But, But...

New requirements keep coming and coming,
and even after the system is deployed.
So, it's impossible to specify *all* requirements.

So why bother writing a specification?

A Big But to the But, But, But...

That’s no excuse for not dealing with the exceptions to the requirements that you have decided are in the scope of what you are building,

especially if the scope will be deployed to users,

who have this annoying habit of discovering and even exploiting all the exceptions ...

Damn them!

A Coop Job is Not Quite Real Life

Note that what happens in a coop term is *not quite* real life.

Your employer has you for only 4 months.

So, to squeeze the max utility out of you, your employer gives to you, on a silver platter, full specs of your scope, …

worked out the hard way (like you're gonna do in this course), in advance of your arrival, by its regular staff.

A Coop Job is Not Quite Real Life

The real irony is that the employer's regular staff do not learn that what they did for you to save time would save time also for them.

Maybe it's because they would need to notice that the sum of (1) the time to prepare complete specs for you and (2) your implementation time is less than the time that they spend implementing without having prepared complete specs.

An Aside About Agile Methods

I have been talking about Agile methods as I believe most are actually doing it (assuming my observations are representative):

A sprint:

- Pick a backlog item.
- Write a user story for it.
- Implement it.

An Aside About Agile Methods, Cont'd

Actually, according to the manifesto, there should be two more steps to a sprint:

- Pick a backlog item.
- Write a user story for it.
- ***Write thorough test cases for the user story***.
- Implement it.
- ***Test it against the test cases.***

An Aside About Agile Methods, Cont'd

Many skip the additional steps.

After all, a set of upfront test cases *is* effectively and really a requirements specification ☹

Bleah… more paper work!

An Aside About Agile Methods, Cont'd

But think about it!

Doing these extra steps requires identifying the exceptions to the requirements for the sprint!

So actually, a *properly* done full Agile method will *not* suffer the cost disadvantage I described,…

but who does a proper full Agile method??

BOBW (Best of Both Worlds)

Making you do upfront RE for your capstone's scope as another, required course's main deliverable is a way to allow you allow you to

- try both Agile and upfront methods on the same project
- and compare the outcomes
- without taking any time away from the agile work you're doing in the capstone.

Can Make a Rational Decision

Based on what happens with this comparison, you can decide rationally, based on real experience, what you will do in the future!

Wow!!!!!

TESTIMONIALS

Surprise Discovery in S’20 – S’21

- In Summers of ’20 – ’21, we found that about 1/2 of the teams actually changed their capstone systems for what they thought was the better, using what they learned in the SE463 specifications. Wow!!!

Surprise Discovery in S'20 – S'21

- A few of the teams even began to consider SE463/SE490 as one course …

Is that a good thing?

Surprise Discovery in S'20 – S'21

- A few of the teams even began to consider SE463/SE490 as one course …
- and complained that writing the spec was taking time away from implementation! ☹
- Whoa!!! They never had that extra time in SE490 in the first place.
- I could have assigned a concocted system totally unrelated to the capstone.

So Let's Keep it a Gift

- The SE463 assignment to produce a specification of your SE490 project is a gift of time to the SE490 project.
- As with any other gift, you may or may not find it useful.
- If it is, ENJOY its fruits.
- If it is not, well… at least you satisfied your SE463 requirements in a more interesting way than it used to be!

Ain’t the Gift Nice?

Aren’t I nice? ☺️

You may now applaud! ☺️

Some Caveats

You will note that the description of each deliverable is rather vague, …

stating what is to be delivered in a way that is independent of the system being described.

This is intentional for two reasons:

- Each capstone project is different, …
- It's like in a real-life RFP: …

Each Capstone Project is Different

Each capstone project is different, and the description has to work for any project.

In RE terms, the description describes WHAT you must deliver without describing anything about HOW to deliver it 😀.

It’s Like in a Real-Life RFP

It’s like in a real-life RFP: Your client and users don’t know enough about the system to describe it in more detail.

If they did, they would not be asking you to build the system.

Double Learning Experiences

Thus, the deliverables will be double learning experiences:

- learning what the description asks for, and
- learning what you are supposed to learn from what it asks for, …

perhaps not in that order and

perhaps in several iterations.

COURSE DELIVERABLES

Before First Deliverable

You and your group need to self register as an SE463 group at the SE490 or SE463 Learn site and to prepare an abstract of your project by 10 September:

If your SE463 group is also an SE490 team, the self registration you must do at the SE490 Learn site by the same date for SE490 suffices. I will copy that over to SE463's Learn site.

Before First Deliverable

If your SE463 group is not also an SE490 team, your group will self register at SE463's Learn site.

No matter what, look at the course Web site for instructions on how to self register your group.

Before First Deliverable: the Abstract

RE: the abstract:

It's a not-more-than-one-page elevator pitch that tells a potential investor enough detail about the use of the features of your capstone project that E can decide whether to invest.

Many SE490 teams will have an abstract from before.

In that case, use it; if you want to update it, please do so!

Before First Deliverable: the Abstract

RE: the abstract:

If you don’t have an abstract, then make one up.

It doesn’t matter that it will change, because, it’s the basis for the first three deliverables.

Historically, what is learned doing these first deliverables ends up causing changes to the abstract.

First Three Deliverables

The first three deliverables have you describe your capstone system from three different key viewpoints:

- structure (domain model),
- functionality (use-case model), and
- exceptions.

Fourth and Fifth Deliverables

The fourth deliverable is a first draft of the final specification of your system, which is the fifth deliverable.

These are informed by what you learned in the first three deliverables.

Making Learning Safe

- Each of the first three deliverables counts only 1 out of 50 for the whole set of deliverables.
- Even the fourth deliverable counts only 7 out of the 50.
- So you can afford to let each of these first four deliverables be learning experiences towards the last deliverable that counts 40 out of the 50.

We Can Adjust Deliverables to Your Project

- All this said, if you cannot relate a deliverable description to *your* capstone system, please approach your TA and me.
- The TA and I will work with you to find an alternative description or even an alternative deliverable that fits your system and achieves the objectives of the original deliverable.

Is anyone not in a capstone?

Is anyone *not* in some capstone team, either in SE or in another engineering?

If so, →

A Diversion About Smart Ignoramuses

PAST STUDENTS WHO NEARLY FAILED COURSE

Two Teams In S'21

Two teams did total BS for each deliverable.

Always their reasons were that

1. They will figure all this stuff out when they need to (i.e., thinking Agilely).
2. Besides, it's a waste of time that is better spent on solving more immediate problems in the implementation.

Two Teams In S'21, Cont'd

The immediate flaw in Reason #2 is that the SE463 project is neither removing nor adding time from or to the SE490 project.

They are two different courses.

BASIC FACT: No matter what, you have to do SE463's assignments to pass it, and it *is* part of the design core for your SE degree.

Two Teams In S'21, Cont'd

It's only a lucky — or unlucky — accident that the project for SE463 happens to be relevant to your SE490 project.

It could have been some useless, boring piece of dreck that I invented.

Two Teams In S'21, Cont'd

W.r.t. Reason #1, they did not see any reason that they needed to figure this stuff out now.

- I accept that not everyone agrees with the RE field's message that the earlier you figure the requirements out, the faster your implementation will go and the better your software will be.
- So I said, cognizant of the **BASIC FACT**, "Just for a decent grade in this class, humor me!"

Two Teams In S'21, Cont'd

While the BS did not affect the mark received on the 1-mark deliverables, they lost many marks on the last two deliverables, that accounted for almost all of the deliverables' half of the grade.

I instruct the TAs to deduct many points for any ignored comment or request from a previous deliverable.
Do not waste your TA's valuable time!!!!

Two Teams In S'21, Cont'd

Each team ended up with about 35/100 in the project portion of its total grade, ...

when almost every other team got 70/100 or better for the same.

Each team member did fairly well on the final exam.

So, each barely passed the course.

Two Teams In S'21, Cont'd

Their capstone projects came out OK, not superb.

However, with all the usual excitement near the deadline.

So in the end, they passed everything they needed to.

Two Teams In S'21, Cont'd

Note that agile methods *do* work.

But our data say that a method with upfront RE is faster and better.

You may not believe it, especially coming from an old-fart prof who is *so* not with it.

That's fine. (I remember thinking the same way 55 years ago. :-))

Two Teams In S'21, Cont'd

But these teams lost an opportunity to see what happens *if* they figure out requirements up front,

- in a low-risk way,
  (My assignments are not in the real project's schedule.)
- without any pressure on the real project.
  (Since it's something you gotta do for another class.)

Remember

It’s not like I’m asking you to do something totally useless, like my concocted problems.

Eventually, you *will* have to consider the issues covered by the deliverables.

SE463
Software Requirements: Specification & Analysis

Overview and Admin Notes

Fall 2025

Daniel Berry
with Input from Ahmed ElShatshat