SE463
Software Requirements: Specification & Analysis

Overview and Admin Notes

Fall 2024

Daniel Berry

with Input from Ahmed ElShatshat

Table of Contents

- Introduction
- A Brief History Lesson
- Administrative Details
- Course Motivation
- Testimonials
- Course Deliverables
- A Diversion About Smart Ignoramuses
- Past Students Who Nearly Failed Course

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

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

- Joel Rorseth, jerorset ATT uwaterloo DOTT ca
- Yelizaveta Brus, ybrus ATT uwaterloo DOTT ca
- Noble Mathews, ns3mathe ATT uwaterloo DOTT ca
- Shimon, ssarefin ATT uwaterloo DOTT ca
- Mohamed Rouili, mrouili ATT uwaterloo DOTT ca
- Max Zhang, m492zhan 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 page

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

- Lots of details will appear there over the term, especially the lecture slides, supplemental readings, and occasional stuff about the project. This site takes the place of a textbook, which the course does not have.
- Watch for announcements too.

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.

Learn & Piazza

I don’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

Term Project

If you’re in SE490:

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

Term Project

If you’re in CS or another Engr., you will either:

- form a group for a project of your *mutual* choosing,
- join another group as one of its *smart ignoramuses* that thinks out of the group's box, or
- form a group of one, maybe with a smart ignoramus, to write a spec of a major part of your non-SE capstone project.

Term Project

- Your group 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 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")

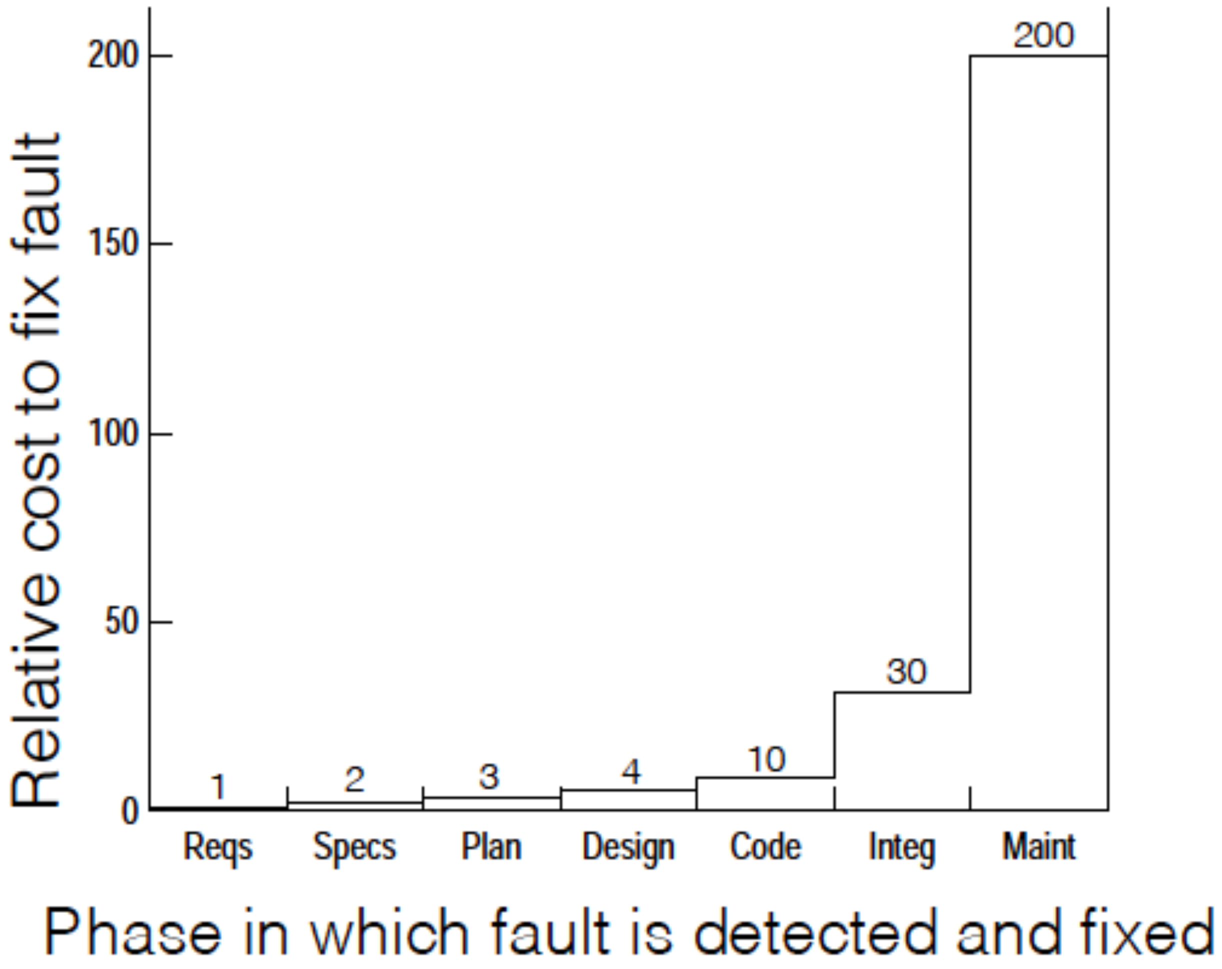
Realities About Software Development Projects

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



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

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!
This leads to ...

Phenomenon A: 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.

Phenomenon A:
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;

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*.

They *invent* them left and right!!

So what difference does it make?

We’re going to have to deal with new requirements after the coding is done anyway?”

Phenomenon B: But But But…

- And some, maybe even many, of these new requirements are not predictable upfront.
- They become apparent only after the system is deployed, and it changes the world so that its own requirements are changed in a totally unpredictable way!
- Think: the Internet over the years!

Phenomenon B is 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?

Phenomena A and B, cont’d

Anytime you have conflicting empirical results in seeming the same situation ...

you are probably overlooking some independent variable (or more than one),

whose values predict the different results.

Two Different Kinds of Requirements

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

- *Scope DeterminiG Requirements (G requirements)* that keep coming and are the ones mentioned in Phenomenon B.
- *Scope DetermineD Requirements (D requirements)* ... Phenomenon A.

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)* ... Phenomenon B.
- *Scope DetermineD Requirements (D requirements)* that are expensive to fix when they are missing, and that should be delivered up front as in Phenomenon A.

Pocket Calculator Example

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

- G requirements: **, *log*
  Adding them is *determininG* 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 is *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!

The Nature of D Requirements

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.

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 G requirements as the scope of your prototype and are heading into design and coding without having fleshed out the G requirements’ D requirements.

You don’t have the time!

The Gift of SE463!

OK.. So what SE463 is going to do is give you the gift of time, time that you *have* to spend to pass SE463, to work out the D requirements for your capstone project!

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 is to write a requirements specification of the selected set of G requirements, the selected scope, of your capstone prototype, in which each of the scope’s D requirements has been worked out 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 has always been the case with the SE1 project, 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!

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 and implement D reqs.

A G req’s D reqs are discovered only during the sprint that implements the G req …

or even later.

Costs

The cost to *fix* this code to handle the D reqs is

at least 10 times

the cost to write the code for the G req with the D reqs already specified, …

so that nothing needs to be *fixed.*

Data Are Clear

The data are clear.

Spend more time figuring out reqs upfront

→

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

Project Managers Don’t Gamble

No manager is willing to buck the trend and to delay the start of implementation to do upfront RE to identify D reqs

- If E bucks the trend, and the project fails, er head will roll.
- If E does the accepted agile method, a failure is regarded as the inevitable occasional shit that happens.

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 this step requires identifying D requirements for the sprint!

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

but who does that??

BOBW (Best of Both Worlds)

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

- try both agile and upfront on the same project
- and compare the outcomes
- without taking away any time from what 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!!!!!

Two Separate Courses

- Note that SE463 and SE490 are separate.
- This gives you the right to ignore in SE490 what you produce in SE463, to throw it out as useless make work that the SE463 prof made you do for a good grade in SE463…
- just as it used to be in the past, and still is in some sections of CS445 and ECE451.

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.

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.

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

First Three Deliverables

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

- 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

A Diversion About Smart Ignoramuses

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 2024

Daniel Berry

with Input from Ahmed ElShatshat