Before We Get Started: A Motivating Overview

CSE 3311
Christoph Csallner
University of Texas at Arlington (UTA)

Several slides in this set are copied almost verbatim (with permission) from:
- Yannis Smaragdakis (University of Athens)
- Eknauth Persaud (CEO of Ayoka)

CSE 3311 vs. CSE 3310

- What do you learn in CSE 3310?

- How is CSE 3311 different?

Selected Student Feedback

- From my earlier editions of CSE 3311
- Sources (all anonymous):
  - UTA's Student Feedback Survey (SFS)
  - Separate anonymous surveys (Survey Monkey)

- Feedback is not comprehensive
  - Some students don't participate in SFS
  - Following is a subset of the feedback I did get

- Format: Quote followed by my comments

Project Uses Iterative Development

- *"By doing a hand on project, I believe i learned a lot more."*
- *"Separating work in iteration was helpful as it forced teams to make progress."*

- Iterative software development in many places, incl.:
  - Microsoft, Facebook, Google, Amazon, Netflix, startups, ..

Programming vs. SE

- *"The class title is Object Oriented but it **does not actually teach you any programming** [..]"*

- CSE 3311 = "Object-Oriented Software Engineering"
  - Programming is key, but not the only part of SE
  - **Pure programming does not scale well to large software development projects**
  - CSE 3311 = Non-programming techniques (+ some programming techniques) to let you succeed at programming in larger projects
- Next slide: CSE 3311 description from UTA's catalog
  - With formatting / highlighting by me

Official CSE 3311 Description

- *"Study of an agile unified methodology and its application to object-oriented software development. Topics include **requirements** acquisition, **use case** derivation, **modeling** and design of interaction behavior and state behavior, introduction to **design** patterns, derivation of design class **diagrams**, **implementation considerations**, and deployment.*
- *Team project.*
- *[..]"*

Heavy Emphasis on Team Project

- *"[T]he class had a good balance; however, the **project was much more intense when compared to senior design** which is not a bad thing."*

- You have to **actively explore and practice software engineering techniques** to truly understand and eventually master them
- The more you practice, the more you learn
- Team project should set you apart (in a good way)
  - Resume highlight hiring manager wants to hear more about

Object-Oriented (OO)

- *"The class title is Object Oriented but it **does not actually teach** [..] **that much on Object**"*

- CSE 3311: Systematically model world (requirements, etc.) in an OO style and convert model to OO code
  - Model each "group of world entities" as a set
  - Model each "relationship between groups" as relation on sets
    - » Previous two modelling steps are useful by themselves
  - Set = model of OO class. Set element: class instance
  - Convert each set to a class, each relation to an instance field
    - » Natural / easy transition between OO model and OO code

OO: Project Language

- *"The class title is Object Oriented but it **does not actually teach** [..] **that much on Object**"*

- In past I let students pick almost any language & coding style for team project
- To address above student feedback, **your team project should use OO, e.g.:**
  - Instead of arbitrary JavaScript, code in TypeScript in a class-based object-oriented programming style
  - Python: Use new-style classes (or Python 3+)
  - C#, Java, ..

Team Composition

- *“Be able to choose your own teams.”*
- Goal of project is to simulate real-world setting in a safe, friendly, supportive environment to optimize learning
- Being able to succeed with a variety of people is good
  - Integrate & leverage individual backgrounds in your team
  - Industry will force you to work with wider variety than here

Each Team Defines Own Project

- *"There was little to no guidance on the actual project as each group would be different. We would be assigned our group and left to pick our project as well as tools and languages [..]"*

- Simulate a startup environment
- You define and justify your own project
  - Come up with project **you** deeply care about
  - Define it in a way that will look good in a job interview
  - Ideally you start the next Google / Facebook in this class :-)

Team-specific Feedback

- *“He encouraged our group to take certain steps towards improving our project and pulling out it's full potential.”*

- My goal is to help each team in reaching their goals and producing the best software possible within the class constraints
  - I will give you custom, team-specific feedback after each project presentation
  - You may get similar feedback if you discuss your project plans & demo your project prototypes to your friends / family

Learning Outside Classroom

- *"It does not help that you have to create an app and have the presentations and all of this paperwork for an app when the course does not teach you how to create an app."*

- CSE 3311 covers several SE techniques that can help you succeed in large projects
- But one 3-credit course cannot cover everything you need to succeed in a large project
  - Data structures, programming basics, frameworks, etc.

Project Grading

- *"If the level of the project is not at a senior design level project, you will not receive a good grade in the class [..]"*

- Goal is to learn how to build actual software
  - Tip: Analyze team-specific feedback from peers, TA, and me
- Individual contribution to project is key to grade
  - High individual effort usually shows in team presentations
    - » I take detailed notes during presentations on who says what
  - Detailed peer evaluations
  - Make project better than sum of individual contributions

In-class Team Activities

- *"[..] definitely helped in applying the material we went over in class. By incorporating it into the context of our project, we were able to solidify out knowledge."*
- *"[..] although annoying to do,* ***convey the material much better and facilitate learning the material*** *now instead of later."*
- *"[..] where we practiced making the diagrams and stuff with our teammates helped a lot"*

- In-class exercises popular in all prior semesters
- Hard to make up by watching recordings after class

Quizzes

- *“I found it challenging to determine what material would be covered on each quiz.”*

- Anything discussed in class and via homework since the last quiz
  - Includes students’ optional tool presentations

Resolving Issues

- *"[..]* ***responsive to his e-mails*** *and made himself available as much as possible"*

- I intend to keep doing that
- But many grading questions handled by TA
  - Written deliverables, homework, quizzes
  - Contact me if TA cannot resolve issue
- Don't be afraid of seeing us during our office hours
- "Office phone" on syllabus is department front desk
  - Many CSE professors no longer have "own" phone number

Asking Questions in Class

- *"Some of the material was a bit hard to grasp, but, if you would ask a question, Prof. Csallner was very good at explaining it."*

- I love questions, ask me anything
  - As long as it is related to software engineering :-)

- Asking questions will up your class participation grade
  - Answering questions will also up this grade

IN-CLASS EXERCISE: SOFTWARE ENGINEERING CHALLENGES

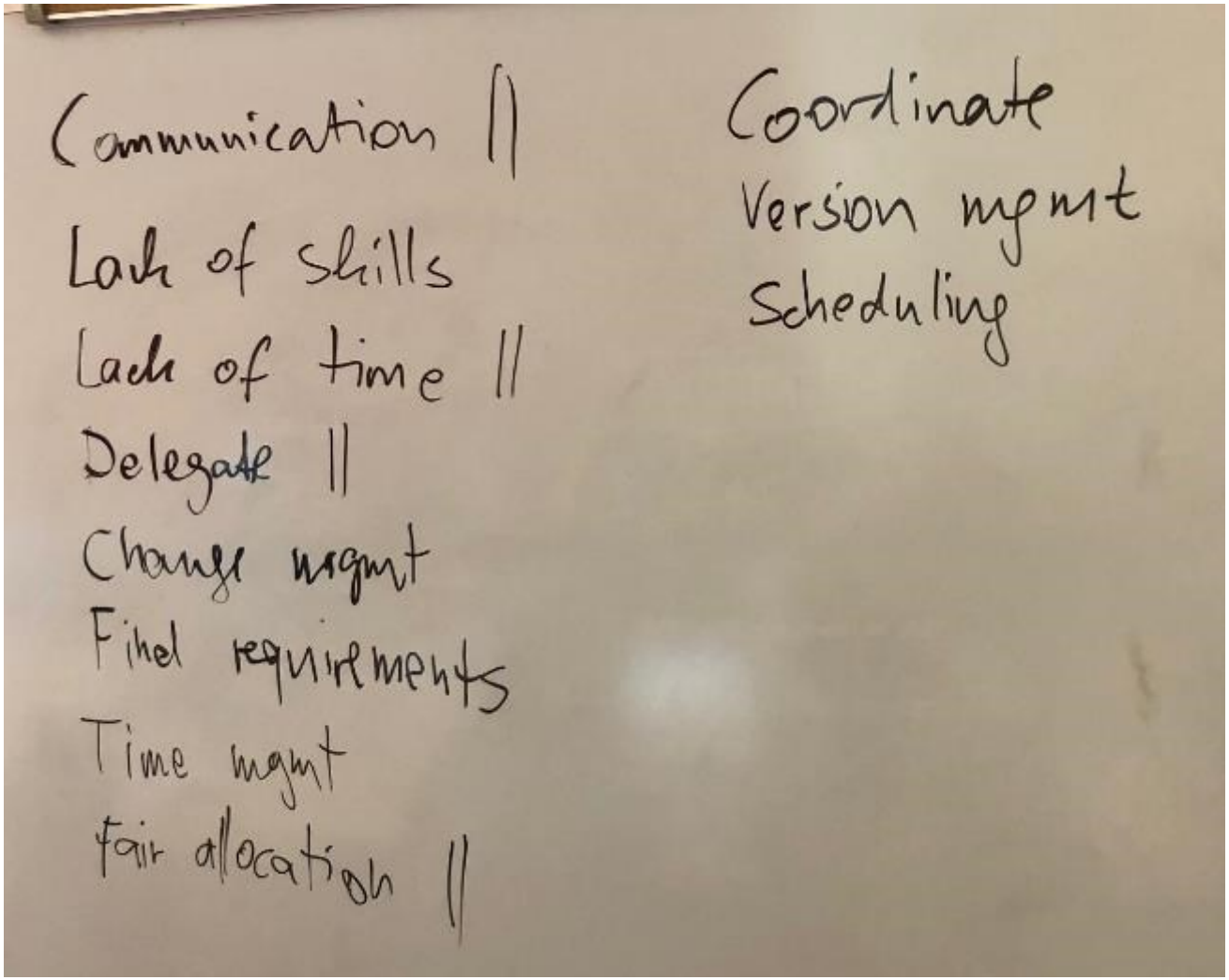
Software Engineering Challenges

- Get together with your team mates
  - Or group of three
- From your own experience, what are the **three biggest challenges** in actual software engineering?
  - From your internships, jobs, other classes, etc.
  - Things that are both important for project success and hard to get right
- Be prepared to present your results

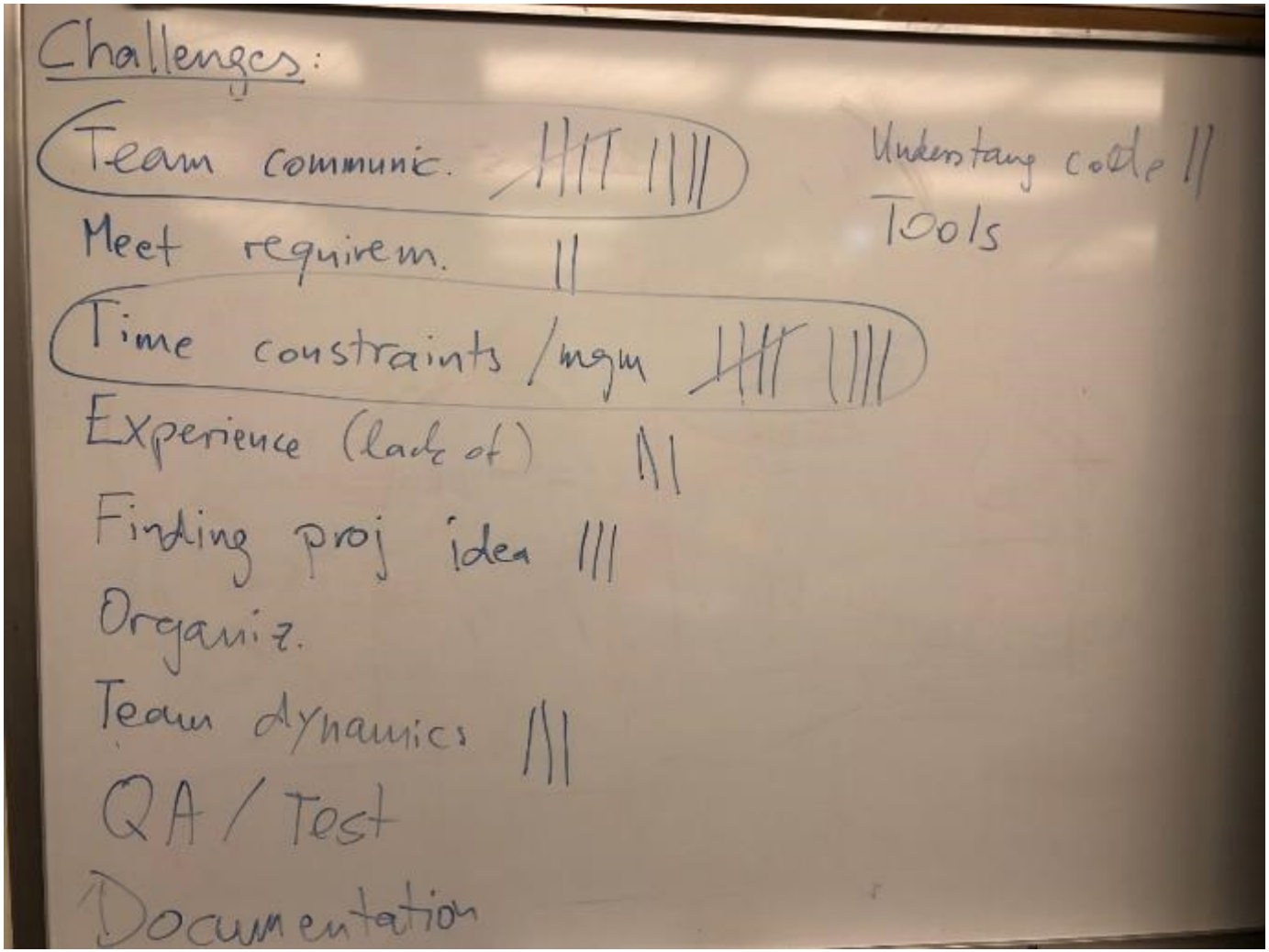
SE Challenges (Updated)

- What are the two biggest challenges in actual software engineering, **from your own experience** (internship / job / class)?
  - Things that are both important for project success and hard to get right
- **I will call on you to speak but will give you three minutes to think about it**
  - Post your answers into the class chat
  - Upvote classmates' post if it captures the essence of your answer

Results: Three biggest challenges in software engineering you have experienced (1/29/2018)



Results: Three biggest challenges in software engineering you have experienced (8/27/2018)



IN-CLASS EXERCISE: SOFTWARE ENGINEERING BEST PRACTICES

Best Practices

- Get back together with your team
- In your experience, what are the **best three things to do** for successful software engineering?
  - E.g., use a certain process, tool, technique, etc.
  - Maybe things you have observed a very senior engineer do or things that gave your own projects a big boost
- Be prepared to present your results

Best Practices (Updated)

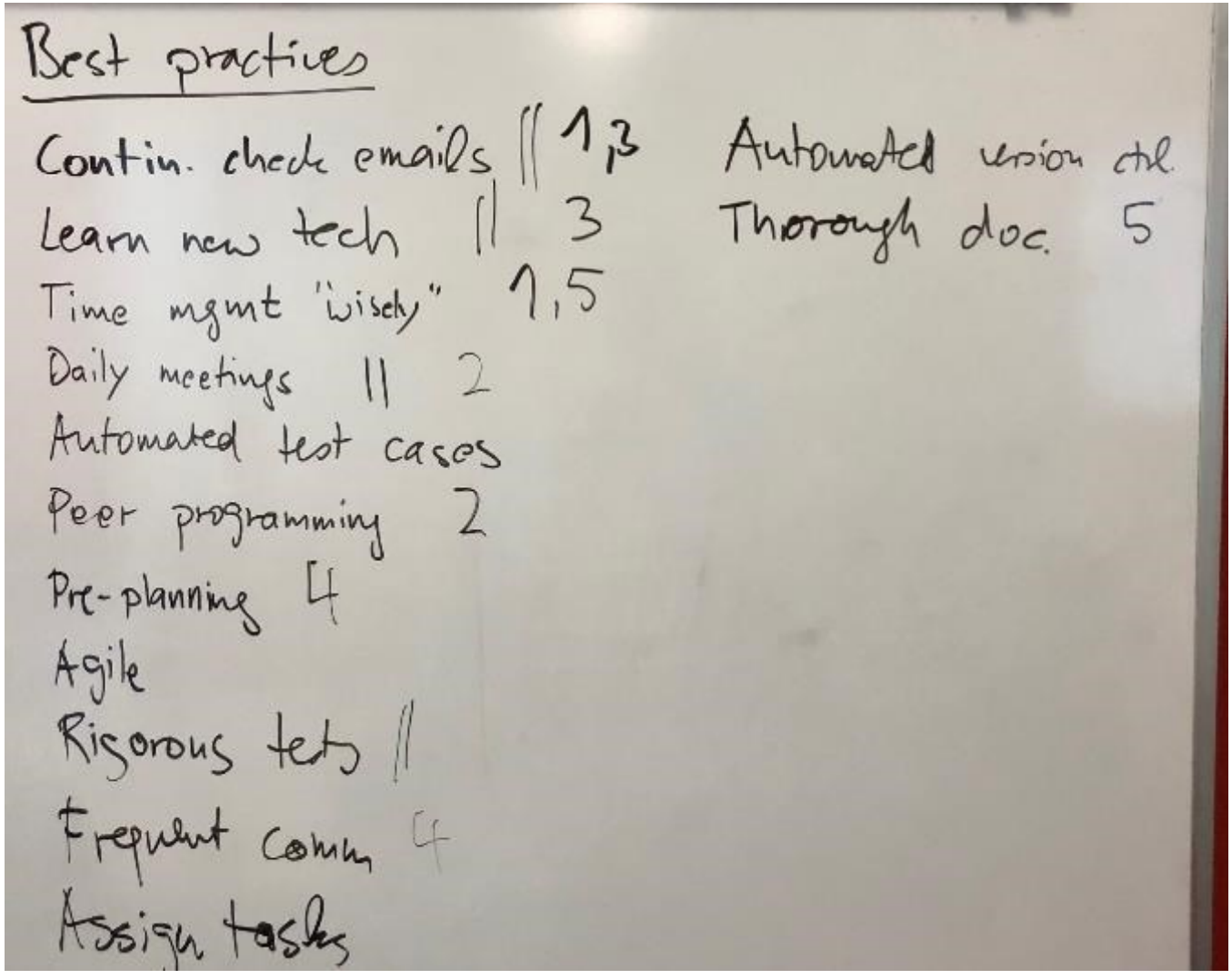
- What are the **2 best things to do** for successful software engineering in your experience?
  - E.g., use a certain process, tool, technique, etc.
  - Maybe things you have observed a very senior engineer do or things that gave your own projects a big boost
- **I will call on you to speak but will give you three minutes to think about it**
  - Post your answers into the class chat
  - Upvote classmates' post if it captures the essence of your answer

IN-CLASS EXERCISE: BEST PRACTICES IN CLASS PROJECT

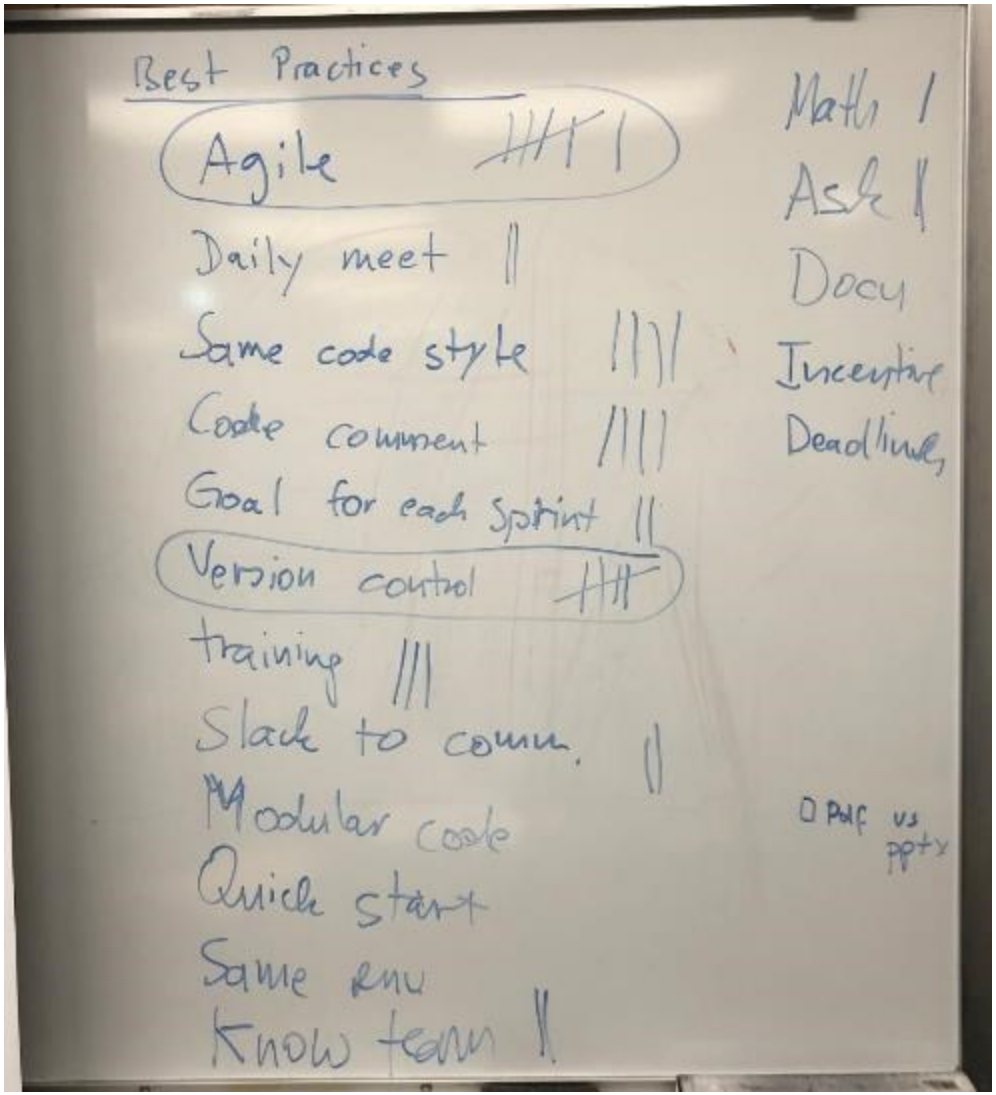
Best Practices in Class Project

- Get back together with your team
- We have heard about several best practices.
- Of these: Which top 2 things do **you personally plan to incorporate** in your CSE 3311 class project?
- Be prepared to present your results

Results: Best Practices (1/29/2018)



Results: Best practices (8/27/2018)



Why Study Software Engineering? (1/2)

- Software is fascinating
- Engineering software is a creative activity
- Software is a key component of many other things
  - Which industry or aspect of life does not need software?
- High pace of innovation
  - Internet search engines, smart phones, social media, big data, deep learning, autonomous driving, blockchain, AI, etc.

Why Study Software Engineering? (2/2)

- Software engineers earn a lot of money
  - Even as an entry-level employee
  - Even when not biggest experts on latest trends (AI, ..)
  - While working in a comfortable (home-) office environment
- Employers want to hire more software engineers
- Easy to start your own company while a student
  - Don’t need lots of capital for factories, etc.
  - Successful examples: Microsoft, Google, Facebook

Main SE Tasks & Artifacts

- What is the problem to be solved?
  - Describe domain
  - Describe requirements
- How should we solve the problem?
  - High-level: architecture
  - Low-level: design
- Actually solve the problem
  - Write code
- Check deliverables
  - Review, analysis, test

But how to do this exactly?

- Describe domain — How?
- Describe requirements — How?

- High-level: architecture — How?
- Low-level: design — How?

- Write code — How?

- Review, analysis, test — How?

- How to move between steps? Who does what?
  - Process, management

In each step: How?

- Which aspects are important and should be described?
- Which aspects can be ignored?

- Which notation or language to use?
- Which tool and format to use?

- Can we generate some aspect from a previous step?
- Can we reuse existing solutions from some standard library or repository?

Software Engineering

- Provides answers to these questions

"Software Errors Cost U.S. Economy $59.5 Billion Annually" [NIST 2002]

- "Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually [..]
- At the national level, over half of the costs are borne by software users and the remainder by software developers/vendors."
  - http://www.nist.gov/public_affairs/releases/n02-10.htm
- Surveyed 2 areas, projected them to entire economy
  - 10 CAD etc. vendors + 179 users, i.e., automotive & aerospace companies
  - 4 financial service developers + 98 banks ("providers")

Interesting Details in NIST Report: E.g., the Software Developers Say

- They find >50% of bugs in late phases
  - Integration, beta test, after release
- Average cost of fixing a bug varies
  - Based on where it is introduced & found
  - E.g., the financial service developers:
  - 18.7 hours: requirements → after release
  - 1.2 hours: requirements → requirements

Another Example Detail: From the 98 Financial Service Providers

- On average 3,970 employees

- X% said it suffered Y from software errors:
  - Delayed product or service introduction: 10%
  - Lost data: “several”
  - Lost reputation: 15%
  - Lost marked share: 5%

» On page 7-22

NIST 2002, continued:

- “The study also found that, although all errors cannot be removed, more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure that enables **earlier and more effective identification and removal of software defects**.
- These are the savings associated with finding an increased percentage (but not 100 percent) of errors closer to the development stages in which they are introduced. Currently, over half of all errors are not found until "downstream" in the development process or during post-sale software use.”

(In-) Famous Software Failures

- MS Excel 2007 showing the result of (77.1 * 850)
  - Not as 65,535 but as 100,000
  - http://blogs.msdn.com/excel/archive/2007/09/25/calculation-issue-update.aspx
- CryoSat satellite destroyed on launch (2005)
  - "[C]aused by faults in the programming of the rocket, which had not been detected in simulations"
  - Satellite cost some EUR 70 million
  - http://en.wikipedia.org/wiki/CryoSat#Launch_failure
- Ariane 5 rocket destroyed on launch (1996)
  - "[A] 64 bit floating point number [..] was converted to a 16 bit signed integer"
  - Rocket + cargo cost some $500 millon
  - http://www.ima.umn.edu/~arnold/disasters/ariane.html

Ariane 5 Investigation with Recommendations

- “The extensive reviews and tests carried out during the Ariane 5 Development Programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure”
- R1: “[..] no software function should run during flight unless it is needed.”
- R2: “[..] **A high test coverage has to be obtained**.”
- R9: “**Include external (to the project) participants when reviewing specifications, code and justification documents.** [..]”
- http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Ariane/Esa/ariane5/COPY/ariane5rep.html

Software Failures Have Killed People

- Patriot Missile failure kills 28 soldiers in 1991:
  - "[..] failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks [..]" http://www.ima.umn.edu/~arnold/disasters/patriot.html
- Many more!
  - Security bugs that enable data leaks, etc.
  - http://catless.ncl.ac.uk/Risks/
  - http://www5.informatik.tu-muenchen.de/~huckle/bugse.html
- More than crashes and exploits:
  - Additional effort to "optimize" sub-optimal software
  - Software that is hard to use or works slowly slows down its users: costs time and money

IN-CLASS EXERCISE: SOFTWARE ENGINEERING WORST PRACTICES

Software Engineering Worst Practices

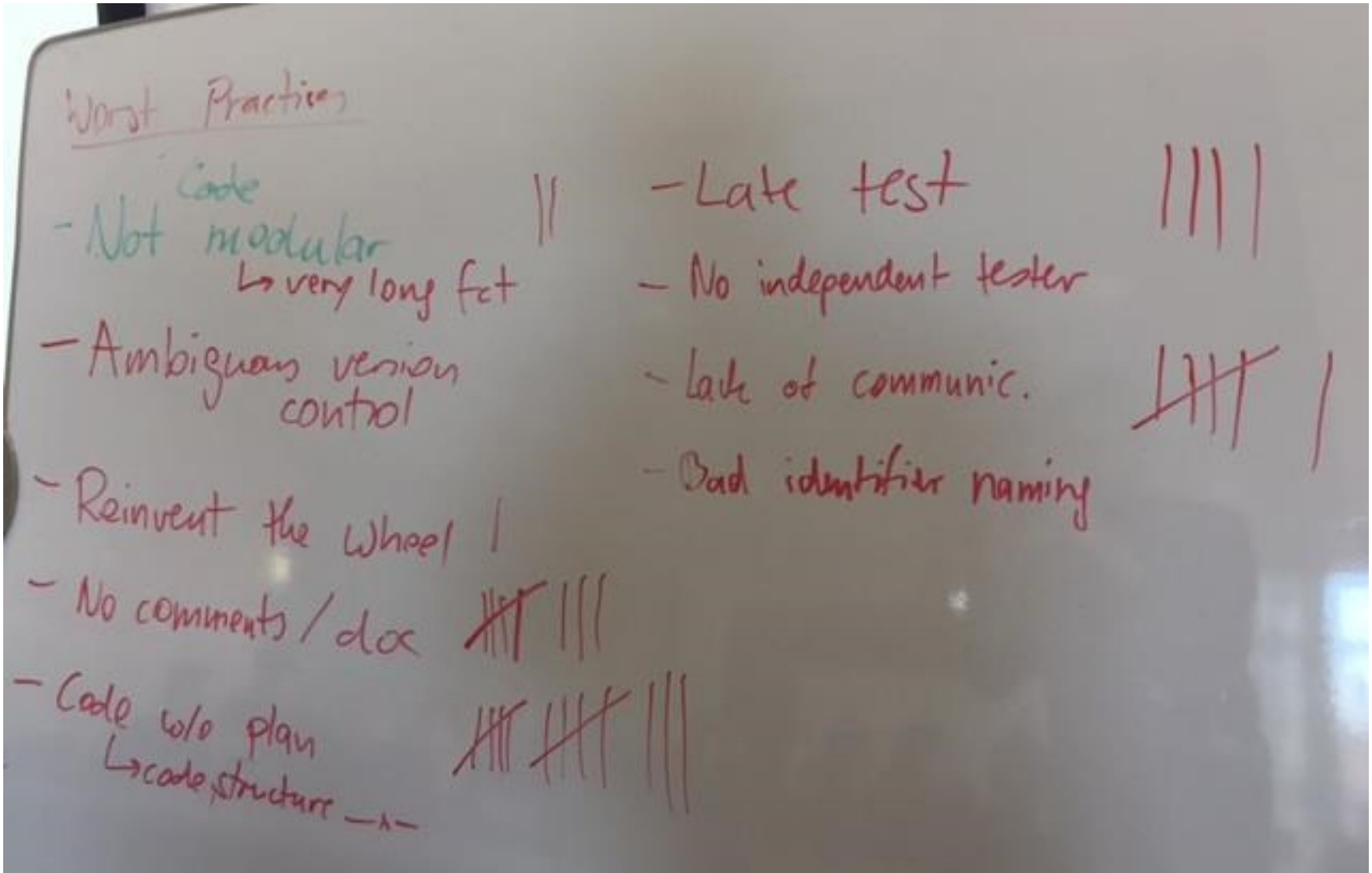
- Get back together with your team
- In your personal experience, what are the **worst three things** to do in SE?
  - Things that lead to low productivity & project failure
  - E.g., use a certain process, tool, technique, etc.
  - Things you have observed people doing when being counter-productive to project success
- Be prepared to present your results

IN-CLASS EXERCISE: WORST PRACTICES TO AVOID

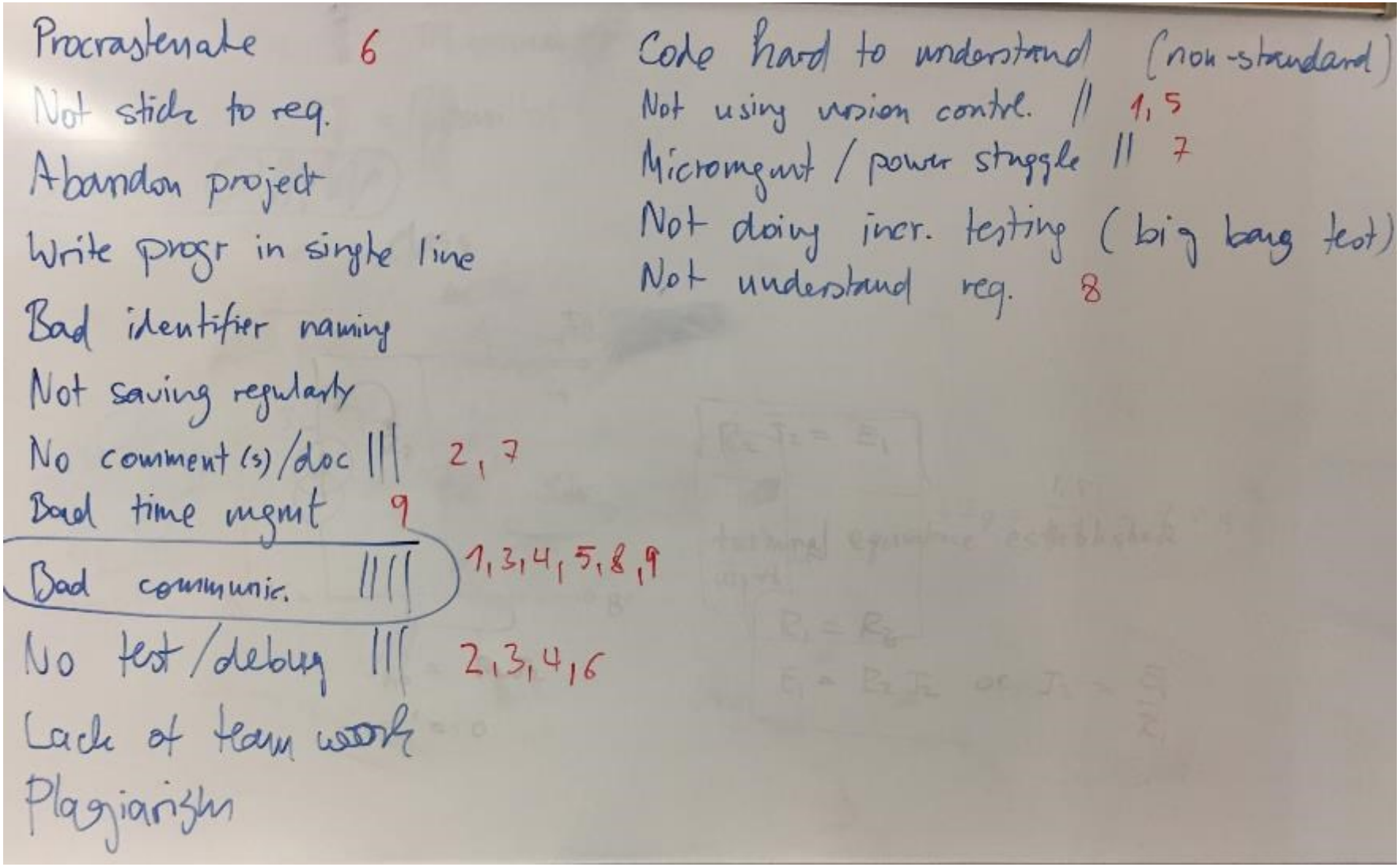
Worst Practices to Avoid

- Get back together with your team
- We have heard about several worst practices.
- Of these: Which top 2 things do **you personally plan to make sure they are not happening** in your CSE 3311 class project?
- Be prepared to present your results

Results: Top 2 things you personally plan to make sure they are not happening in your class project (1/24/2017)



Results: Top 2 things you personally plan to make sure they are not happening in your class project (9/6/2017)



What Do People in the Trenches Think?
(about process, SE practices, etc.)

- Yannis asked people who write software for a living
  - –in a group environment, large software company (10+ years ago)
  - –mostly unedited quotes
- Yannis worked with two of these people before
- For emphasis, I made some words **bold**

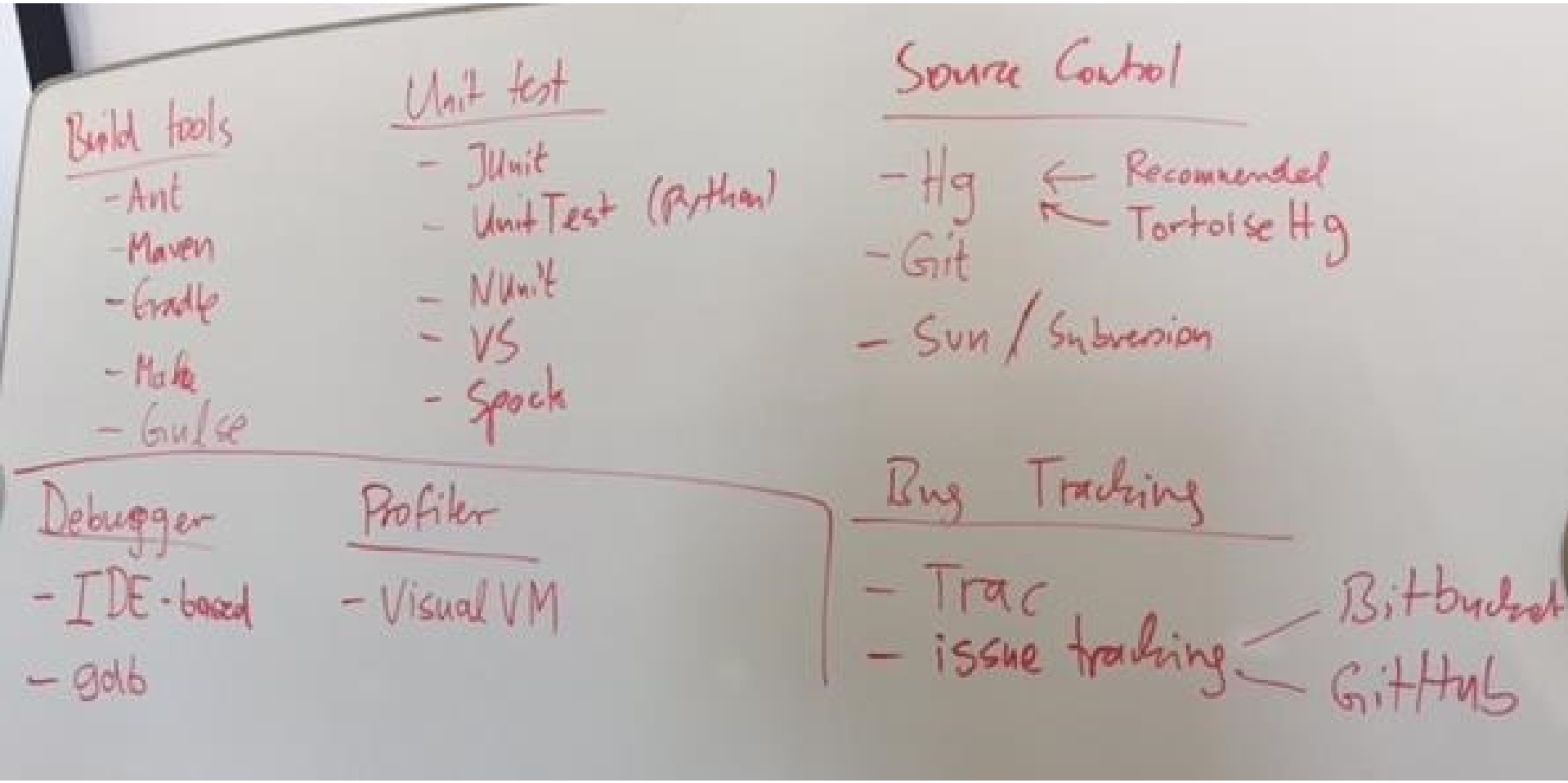
Preliminaries, “Minimum Bar”

- *No matter what size project you're on, you must have a source control system, a bug tracking system, unit tests that are easy to run, and a rolling build system that ensures checkins don't break the build. If any of those are missing, then just fold up the tent and go home.*
- *Get your dev tools, source control, build processes, debugger and profiler setup, automated test tools, basic performance tests, etc. in place at the beginning.*

Tools Are Important in Practice

- *No matter what size project you're on, you must have a **source control system**, a **bug tracking system**, **unit tests** that are easy to run, and a rolling **build system** that ensures checkins don't break the build. If any of those are missing, then just fold up the tent and go home.*
- *Get your dev tools, source control, build processes, **debugger** and **profiler** setup, **automated test tools**, basic performance tests, etc. in place at the beginning.*
- **Tools:** Great topics for extra credit presentations

Tools You Can Use in Your Class Project (1/24/2017)



Scaffolding

- ***No one invests enough in good build systems and tools.*** *It surprises me at [company] how much effort we put into the product itself, but then the build system that actually produces the product is always duct tape and bailing wire. In my limited experience, it's a universal nightmare. There's always an awful mix of makefiles and perl scripts and [proprietary tool] and all kinds of crazy unreliable crap, and if you're like me and actually care about this part of the process and how it is sucking away everyone's productivity, then it can drive you nuts.*

Conventions

- *Have an "**architectural checklist**" for key issues such as concurrency, re-entrancy, fault-tolerance (that might be too advanced), asynchronous I/O that may block indefinitely (much more realistic), handling of bad input (including weird cases like an object created by the wrong instance of the factory), low-memory conditions, etc..*

  *Also decide on key **conventions** such as (in unmanaged code) caller or callee initializing output parameters, silly things like parameter order (outputs last?), etc. I've found that in our code for [latest project], there's an amazing amount of boiler-plate for this sort of stuff (including the architectural aspects)*

Process

- *If I were in charge of a team, I would **emphasize rapid prototyping**. The less "adventurous" a project is, the less this is needed, but unless there's a **really complete spec**, I'd say a prototype pays for itself very quickly. One can also use the prototype to build up an initial test suite, although it's probably best to keep that away from the devs when they start the real implementation, or else they'll just code against it.*
- *"Scrum" is a very popular method lately; I've only had a taste of it, but short milestones, good prioritization, and high-touch interactions among team members seem like good stuff for small teams.*
- Scrum is an iterative process model

On Macro-Processes

- *I've noticed that when people are given some procedure to follow (e.g. fill out a "security threat model" form), they're happy to do that, as they'll be able to make concrete progress for that hour or whatever of their day. It's very easy to fill up the hours of an employee's day with this sort of stuff, and it will get prioritized, as it's far easier for a manager to say "you didn't fill in form X", than to say "you're coding too slowly".*

Process and Project Size

- *Large teams working on large code bases with more complexity have got to have a lot of 'process' in place.* ***Good large teams will choose the minimal processes for the maximum benefit.***

- *But in the end, if you are personally not big on process, you will probably be happier on a smaller team/product.*

Process and Design (1/2)

- *We definitely needed more time for design up front. We had **plenty of time (months!), but squandered it** in "working groups" that met many times and **agonized over some basic points**.*

- *Then we were distracted by another project management dumped on us, and then we were given a week to fill in the remaining 95% of the details.*

Process and Design (2/2)

- *A friend of mine is on another team at [company] that is having a "quality crisis"; I guess in their last milestones they had a regression rate of 30% per checkin (I have no idea about industry stats overall, but that doesn't seem too extremely high to me; maybe 15% is more reasonable?) and so the "solution" is that one of their architects (whom my friend thinks is a useless idiot) is trying to impose a bunch of metrics (e.g. cyclomatic complexity, comment percentage per lines of code, ...) which will be enforced on checkins to improve the code base. My friend sent me a copy of the doc this guy sent, it was basically phrased as "to fix our problems, we'll do this" with metrics apparently picked out of the air with no sound justification and no plans to evaluate if the new process is working.*

Metrics

- *A friend of mine is on another team at [company] that is having a "quality crisis"; I guess in their last milestones they had a regression rate of 30% per checkin (I have no idea about industry stats overall, but that doesn't seem too extremely high to me; maybe 15% is more reasonable?) and so the "solution" is that one of their architects (whom my friend thinks is a useless idiot) is trying to* ***impose a bunch of metrics*** *(e.g. cyclomatic complexity, comment percentage per lines of code, ...) which will be enforced on checkins to improve the code base. My friend sent me a copy of the doc this guy sent, it was basically phrased as "to fix our problems, we'll do this" with metrics apparently picked out of the air with no sound justification and no plans to evaluate if the new process is working.*
- Metrics are often not a good management technique
  - Most metrics can and will be circumvented

Testing: Block Coverage

- *One process I really like is test-driven development. I wrote a blog entry about it a while back [...]*
  It also mentions "code coverage" as a useful metric; it is the only metric I know that is nearly universal; most teams at [company] seem to measure it, with the idea that ***if you've got less than ~80% block coverage from your tests, then you are doing something wrong****.*

Automated Checks

- *[T]he best approach to software engineering is to find a way of "doing everything twice", sorta like double ledger bookkeeping. In some cases, one relies on the type system. In others, on asserts. In others, on unit tests.* ***I'd say this is a key thing to focus on -- is there some way the machine can check absolutely every aspect of a program against another aspect?*** *Check as much as possible as early as possible, using compile-time asserts and complex types that may compile to almost nothing. (I found a ton of bugs a couple of months ago when I changed a typedef that was used in subtly different ways into a class template with four incompatible instantiations that required explicit conversions between them.)* ***Have a build that runs the automated tests extremely slowly, but does massive run-time consistency checks on all data structures.***

Dedicated Testers

- [This is the same person who said 3dev + 3test + 1PM = 2dev] *One problem we've had with separate testers is that there's zero accountability for them. They can just say "it's tested" and everyone (well, management) believes them. I suggested two potential levels of accountability... The first is to allow devs to insert at key points in the code some sort of macro/whatever to say "this codepath should be tested". ... [T]ests should be required to hit *all* of them. Our testers rely on code coverage numbers, and because of various little error-handling macros and the like, numbers like 70% are considered acceptable. Alas, no-one knows if major cases are being missed. The even higher level of accountability would be to allow devs to add macros that introduce deliberate bugs in a special build.*
- [different person, not responding to previous] *Trust me, I speak from experience. My testers are finding plenty of bugs.*

From Ayoka CEO Eknauth Persaud during 2011 UTA CSE Industrial Advisory Board meeting (used with permission)

CLOSER TO HOME

Ayoka

- Located close to UTA campus
- “made in U.S.A. software services”
- In the last 5 years hired 40+ UTA alumni
  - Computer Science
  - Business, Management, Marketing
- Application development
  - Java, .Net, C#, Silverlight
  - Ruby on Rails, Python, Php
  - Mobile Apps – Apple, Android, Blackberry
- Enterprise systems, systems integration, ...

UTA: Opportunities for Improvement [verbatim + my comments in black]

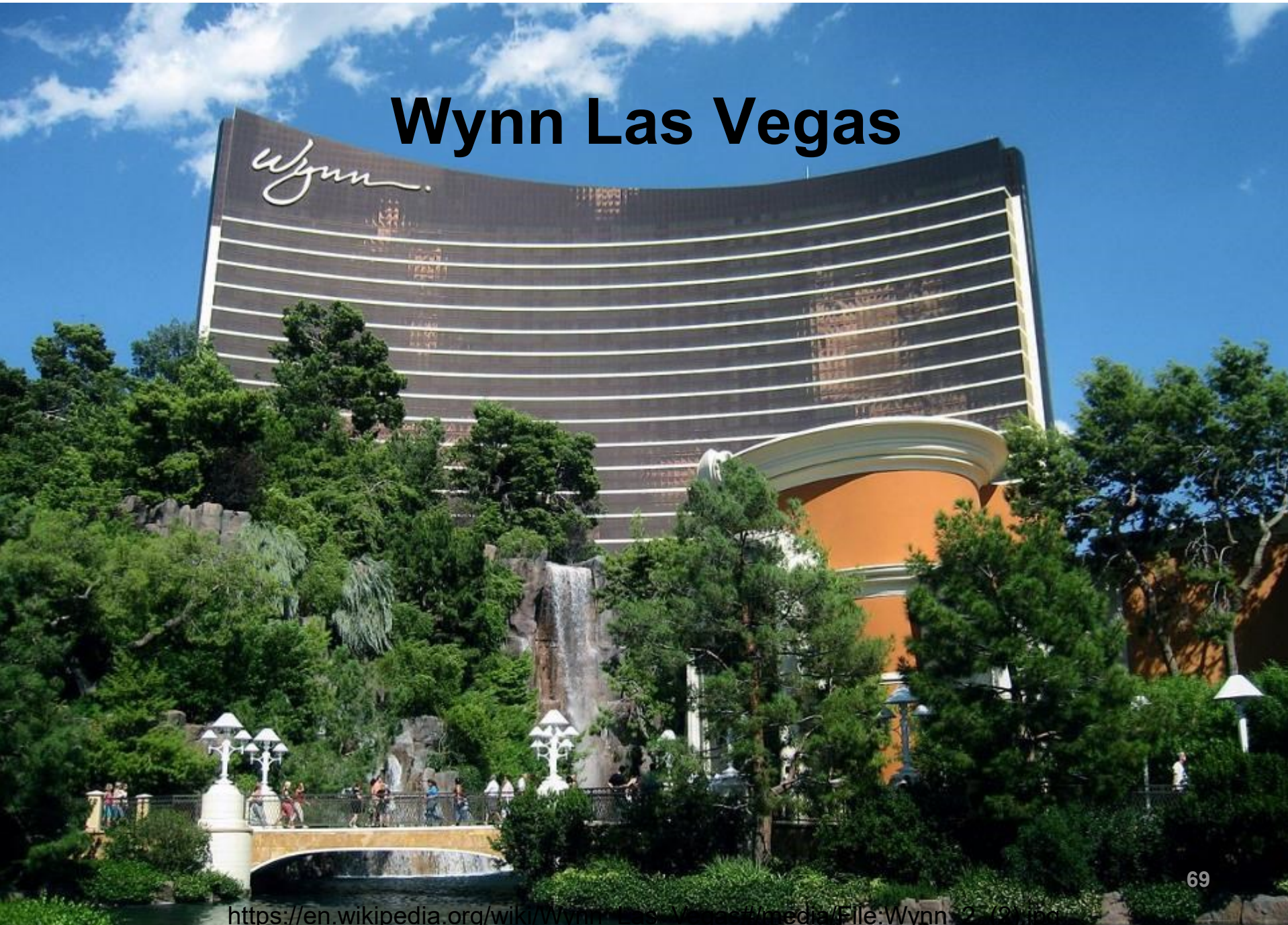
- *Coursework*
  - *Teach modern languages, inspect code*
    ➔ Java, C#, Python, etc. + regular code reviews
  - *Learn web architecture, read + write to databases*
    ➔ Often part of class project
  - *Understand differences of embedded vs. web*
  - *Teamwork: code repository, commenting, etc.*
    ➔ Encourage you to use Bitbucket / Github
  - *Model class projects to real-world*

UTA: Opportunities for Improvement (continued)

- *Behavior*
  - *Work ethic – ability to deal with pressure*
    ➔ Iterative development model: Frequently re-prioritize & move less pressing tasks to next iteration
  - *Timeliness, dress properly, social grace, ethics*
  - *Business etiquette / adult behavior*

SOFTWARE ENGINEERING VS. OTHER KINDS OF ENGINEERING

Wynn Las Vegas



https://en.wikipedia.org/wiki/Wynn_Las_Vegas#/media/File:Wynn_2_(2).jpg

Reason 1: Complexity Per Construction-USD

- For large projects: Estimate USD / LOC
- One estimate for large projects: USD 100 / LOC
- 10 MLOC ➔ 1B USD
  - Roughly the size of Linux kernel (incl. drivers)
- Software:
  - **Almost every part is unique / custom design**
  - **No expensive construction materials (marble, etc.)**
  - **Almost all construction cost goes into complexity**
- Which project is easier to staff / plan / control?
  - Wynn Las Vegas (USD 2.7B project) or Linux kernel?

Reason 2: “It’s never too late to change your mind”

- Half-way through project construction you make a fundamental change request
  - In your software project
  - To your house builder
- What kind of responses do you expect in the above two projects?

- “Software is easy to change”
  - “Just fix a couple of lines”
  - “No big deal”

Reason 3: Legal Framework
Quotes from online discussion:

- I've seen a large number of houses that are just shacks. Houses that can, and often do, simply collapse in heavy weather.
- To get away with building a shack instead of a house, you have to build your shack where there are no housing codes or standards. Or, you have to evade inspection by claiming you don't "live" there. Or you have to be heavily armed so that the inspectors don't bother you.
- Also relevant is that many of us live in countries where **housing is strictly regulated**, and poor quality is considered unacceptable even if both builder and purchaser want to cut corners. In the **absence of that enforced regulation**, housing runs the whole range including ghastly deathtraps; that is what we see in the field of software.

http://stackoverflow.com/questions/347584/why-is-software-quality-so-problematic/347634#347634

Legal Framework (continued)
Quote from online discussion:

- Most **consumer products have warranties** (expressed or implied). Some even have **applicable standards for safety**.
- Consumer shrink-wrapped software **specifically avoids offering a warranty of any kind**. It's a sad, shabby business. To avoid warranty claims, they don't let you purchase software; you merely license it, or purchase a right to use. **Read your EULA (End-User License Agreement). There's no warranty.**
- In-house software, developed by large IT organizations, has no warranty of any kind either.

http://stackoverflow.com/questions/347584/why-is-software-quality-so-problematic/347634#347634

Existing Pockets of Software Regulation

- Gambling, e.g.: Nevada Gaming Commission
  - "The Commission consists of five members appointed by the Governor to four-year terms [..]"
  - "The Commission is also charged with the responsibility of adopting regulations to implement and enforce the State laws governing gaming. "
  - "The Commission is the final authority on licensing matters, having the ability to approve, restrict, limit, condition, deny, revoke or suspend any gaming license"
  - https://gaming.nv.gov/index.aspx?page=3
- Airplanes: Federal Aviation Administration (FAA)
- Medical devices: Food and Drug Administration (FDA)

Reason 4: Young Discipline

Quote from online discussion:

- It used to be considered a truism that when IBM wrote OS/360 it was, *at that point [1960s]*, the most logically complex system ever developed by humans.
- Since then we've developed techniques of handling more and more complex systems. Our languages, APIs and tools have added layers of conceptualization and abstraction not dreamed of when OS/360 was hand-cranked together. The trouble is **the complexity of what we are trying to achieve has increased** in step - or maybe even a little more than.
- Software development is in its infancy. **While engineers have had hundreds, even thousands of years to perfect their craft, software engineers have had a handful of decades**.

http://stackoverflow.com/questions/347584/why-is-software-quality-so-problematic/347863#347863

Fewer Parts = Fewer Things Can Fail
Quote from online discussion:

- One major reason is that for the most part, software "engineers" aren't really trained as engineers**. One of the most important principles in engineering is to keep designs as simple as possible** in order to maximize reliability (fewer parts = fewer things that can fail).
- Most software developers [..] are not just unaware of the KISS principle, but also actively committed to making their software as complicated as possible. **Programmers by their nature enjoy working with complexity, so much so that they tend to add it if it isn't there already**. This leads to buggy software.

http://stackoverflow.com/questions/347584/why-is-software-quality-so-problematic/347601#347601