What is a good SW project?

CS350 Introduction to Software Engineering

Shin Yoo

“When you can measure what you are speaking about, and express it in numbers, you know something about it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarely, in your thoughts advanced to the stage of science.”

Lord Kelvin

The Need to Measure Things

- Otherwise how do we know whether we are doing well or not?
- “One of the problems that is central to the software production process is <u>*to identify the nature of progress and to find some way of measuring it*</u>. Only one thing seems to be clear just now. It is that program construction is not always a simple progression in which each act of assembly represents a distinct forward step and that the final product can be described simply as the sum of many sub-assemblies.”, *Fraser - Software Engineering (1968) pg 51*

Measuring the abstract is not easy

- Lord Kelvin had it easy - temperature is a natural phenomenon, and you can define a metric based on itself
- How do you measure abstract qualities such as:
  - how good your software architecture design is
  - how good your code quality is
  - how well you have tested your program
  - how productive your development team is

Example: Measuring Good Design is Hard

(due to the inherent limitations in quantitative measures)

- A recommended high level principle in architecture design is that software modules should have high cohesion and low coupling
  - Cohesion: a single module should be about one thing and one thing only. For example, if your class is named `BankAccount`, it should contain minimal information about the holder of the account, i.e., it is not ideal for the class to have a field named `accountHolderHomeAddress`.
  - Coupling: two modules should share the minimum amount of information/ connections between them. Otherwise, changing one of them will require unnecessary changes in the other.

Example: Measuring Good Design

(due to the inherent limitations in quantitative measures)

- How would you design a quantitative metric that measures cohesion of a class?
    - count the number of attributes that are shared between its methods
    - count the pairs of methods that access the same attribute of another class
    - compute the ratio between shared attributes and the total attributes they access
    - …

Example: Measuring Good Design

(due to the inherent limitations in quantitative measures)

- A single change in source code should either increase of decrease the degree of cohesion (in the abstract sense)
- Take two cohesion metrics, X and Y, and change the source code so that X is constantly improved - what would happen Y?
- The quantisation itself can have unexpected impact.

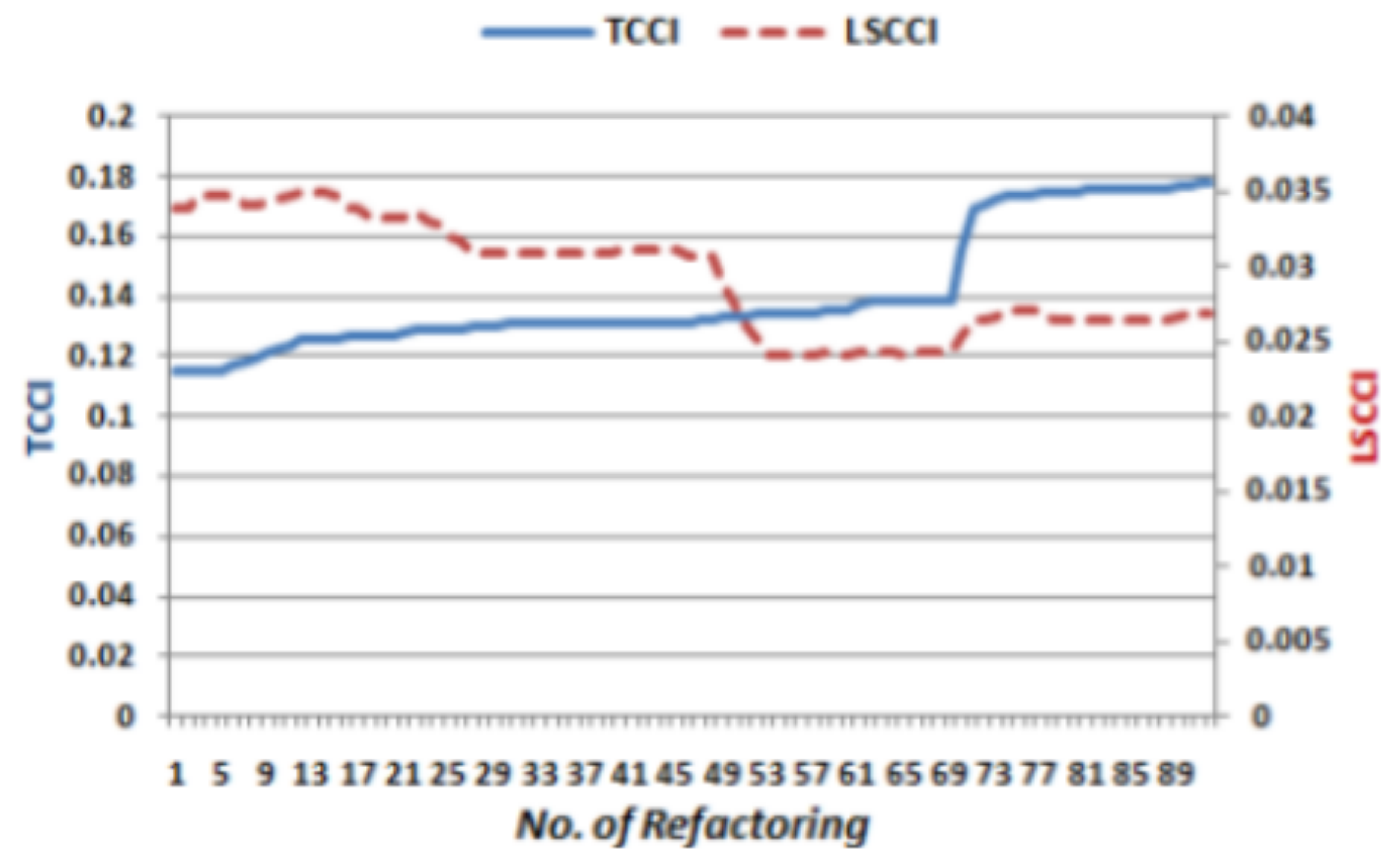
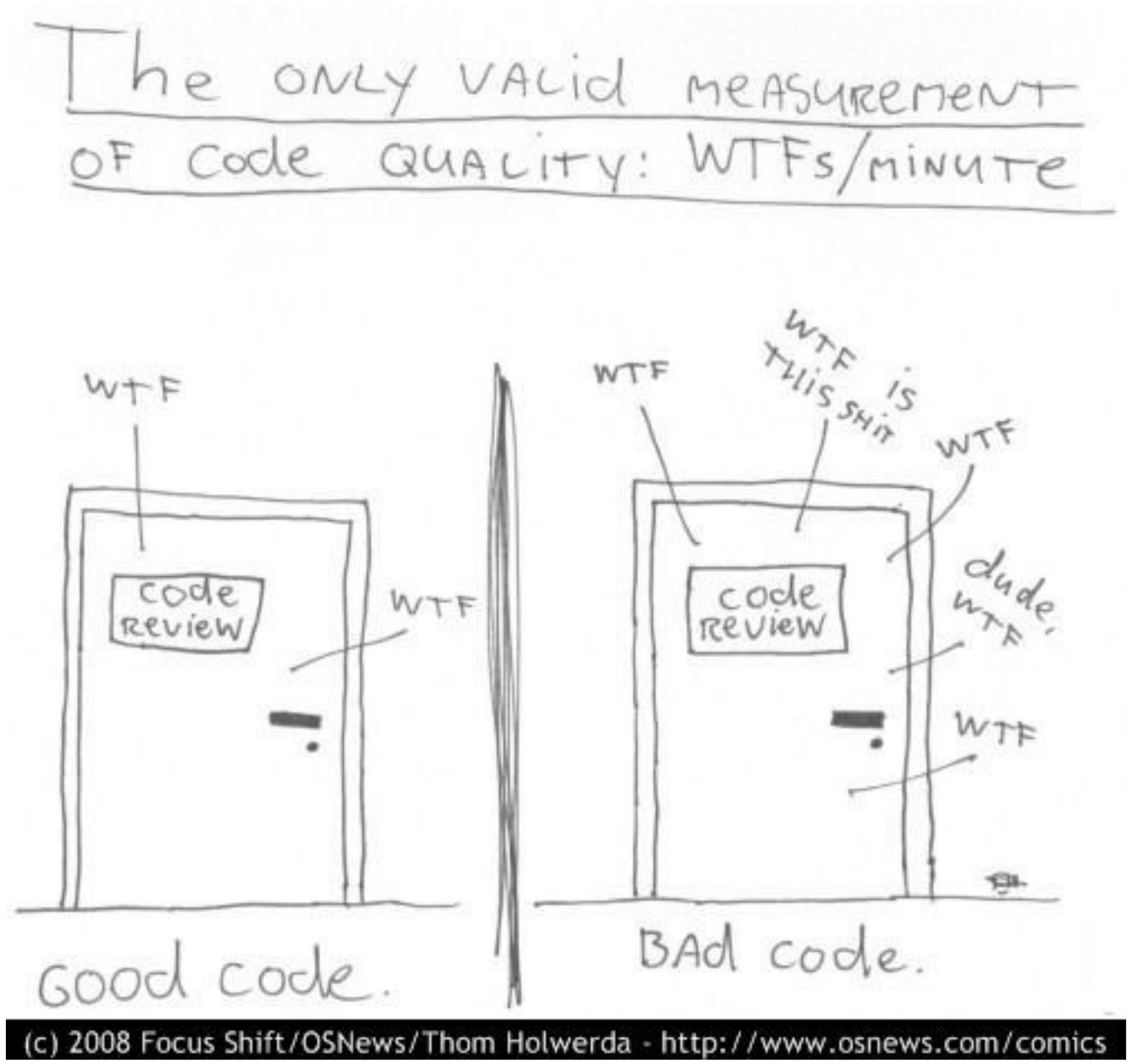


Figure 5: Graph of $LSCC_i$ improving as TCC_i is improved by refactoring JHotDraw.

M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell, and I. Hemati Moghadam. Experimental assessment of software metrics using automated refactoring. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, ESEM '12, pages 49–58, 2012.

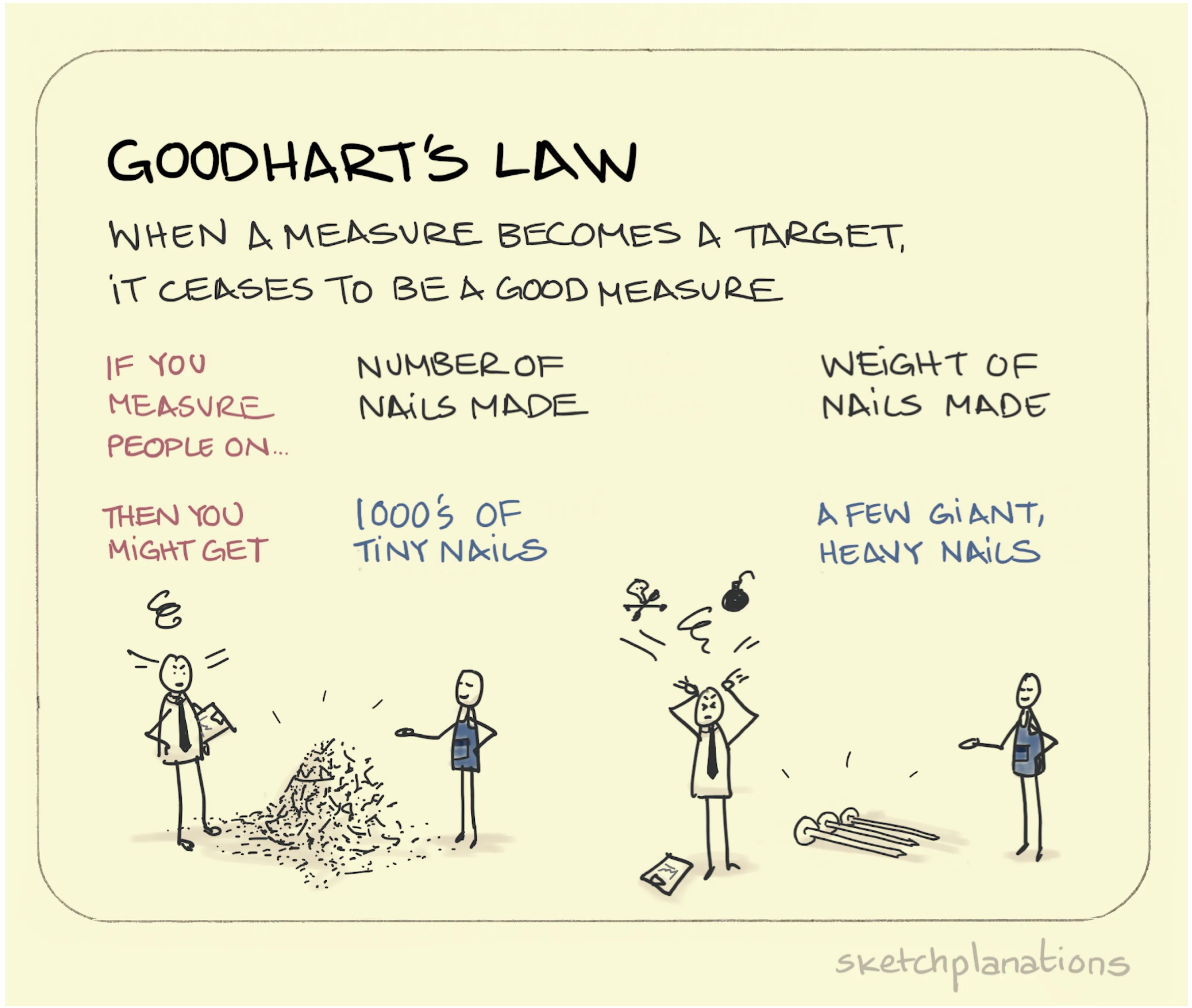

The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
Bad code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

We will talk about code reviews later in the course, but this cartoon really fits in today's lecture.

Goodhart’s Law

(aka why measurement is hard at management level)

- “Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes” - Charles Goodhart, 1975
- In other words, once you make a metric a target, **it will be abused and gamed**.
- Can you think of examples?



https://sketchplanations.com/goodharts-law

Goodhart's Law applied to SW Development

(aka why measurement is hard at management level)

- How should managers measure team's productivity?
    - SLOC (Soure Code Lines of Code), i.e., by asking how many lines of code have been written?
    - Lower man-hour, i.e., have we done this efficiently by spending minimum hours?
    - Velocity (an agile term, we will look at this in more detail later), i.e., by measuring how many "story points" a team completes in an iteration?

There are tons of theories and best practices

…just for measuring stuff

- For example, the Software Engineering Institute (SEI) at CMU as a whole group devoted to measurement, named Software Engineering Measurement and Analysis (SEMA)
- Their 2009 technical report on the topic, titled “Measurement and Analysis Infrastructure Diagnostic (MAID) Evaluation Criteria” is a guideline on how to implement a proper measurement infrastructure in organizations (51 pages long), requiring organizations to define
  - how to make everyone aware of MAID, where to store data, what to collect, how frequently to collect data, how to plot graphs…

Joel Test: 12 Steps to Better Code

(https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/)

- Joel Spolsky: MS Excel Product Manager (1991-1994), created of Trello, as well as StackOverflow (with Jeff Atwood), writes the blog “Joel on Software”
- Joel Test is his suggestion on how to evaluate a SW development team, NOT based on a set of metrics, but as a set of good practices
- Many of the test items correspond to topics in this course

A score of 12 is perfect, 11 is tolerable, but 10 or lower and you've got serious problems. The truth is that most software organizations are running with a score of 2 or 3, and they need serious help, because companies like Microsoft run at 12 full-time.

Of course, these are not the only factors that determine success or failure: in particular, if you have a great software team working on a product that nobody wants, well, people aren't going to want it. And it's possible to imagine a team of "gunslingers" that doesn't do any of this stuff that still manages to produce incredible software that changes the world. But, all else being equal, if you get these 12 things right, you'll have a disciplined team that can consistently deliver.

Joel Spolsky

Joel Test

- Do you use source control?
- Can you make a build in one step?
- Do you make daily builds?
- Do you have a bug database?
- Do you fix bugs before writing new code?
- Do you have an up-to-date schedule?
- Do you have a spec?
- Do programmers have quiet working conditions?
- Do you use the best tools money can buy?
- Do you have testers?
- Do new candidates write code during their interview?
- Do you do hallway usability testing?

Do you use source control?

- Version Control System (VCS) allows you
  - To see what others are doing
  - To easily try out an idea
  - To be able to revert back when mistakes are made
  - To store duplicate copies of the code so that you do not lose much of it

Can you make a build in one step?

- Automated build is a key foundation in today’s software development lifecycle that really focuses on fast delivery
- A build pipeline that is broken down to many individual/manual steps will introduce errors —> you won’t be able to move fast if checking whether your changes are correct requires you to spend 30 minutes every time…

Do you make daily builds?

- Once your build is automated, doing it daily (instead of just before release/deployment) has many benefits
  - You will notice errors earlier, which means they are also cheaper to fix
  - Especially if the development is distributed, with people working on different branches/features using distributed VCS

Do you have a bug database?

- You CANNOT remember all the bugs in your head, so you need to document them in a reliable database.
- Each entry should include:
  - How to reproduce the buggy behavior
  - Who should fix it
  - Whether it has been fixed or not, and by which code change
  - Ideally, a test case that checks this bug will not reappear

Do you fix bugs before writing new code?

- Rushing to meet development schedule without ensuring that bugs have been fixed will result in… buggy software.
- At any given time, fixing bugs should be the top priority, HIGHER THAN writing new code
  - Bugs get increasingly more expensive to fix!
  - You all experienced not being able to understand the code <u>you</u> wrote six months ago, right? :)

Do you have an up-to-date schedule?

- Admit it, developers are not good at keeping schedule.
- There are stages in business that are important but not directly related to development: if the management is to make informed decisions about them, you need to have an up-to-date, reliable schedule.
- An up-to-date schedule allows (or forces) you to give up the least important features (which can be a good thing!)

Do you have a spec?

- Writing specification, i.e., the description of what the software is expected to do, is often perceived as a tedious task.
- Why? :)
- Think of writing spec documents as programming without coding - this will lower the cost of bug fixing even further!

Do programmers have quiet working conditions?

- People, especially knowledge workers, work best when they are “in the zone”
- It is difficult to get *into* the zone - it takes build-up time.
- It is extremely easy to get knocked *out* of the zone.
  - Programming is all about maintaining short term memory of various things.
- Small interruptions end up adding a significant amount of build-up time, preventing people from getting into the zone - this should be costed into everything, including working environment.

Do you use the best tools money can buy?

- Because any latency in workflow will lead to losing focus, which leads to losing productivity - which is more expensive than the piece of HW or SW that could have been bought.
- Obviously one cannot have EVERYTHING that one wants, but the cost-effective analysis should factor in people’s workflow and productivity. Organisations are on average getting better at this.

Do you have testers?

- Because, otherwise, you will be shipping/deploying buggy software.
- In general, testers are supposed to be cheaper than programmers.
- More to unpack about this later…

Do new candidates write code during their interview?

- “Would you hire a magician without asking them to show you some magic tricks?” - Joel

Do you do hallway usability testing?

- Hallway usability testing: grab five random people from the hallway, show them your code, make them to try to use your code
- You will spot problems that you haven’t recognized - developers tend to have a mental model where things work… in the way they initially imagined…
- This will get done much with very little cost (i.e., grabbing people from the hallway)

Is it still relevant?

- Some of the items are a bit dated and need to be interpreted in today's context - but they are all pointing in meaningful directions :)
    - Coding Interview: de facto standard, but what are we exactly measuring?
    - Working Environment: open spaces are almost universally recognized as sub-ideal, but WFH…?

Is it still relevant?

- Some are challenged by companies and projects that emerged later…
  - Google famously have very little testers - can you guess why? (But this does not mean that they do not test!)

Do not be dogmatic about anything

- Joel Test, when interpreted as an absolute target, will also succumb to Goodhart’s Law :)
- Remember that SW is infinitely variable
  - Grasp what is behind each test, embrace the higher level objective and not the low level test/metric
  - Do your best to improve ongoing practices