CS 428
Facts & Fallacies of Software Engineering (chapters 4-7)

WINTER 2023

BRUCE F. WEBSTER

Chapter 4: About Research

- #55: Many software researchers advocate rather than investigate. As a result, (a) some advocated concepts are worth far less than their advocates believe, and (b) there's a shortage of evaluative research to help determine what the value of such concepts is.
  - A lot of received wisdom in software engineering is not based on actual evaluative research, but rather what seems like common sense (at best) or wishful thinking (at worst).
  - This underlies a lot of promotion and hype about languages, tool, techniques, and methodologies.
  - "~~Trust, but~~ Verify."

Chapter 5: Fallacies about Management

- Fallacy #1: You can't management what you can't measure.
  - We manage things we can't measure all the time.
  - Much of software development is intellectual and sociological – it's pretty hard to measure that.
  - Compounding that, most 'metrics' used in software development fail the core requirements (informative/predictive, objective, automated)
  - Finally, trying to manage to metrics carries its own major pitfalls
    - *Measuring and Managing Performance in Organizations*, Robert D. Austin (1996)
    - Key finding echoes Weinberg's Law of Metrics: "That which gets measured get fudged."

Chapter 5: Fallacies about Management

- Fallacy #2: You can manage quality into a software product.
  - That is, "rah rah" approaches and motivational posters do absolutely nothing for quality
  - Quality ultimately depends upon the care, skill, and intelligence of the engineers
  - From my "Anatomy of a Runaway IT Project" article [link]
    - Mid-level management tells the developers that mood, sincerity, and commitment are everything, and that with them "we can accomplish anything." … Such assertions don't hold water. I can be in a great mood and have a team of very sincere and committed people, but if we try to build a commercial airliner without the proper expertise, requirements, engineering, materials, and testing, the plane will crash and people will die, assuming it ever gets built and off the ground (which is extremely unlikely). … When it comes to engineering, sincerity and commitment, while important, can never substitute for expertise and quality of work.

Chapter 5: Fallacies about Management

- Fallacy #3: Programming can and should be egoless.
  - This came from Jerry Weinberg's outstanding work, *The Psychology of Computer Programming* (first published in 1971)
  - His basic idea – that we need to separate our ego enough to allow others to view and critique our work – was a good one, but outstanding performance requires a healthy ego (and ego investment)
  - On the other hand, many engineering teams have ripped themselves apart precisely because one or more members couldn't set their egos aside.

Chapter 5: Fallacies about Management

- Fallacy #4: Tools and techniques: one size fits all
  - This often goes along with "silver bullet" syndrome.
  - Glass identifies some aspects that can vary widely between projects: size, domain, criticality, level of required innovation.
  - I'll add one more: people. This is why (Fact #1) people matter most.
  - Solution: get your team together, then figure out the tools and techniques that will best suit your project.
- Fallacy #5: Software needs more methodologies
  - There was a period in the 80s & 90s when new SDLC methodologies (and notations) were being proposed on a monthly basis. Fortunately, that has died down.
  - "Most people who use a methodology adapt it to fit the situation at hand."
- Fallacy #6: To estimate cost and schedule, first estimate lines of code.
  - No. Just, please, no.

Chapter 6: Fallacies about The Life Cycle

- Fallacy #7:Random test input is a good way to optimize testing.
  - Likewise, just no. The "infinite number of monkeys" approach just doesn't work.
  - Tests must very much be designed with input based on known stress areas.
- Fallacy #8: "Given enough eyeballs, all bugs are shallow."
  - No evidence to support this assertion, and deep/subtle bugs remain deep/subtle regardless of how many people are looking at them (cf. chromosome count).
  - Plus many engineers would rather add new features than track down and fix defects.
- Fallacy #9: The way to predict future maintenance costs and to make product replacement decisions is to look at past cost data.
  - One problem is that "cost data" tends to focus on defects rather than enhancements.
  - Another is that it can be unclear where the software is in its overall lifespan.

Chapter 7: Fallacies About Education

- Fallacy #10: You teach people how to program by showing them how to write programs.
  - His contention: we should start by teaching people how to read code and then have them read it and test them on how well they understand it and can discuss it.
  - I agree with him, though (like Glass) I know of no CS program that takes this approach.
  - There are some texts that show and examine code examples, but (as Glass points out) these are almost always written for people who already know how to code.
    - Even that's a help, since most of your coding career will involve picking up and reading other people's code