CSE 503 Winter 2013 • David Notkin

Lecture #1 Notes: Introduction

1. "Software engineering research"
    a. Software is software
    b. Software engineering is (roughly) the systematic construction of software
    c. Software engineering research is the study of software and software engineering and of ways to improve them both
2. So, what comes to mind when I say any of these three phrases? Shout it out! I'll write them on the board… (Can somebody write these down and mail them to me? Or take a picture of them?)
3. In the beginning … computers were a much more precious resource than were people – "it's the money, honey"
    a. "When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem." Dijkstra, 1972 Turing Lecture
    b. The Intel 8008 was released around 1972: It was an 8-bit CPU with an external 14-bit address bus that could address 16KB of memory
    c. Programming languages that were developed around that time include C, Smalltalk, Pascal, and Prolog
4. 1968 and 1969 NATO conferences on software engineering
    a. Friedrich Bauer chaired in 1968, with about 50 attendees including Turing Award winners Alan Perlis, Edsger Dijkstra and Peter Naur
    b. There were increasing difficulties and costs in developing software – the "human" vs. "computer" tradeoff had to be reconsidered
    c. Topics covered aspects of software including
        i. relation of software to the hardware of computers (including "the highly controversial question of whether software should be priced separately from hardware")
        ii. design of software
        iii. production, or implementation of software
        iv. distribution of software
        v. achieving sufficient reliability for software that is increasingly integrated into the central activities of modern society
        vi. the difficulties of meeting schedules and specifications on large software projects
        vii. the education of software (or data systems) engineers
    d. Rate of growth concerns

        i. Helms: In Europe alone there are about 10,000 installed computers — this number is increasing at a rate of anywhere from 25 per cent to 50 per cent per year. The quality of software provided for these computers will soon affect more than a quarter of a million analysts and programmers.
        ii. David: …OS/360 cost IBM over $50 million dollars a year during its preparation, and at least 5000 man-years' investment. TSS/360 is said to be in the 1000 man-year category. It has been said, too, that development costs for software equal the development costs for hardware in establishing a new machine line.
        iii. d'Agapeyeff: In 1958 a European general purpose computer manufacturer often had less than 50 software programmers, now they probably number 1,000-2,000 people; what will be needed in 1978?

5. The term "software crisis" was (perhaps) coined during these meetings and has come down to questions like
    a. Why does software cost so much?
    b. Why does software [testing | maintenance | …] cost so much?
    c. Why are there so many errors in software?
    d. Why do so many software projects fail?
    e. Why can't software be more like hardware or cars or buildings or bridges or …?
    f. Why can't software engineering be more like *real* engineering?
    g. Where's Moore's Law for software?
6. Standish Report 1995
    a. U.S. spends more than $250 billion annually on IT application development
    b. The average cost of a development project ranges from $434K (for small) to $2.3M (for large) projects
    c. 31.1% of projects will be canceled before completion
    d. 52.7% of projects will cost 189% of their original estimates
    e. The failure to produce reliable software to handle baggage at the new Denver airport [cost] the city $1.1 million per day
    f. "A great many of these projects will fail. Software development projects are in chaos, and we can no longer imitate the three monkeys -- hear no failures, see no failures, speak no failures."
    g. "The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars."
    h. "One just has to look to the City of Denver to realize the extent of this problem."
7. "Software's Chronic Crisis" by Gibbs *Scientific American* September 1994

    a. “To veteran software developers, the Denver debacle is notable only for its visibility. Studies have shown that for every six new large-scale software systems that are put into operation, two others are canceled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three quarters of all large systems are ‘operating failures’ that either do not function as intended or are not used at all.”
8. More recent Standish Group reports show some improvement in the statistics
    a. The, however, reports are still clear about the continuing presence of the software crisis.
    b. Jim Johnson, the founder and chairman of the Standish Group, said in 2006: “People know that the more common scenario in our industry is still: over budget, over time, and with fewer features than planned.”
9. Software lifecycle costs: Evolution/maintenance ≅ 90%
   http://www.cs.jyu.fi/~koskinen/smcosts.htm
    a. “The relative cost for maintaining software and managing its evolution now represents more than 90% of its total cost”
    b. “[A]lthough there has not been much empirical research on this particular area, the magnitude of the maintenance cost effects is clearly identifiable.”
10. Software lifecycle costs: Testing/verification ≅ 50-75%
    Hailpern & Santhanam, IBM Sys. Journal 2002
    “In a typical commercial development organization, the cost of providing [assurances of functional and non-functional performance] via appropriate debugging, testing, and verification activities can easily range from 50 to 75 percent of the total development cost.”
11. So, these data show that…
    a. …software costs too much on an absolute basis
    b. …software lifecycle phases cost too much on a relative basis
    c. …software projects are cancelled too often
12. "When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge but you have scarcely, in your thoughts, advanced to the stage of science." *—Lord Kelvin*
    a. I don’t agree with this – there is great value in qualitative analysis as well
    b. Regardless, how do we measure “too”? as in …software costs *too* much…, software lifecycle phases cost *too* much …, software projects are cancelled *too* often…
13. Per-phase “pie” charts always have something in common – they total 100%, and they always will, even if software gets more expensive and worse
    a. A great way to compare the costs of phases
    b. A terrible way to assess costs in any absolute sense

14. Testing and evolution: why might we care about what "too" means?
    a. Is 50-75% too high for testing? What would be acceptable? Why? Is 0% a good goal? Are there benchmarks from other engineering disciplines and, if so, should we believe they may be analogous?
    b. Is 90% for evolution and maintenance too high? What would be a good goal? 50%? 10%? 0%? Or is it too low, and 99% would be a better goal? Even the desired directionality for this is not entirely clear.
15. Cyber-physical systems
    a. Dizzying increase in physical systems that have a significant software component: medical devices, spacecraft and airplanes, appliances, automobiles, bridges and buildings, telephones, and *many* more.
    b. When these systems fail – consider the Denver airport, the Mars Polar Lander, the Arianne V, and *many* more – they are generally reported as software failures
    c. In an important sense, this is accurate – specifically, in most cases, a variant on the software could have eliminated the failure – however, at the same time, these could be reported as hardware failure, as in most cases, a variant of the hardware could also have eliminated the failure
16. Mars Polar Lander: $100M lost (plus opportunity costs)
    "…the most likely cause of the failure of the mission was a software error that mistakenly identified the vibration caused by the deployment of the lander's legs as being caused by the vehicle touching down on the Martian surface, resulting in the vehicle's descent engines being cut off while it was still 40 meters above the surface, rather than on touchdown as planned." [Wikipedia]
17. Co-design decisions
    a. Allocation of function to physical vs. software components is critically important
    b. In some domains these decisions come from those with more know-how on the physical side
    c. Even more commonly, these decisions are made with a clear view that much complexity can and should be pushed into the software
    d. Thus, it is tautological that software would cause more problems in cyber-physical systems simply because it is "assigned" greater complexity.
    e. That is, Increasing the complexity of the software is (surely at times) a fine decision – but one should not then later be surprised at increased risks and costs
18. Physical components generally require a long lead time for design and manufacture; for practical reasons, this is done concurrently with software production
    a. The physical components and their means of production necessarily and practically become more stable and more costly to change over time
    b. Changes made at later stages tend to be much more costly to fix

    c. Just like software ☺ but even *more* costly!
    d. If you really think software is too hard to change, try changing the physical components instead!

19. Changes to software requirements
    a. Unexpected shortcomings on the physical side are often handled by changing the software requirements
    b. This adds complexity and cost to the software because numerous design and implementation decisions have already been made during the concurrent development
    c. To accommodate flaws in the engineering of the physical components, even more complexity is injected into the software
    d. "Better" software can generally overcome these flaws, but the need to do so is induced by weaknesses on the physical side
20. Software is last
    a. Testing software on the physical system instead of on simulators, mockups, etc. may be cheaper and easier
    b. When software is changed to overcome physical flaws, the software is necessarily later
    c. There is, quite reasonably, a perception that software is indeed "soft" compare and thus it seems to be able to withstand changes until (and often after) the last moment
    d. But just because it is last doesn't mean it is (entirely) at fault
21. Software: breaking [Moore's] law" [Wikipedia]
    a. "… exponentially improved hardware does not necessarily imply exponentially improved software performance to go with it. The productivity of software developers most assuredly does not increase exponentially with the improvement in hardware, but by most measures has increased only slowly and fitfully over the decades."
    b. The performance of software and software developers is compared to transistors on an integrated circuit
    c. What human activity has matched the growth of Moore's Law?
    d. Do we (or should we) compare the performance of trains to their tracks? Of train designers to their trains?
    e. What other technology has matched the growth of Moore's Law? Batteries, displays, ??? IC circuits are a (wonderful and probably) singular technology
22. Blame isn't the goal
    a. Simply blaming software for the problems because it *could* fix the system and because it *was* (naturally) last to be stabilized ***cannot*** easily lead us to better solutions to costly fiascos
    b. Not considering the role of software would also fail to lead to better solutions

   c. Of course we as software engineering researchers and engineers must work hard to do better – indeed, much better
   d. We must not, however, let the playing ground be set in a way that is not helpful towards achieving critical goals
23. Value: missing from most discussions
24. Value is definitely hard to measure – but the world has surely agreed that software has value, or else companies that produce and sell it would not exist!
   a. Barry Boehm, Kevin Sullivan, Mary Shaw, and others have worked on software engineering economics – this is crucial but very difficult
   b. But we have to remember that the reason software is important is because it provides value – real value to society, to the economy, to people – and if it didn't, nobody would care about cost, dependability, etc.
25. Reprise: Standish '95: U.S. spends > US$250B annually on IT application development
   a. Software industry (2008, worldwide) US$304B DataMonitor via Wikipedia
   b. Advertising industry (2009, Worldwide) US$445B http://www.plunkettresearch.com/advertising%20branding%20market%20research/industry%20statistics
   c. Travel industry (2008, Worldwide) US$944B Wikipedia
   d. Porn industry (2004, Worldwide) US$57B http://www.toptenreviews.com/2-6-04.html
   e. Size is an inherently limited way to assess how well an industry is doing…
26. Different kinds of question…that could and should drive software engineering research
   a. What should software systems cost to design, build, maintain? Can we find a useful lower bound?
   b. If we had infinite cycles to help software engineers, what problems would still exist?
   c. When changing software, we assume that new behavior can be arbitrarily far from old behavior. What if we instead focused on the common-case – a small Δ?
   d. Under what conditions is it reasonable/unreasonable to characterize a class of software systems as similar/dissimilar?
   e. How should we legitimately assess and achieve important properties that are – even if we dislike it – not binary, not efficiently computable, not even precisely defined, etc.?