P1L1 Introduction and Overview

Notes 08/18/2014

Overview of Course and Answer Two questions:

- *What is Software Engineering?*
- *Why do we need it?*

Will interview academia and industry experts

Software Engineering – Introduction and Overview

Course Overview

- Processes – Rigid to Agile
- Phases – Activities in Software Development Process
- Tools – Modern ones that can help improve a programmer's productivity
- Projects - Put what we learn into practice through realistic projects of increasing complexity

Course Structure (4 main parts – each with a main topic and a related project)

- Software Engineering Introduction – basic concepts
    - Different software phases and different types of development processes
- Remaining 3 parts focus on specific activity within software development
    - Requirements and Prototyping
    - Design and Unified Software Process
    - Testing and Test Driven Development

Requirements

- Install tools – some virtual machines may be provided that you can use out of the box
- Teamwork – but, assignments should be done individually and not as a group
- Online submission – code, documents, or both
- Reading – highly recommended that you read material listed under the instructor notes

What is Software Engineering? Why do we need it?

- Tevfik Bultan (UC Santa Barbara) – A computer is a programmable device. Essence of computing is programming. Program development is the most essential use of the computer. SE is the discipline that investigates development. How can it be done more efficiently? What's the best way to do program development? How can you develop reliable programs?

- Jane Cleland-Huang (DePaul University) – SE is the systematic application of methods to build software in a rigorous way. Not only technically building the system but understanding the requirements, working with stakeholders, trying to find a solution that balances all of the stakeholders' needs in order to deliver the software in a way that's tested and rigorous. Healthcare.gov not engineered correctly. Intersection between requirements, architecture, politics, and project management are important concepts that have to go into the software mix.
- Mukul Prasad (Fujitsu Laboratories of America) – Creation of Software using engineering principles. If not done in a principled way it will be bad and every user will suffer.
- Margaret M. Burnett (Oregon State University) – Programming is about the create part of software. SE is about the entire lifecycle. It's about quality. You have to consider aspects of things long after you ship.
- Tao Xie (Univ of Illinois Urbana-Champaign) – Need it for better software productivity. Want faster and higher productivity.
- Wolfgang Emmerich (CEO at Zuhlke Engineering Ltd) – Set of activities one engages in when building software systems. Fundamentally a venue creating activity. Involves social processes.
- Andreas Zeller (Saalend University) – Act of many people working together and putting together many versions of large and complex systems. Would end up in chaos because people wouldn't know how to organize themselves or the software. Much of it is simple rules you must apply to get things done. Some look at them and think they are obvious. But once you try to apply them, they're not obvious at all.
- Kevin Sullivan (University of Virginia) – Activity of envisioning and realizing valuable new functions with sufficient and justifiable confidence that the resulting system will have all of the critical quality attributes that are necessary for the system to be a success. SE is the activity of doing this for not only the components but the system overall. Crucial; engineering discipline uniquely capable of carrying out engineering system.
- John Penix (Google) – Art and practice of building software systems. If you don't think about how you're building the system and trading off different aspects like performance and scalability and reliability, then it's going to end up breaking, not lasting very long, or not doing what you want it to do, or be really expensive.
- Nenad Medvidovic (Univ of Southern California) – Set of methods, principles, and techniques developed that enable us to engineer or build large software systems that out-pace a small team's abilities to understand and construct and maintain overtime. Long term investment. Often requires support for systems intended for one purpose but get used for many others. Software is everywhere around us and the way you build it and maintain it determines almost the basic quality of life nowadays. Getting the software right is difference between a fun product and one you won't want to use, a

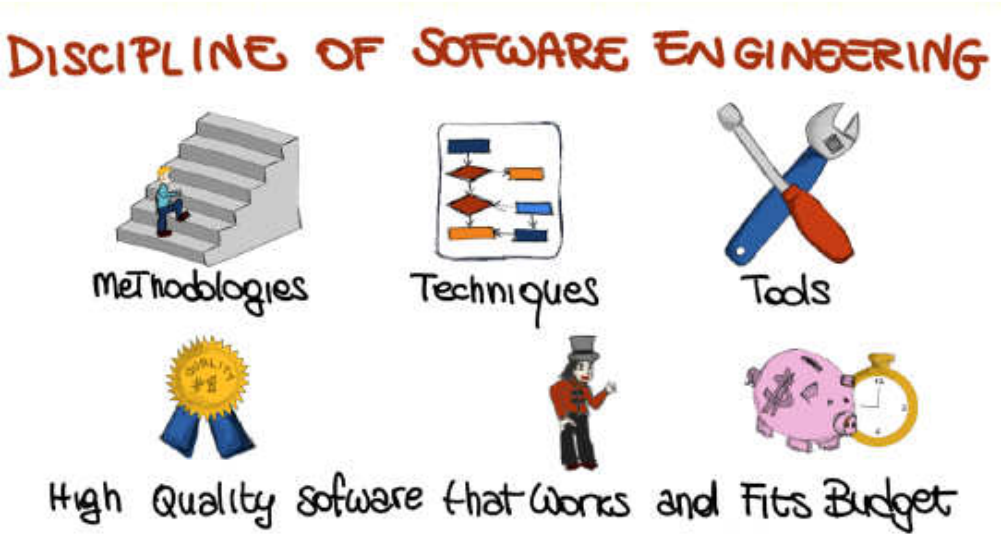
reasonably successful company and one that fails, even life and death if you consider something like the programming in an airplane.

- Jonathan Maletic (Kent State University) – About building and constructing very large scale and quality systems. Everything's built on software systems and these are ubiquitous in our society.
- Lionel Briand (Univ of Luxembourg) – SE is engineering discipline of developing software based systems usually embedded into larger systems of hardware and humans and business processes and processes in general. Software is pervasive in all industries and systems must be reliable, safe, and secure. SD is distributed and involves teams of people that have to work together; clients and users; domain experts; so we can't simply sit down and start coding.
- Willem Visser (StellenBosch Univ) – Mostly being able to program. Need to be able to put big systems together so they work. You could maybe hack something up but when you try to change it, it will break.
- Bernd Fischer (Stellenbosch Univ and Univ or Southampton) – There are programs out there that if they screw up, we're all screwed

Explosion of Ariane 5 rocked due to software errors (10 years, 7 billion, 5 million of cargo on board)
[Link to History's Worst Software Bugs]

Even without extreme examples, we've all experienced crashes.

Why is it so hard to build (good) software? It's the reason why software engineering is a fundamental discipline in computer science.



Discipline of Software Engineering

- Methodologies
- Techniques
- Tools
- The above 3 allow us to:
  - Build high quality software that does what it's supposed to do.
    - This makes our customers happy.
  - Build software within the given time and money constraints, i.e., within the budget that is allocated for the software.

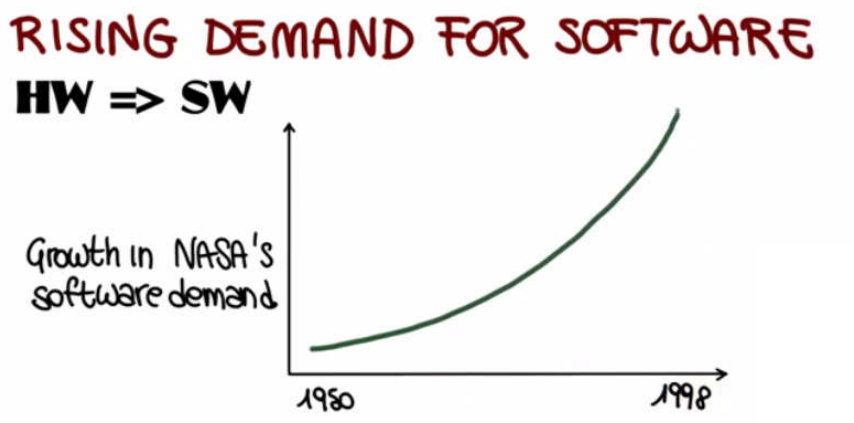
Important to look at Historical Perspective

The 1960s

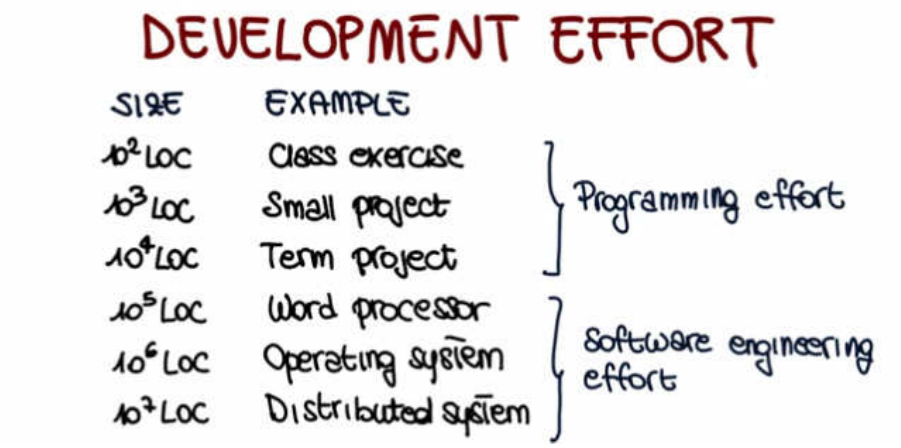
- Man on the Moon
- Woodstock
- Polaroid (first 60 sec picture)
- Concurrently, people realized they couldn't create the software they needed
  - This led to the Software Crisis

The Software Crisis (Reasons)

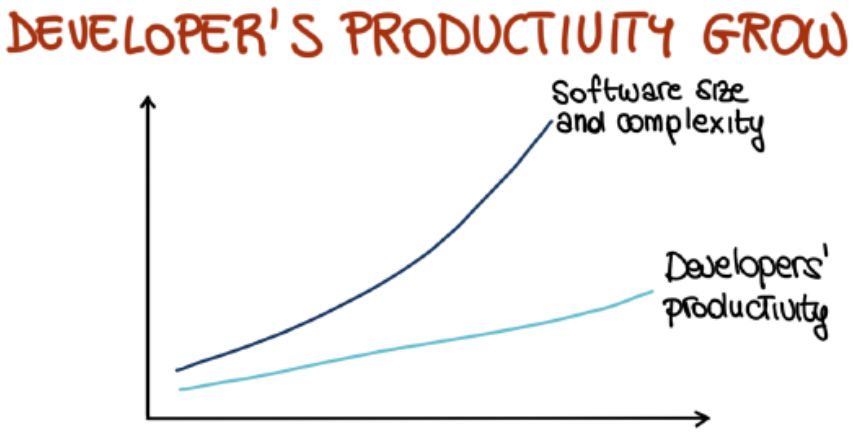
- Rising Demand for Software
  - Before 60s, size and complexity of software was very limited and hardware components were dominating the scene
  - Then the situation changed and software became more prevalent
  - We moved from a situation where everything was mostly hardware to a situation in which software became more and more important

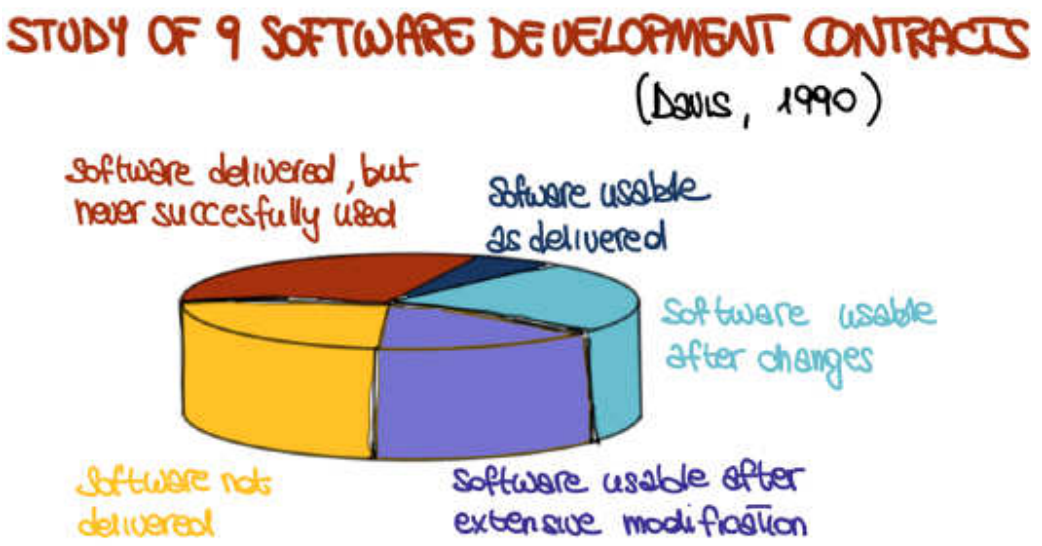


- Development Effort
    - Increasing amount of development effort needed due to product complexity
    - Effort is not linear
    - Programming effort – Heroic effort of an individual developer can get the job done; if you're a good programmer you can go sit down and do it right
    - Software engineering effort – No matter how much programming languages, development environments and software tools improved, developers could not keep up with increasing software size and complexity



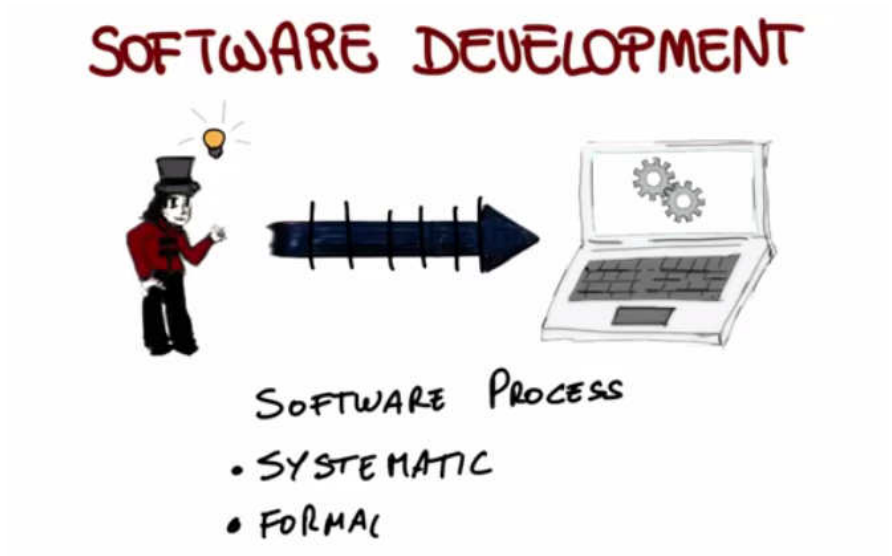
- Developer's Productivity Growth
    - Developer Productivity couldn't keep up with software complexity
    - Gap between what was needed and what was available





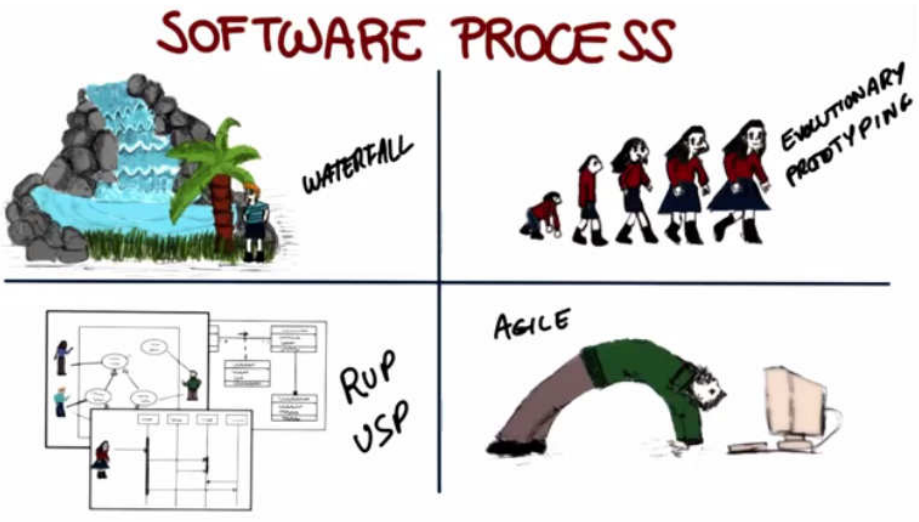
Study of 9 Software Development Contracts (Evidence of a Crisis)

- Done by Davis in the 1990s
- Total cost $7 million
- Extensive modification means additional cost
- Largest parts (red and orange together) accounted for $5M.
- That's a grim picture for software development and its success
- This led to NATO Software Engineering Conference in Jan 1969 (what we consider the birth of software engineering development)
    - Link to proceedings: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/



Software Development (in a smarter, better, successful way)

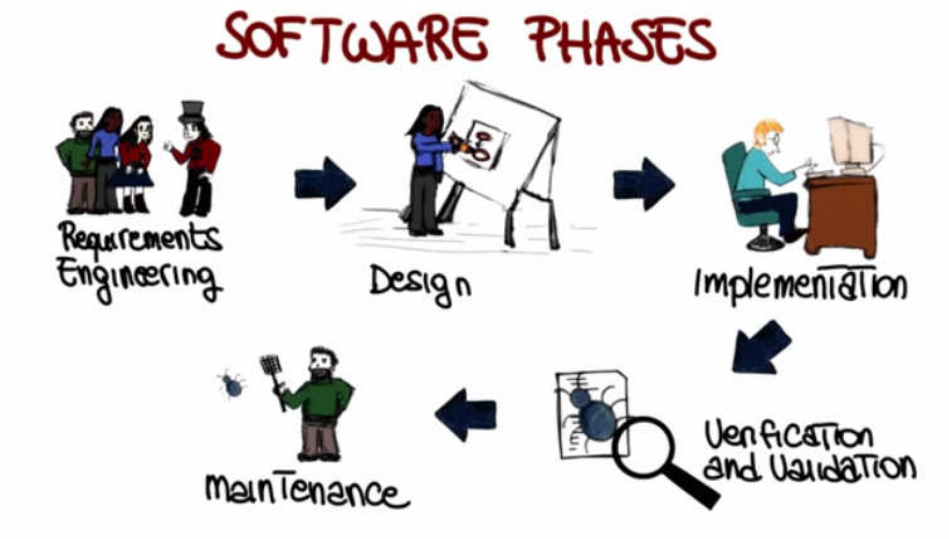
- SE is fundamentally going from an abstract idea in somebody's head to a concrete system that actually implements that idea and hopefully in the right way
- Very complicated and overwhelming
- That's when software process comes to the rescue
- SP is a way to break down an unimaginable task into smaller steps that we can handle individually
- Having SP is fundamentally important for several reasons:
    - For not trivial systems, you can't just sit down and develop
    - Need to break it down in a SYSTEMATIC way
    - Need to break it down in a more or less FORMAL way



Software Process (multiple possible process depending on context and kind of applications you're developing)

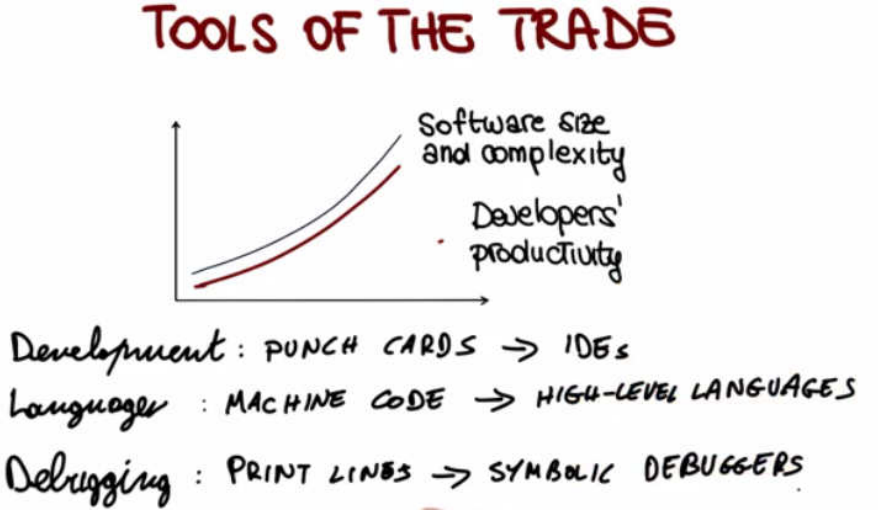
- There are four main software processes
    - Waterfall – go from one phase to another like water falls in a waterfall
    - Evolutionary Prototyping – instead of rigid steps like waterfall, start with an initial prototype and evolve it based on feedback from customer
    - Rational Unified Process (RUP)/Unified Software Process (USP) – heavily based on use of UML
    - Agile – sacrifice discipline a little bit to be more flexible and more able to account for changes in requirements

Studies have shown the professional software engineers produce between 50 and 100 Lines of Code (LOCs) per day. May not seem much, but coding is not everything. Coding is not the only activity you have to perform.



Software Phases (What Software Processes are normally characterized by)

- Requirements Engineering – phase where we talk to customers or stakeholders or whoever we're building the software for and we try to understand what kind of system we need to build
- Design – high level structure that can become more and more detailed of our software system
- Implementation – write code that implements the design we just defined
- Verification and Validation – make sure code behave as intended
- Maintenance – involves several activities including adding new functionality or eliminating bugs from the code or responding to problems that were reported from the field after release of software



Tools of the Trade (Things that can improve the software phases, and can support software development tasks in general)

- Tools and automation are fundamental in software engineering
- Fundamental for improving productivity and effectiveness of our activities in the SD process
- How tools can improve productivity
    - Think about what kind of improvement it was to go from punch cards to modern IDEs
    - Think about of how much more productive developers became when going from writing machine code to writing code in high-level languages
    - Debugging
        - Very important and expensive activity
        - Moving from the use of print lines to the use of symbolic debuggers dramatically improved the effectiveness and efficiency of development
- We will use three main kinds of tools
    - IDEs – Integrated Development Environments: advanced editors in which you can write, compile, run, debug, and test code
    - VCS – Version Control Systems (Ex: Git): systems that allow you to save, restore, and check differences between different versions of code
    - Coverage and Verification Tools: tools that help you during testing