Lecture Notes
September 29, 2020

Working in a team: *software process*

Software Engineering is **not** Programming

Many people consider themselves IT staff, coder, programmer, web/mobile developer, etc., but that doesn't make them a *software engineer*

Programmers **Program**	**Software Engineers** **Build Software Products**
Well-defined problem	Ill-defined problem
Alone or small group	Teams or Teams of Teams
Hundreds/thousands LOC	Tens-thousands/millions LOC
Submitted for grading or used personally	Runs in production with real-world users
Fresh codebase	Existing codebase
Stop and Restart to Change	Continual Availability
Low cost of errors	Potentially high cost of errors

> [Top 10 famous computer bugs that cost millions of dollars]

Addendum:

Are software engineers, or their companies, legally liable for any deaths (or other injuries) caused by their code?

See [What you need to know about software liability]

This article does not address whether the individual software engineer(s) might have some liability. It refers to “software developers” but apparently means the companies employing the developers.

Companies that sell devices together with embedded software can be held liable in the same way companies are accountable for manufacturing defects or defective products.

In principle, the same rule applies to a stand-alone software product, or to a software product developed for a device separate from the device manufacturer, when there is no contract or license agreement that states otherwise.

However, most software vendors use end-user license agreements (EULAs) and/or terms and conditions agreements (T&C) to define the terms of the relationship. These contracts contain provisions that limit the liability of the software vendor, even if the software fails entirely. These provisions are generally enforceable.

The article gives some actual and hypothetical examples of liability, but they all involve devices where the user cannot use the device without also using the software and the software is "safety-critical software" (e.g., Toyota Prius, Boeing 737 MAX).

Nevertheless, the article recommends that software development companies (particularly small businesses) purchase three kinds of insurance: general liability insurance, software product liability insurance, errors and omissions insurance. It is important to note that this article is posted on the website of a company (Insureon) that markets all three kinds of insurance.

Typical EULA warranty disclaimer:

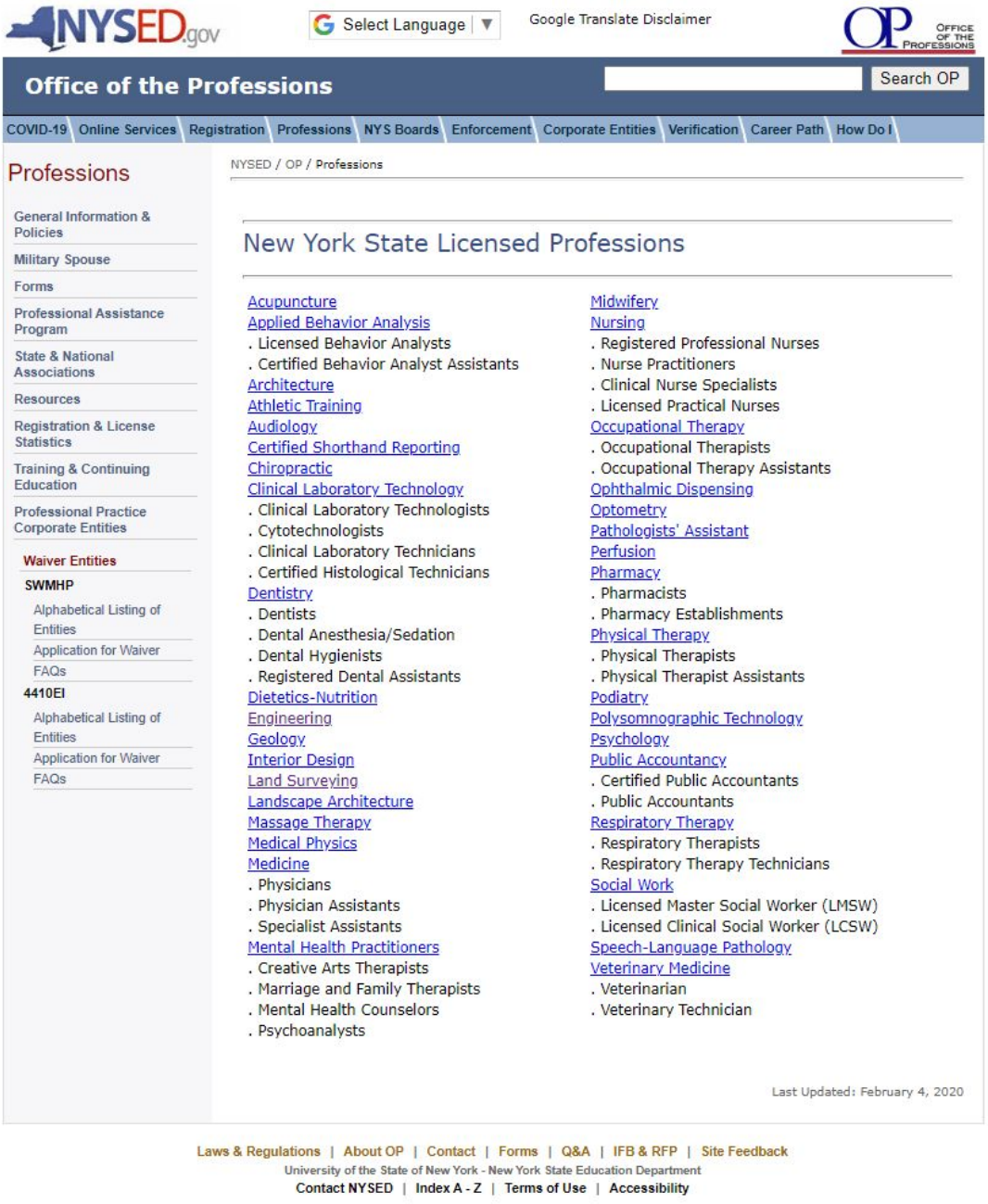
8. Disclaimer of Warranties. YOU ACKNOWLEDGE AND AGREE THAT THE APPLICATION IS PROVIDED ON AN "AS IS' AND "AS AVAILABLE" BASIS, AND THAT YOUR USE OF OR RELIANCE UPON THE APPLICATION AND ANY THIRD PARTY CONTENT AND SERVICES ACCESSED THEREBY IS AT YOUR SOLE RISK AND DISCRETION. COMPANY AND ITS AFFILIATES, PARTNERS, SUPPLIERS AND LICENSORS HEREBY DISCLAIM ANY AND ALL REPRSENTATIONS, WARRANTIES AND GUARANTIES REGARDING THE APPLICATION AND THIRD PARTY CONTENT AND SERVICES, WHETHER EXPRESS, IMPLIED OR STATUTORY, AND INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. FURTHERMORE, COMPANY AND ITS AFFILIATES, PARTNERS, SUPPLIERS AND LICENSORS MAKE NO WARRANTY THAT (I) THE APPLICATION OR THIRD PARTY CONTENT AND SERVICES WILL MEET YOUR REQUIREMENTS; (II) THE APPLICATION OR THIRD PARTY CONTENT AND SERVICES WILL BE UNINTERRUPED, ACCURATE, RELIABLE, TIMELY, SECURE OR ERROR-FREE; (III) THE QUALITY OF ANY PRODUCTS, SERVICES, INFORMATION OR OTHER MATERIAL ACCESSED OR OBTAINED BY YOU THROUGH THE APPLICATION WILL BE AS REPRESENTED OR MEET YOUR EXPECTATIONS; OR (IV) ANY ERRORS IN THE APPLICATION OR THIRD PARTY CONENT AND SERVICES WILL BE CORRECTED. NO ADVICE OR INFORMATION, WHETHER ORAL OR WRITTEN, OBTAINED BY YOU FROM COMPANY OR FROM THE APPLICATION SHALL CREATE ANY

Typical EULA liability limitations:

9. Limitation of Liability. UNDER NO CIRCUMSTANCES SHALL COMPANY OR ITS AFFILIATES, PARTNERS, SUPPLIERS OR LICENSORS BE LIABLE FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH YOUR ACCESS OR USE OF OR INABILITY TO ACCESS OR USE THE APPLICATION AND ANY THIRD PARTY CONTENT AND SERVICES, WHETHER OR NOT THE DAMAGES WERE FORESEEABLE AND WHETHER OR NOT COMPANY WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. WITHOUT LIMITING THE GENERALITY OF THE FOREGOING, COMPANY'S AGGREGATE LIABILITY TO YOU (WHETHER UNDER CONTRACT, TORT, STATUTE OR OTHERWISE) SHALL NOT EXCEED THE AMOUNT OF FIFTY DOLLARS ($50.00). THE FOREGOING LIMITATIONS WILL APPLY EVEN IF THE ABOVE STATED REMEDY FAILS OF ITS ESSENTIAL PURPOSE.

Both images from here.

New York State Licensed Professions



NYSED.gov | Select Language | Google Translate Disclaimer | OP Office of the Professions

Office of the Professions | Search OP

COVID-19 | Online Services | Registration | Professions | NYS Boards | Enforcement | Corporate Entities | Verification | Career Path | How Do I

Professions

- General Information & Policies
- Military Spouse
- Forms
- Professional Assistance Program
- State & National Associations
- Resources
- Registration & License Statistics
- Training & Continuing Education
- Professional Practice Corporate Entities

Waiver Entities

SWMHP
- Alphabetical Listing of Entities
- Application for Waiver
- FAQs

4410EI
- Alphabetical Listing of Entities
- Application for Waiver
- FAQs

NYSED / OP / Professions

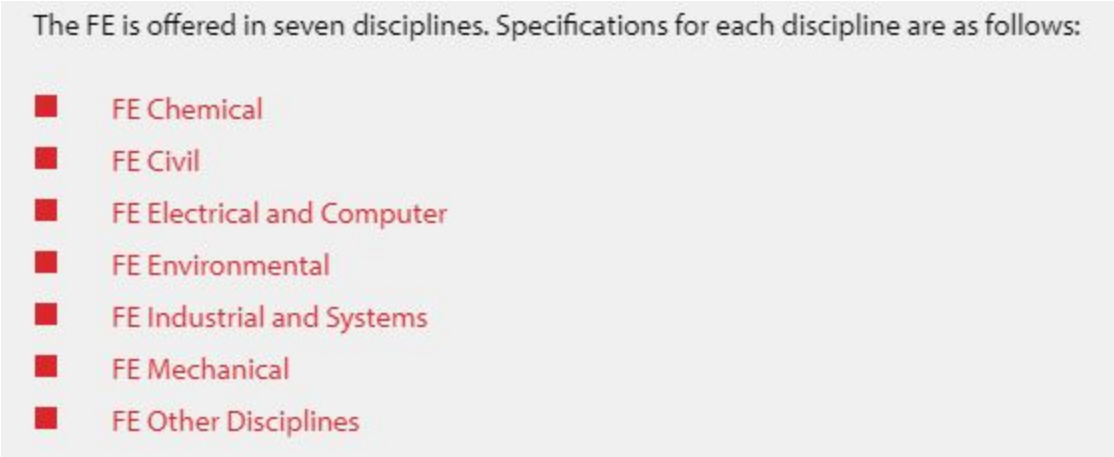
New York State Licensed Professions

Acupuncture
Applied Behavior Analysis
. Licensed Behavior Analysts
. Certified Behavior Analyst Assistants
Architecture
Athletic Training
Audiology
Certified Shorthand Reporting
Chiropractic
Clinical Laboratory Technology
. Clinical Laboratory Technologists
. Cytotechnologists
. Clinical Laboratory Technicians
. Certified Histological Technicians
Dentistry
. Dentists
. Dental Anesthesia/Sedation
. Dental Hygienists
. Registered Dental Assistants
Dietetics-Nutrition
Engineering
Geology
Interior Design
Land Surveying
Landscape Architecture
Massage Therapy
Medical Physics
Medicine
. Physicians
. Physician Assistants
. Specialist Assistants
Mental Health Practitioners
. Creative Arts Therapists
. Marriage and Family Therapists
. Mental Health Counselors
. Psychoanalysts
Midwifery
Nursing
. Registered Professional Nurses
. Nurse Practitioners
. Clinical Nurse Specialists
. Licensed Practical Nurses
Occupational Therapy
. Occupational Therapists
. Occupational Therapy Assistants
Ophthalmic Dispensing
Optometry
Pathologists' Assistant
Perfusion
Pharmacy
. Pharmacists
. Pharmacy Establishments
Physical Therapy
. Physical Therapists
. Physical Therapist Assistants
Podiatry
Polysomnographic Technology
Psychology
Public Accountancy
. Certified Public Accountants
. Public Accountants
Respiratory Therapy
. Respiratory Therapists
. Respiratory Therapy Technicians
Social Work
. Licensed Master Social Worker (LMSW)
. Licensed Clinical Social Worker (LCSW)
Speech-Language Pathology
Veterinary Medicine
. Veterinarian
. Veterinary Technician

Last Updated: February 4, 2020

Laws & Regulations | About OP | Contact | Forms | Q&A | IFB & RFP | Site Feedback
University of the State of New York - New York State Education Department
Contact NYSED | Index A - Z | Terms of Use | Accessibility

There are two levels of licensing for “Engineers”, Fundamentals of Engineering (FE), for final-semester engineering students and recent graduates, and Principles and Practice of Engineering (PE), for 4+ years experience.



The FE is offered in seven disciplines. Specifications for each discipline are as follows:

- FE Chemical
- FE Civil
- FE Electrical and Computer
- FE Environmental
- FE Industrial and Systems
- FE Mechanical
- FE Other Disciplines

The Principles and Practice of Engineering (PE) exam tests for a minimum level of competency in a particular engineering discipline. It is designed for engineers who have gained a minimum of four years' post-college work experience in their chosen engineering discipline.

For exam-specific information, select your engineering discipline.

- Agricultural and Biological Engineering
- Architectural Engineering
- Chemical
- Civil
- Control Systems
- Electrical and Computer (October 2020 pencil-and-paper exam)
- Electrical and Computer (computer-based testing)
- Environmental
- Fire Protection
- Industrial and Systems
- Mechanical
- Metallurgical and Materials
- Mining and Mineral Processing
- Naval Architecture and Marine
- Nuclear
- Petroleum
- Structural

Follow link from FE for Electrical and Computer
End of Addendum

There’s nothing explicit about testing in the table listing differences between programming and software engineering, but testing is implied by:

- Ill-defined problem - you won’t get it right the first time
- Teams - even if everyone’s code works perfectly in isolation, it won’t work perfectly when integrated
- Thousands->millions LOC - there will be some bugs lurking in there somewhere
- Runs in production - users will do things never imagined by the developers and consider it buggy if the software doesn’t work the way the users expected
- Existing codebase - some code may have worked once upon a time with earlier versions of the language, compiler, browser, operating system, network protocol, database, hardware, etc. but doesn’t continue to work when platform changes
- Continual availability - for conventional business/consumer software, users have very limited patience waiting for critical errors to be fixed, they will switch to a competitor’s product that works right now
- Potentially high cost of errors - bugs in conventional business/consumer software can lead to substantial financial, security and privacy losses (bugs in safety-critical software can cause human deaths)

My experience teaching software engineering courses is that *most students hate testing*. If the code compiles, doesn't crash, and seems to do the right thing with a couple examples, it works!

Most computer science and computational science faculty, research staff and PhD students also hate testing. Particularly machine learning folks, who use the term "testing" wrt prediction accuracy, not checking for logic errors. If the code compiles, doesn't crash, seems to do the right thing with a couple examples, and has greater than random chance accuracy, it works!

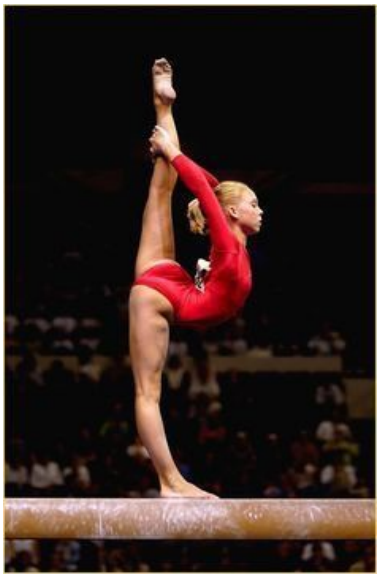
But the first thing almost every new software engineer does on their first job is test someone else's code.

The second thing is get their new code working with someone else's code - which also involves testing someone else's code. (Individual project.)

Only after that will the new engineer get a chance to develop their own *new* project. (Team project.)

Guest lecture on Thursday that I hope will get you excited about testing - Ilica Mahajan

Team software processes divide into “waterfall” vs. “agile”, which have become catch-all terms: waterfall means not agile and agile means not waterfall



Solo projects usually follow “quick-fix” process

For this course, waterfall = *phase-driven* process and agile = *adaptive/iterative* process. There are lots of other names for processes, e.g., spiral model and V-model. They are all variants of waterfall or agile.

Waterfall is a step by step sequential software development process that is simple to explain to beginning students, high-level (non-technical) management, and government procurement agents who review and signoff (and pay for) at each step

Do the entire first step: requirements analysts work with the customer to specify all requirements. Requirements analysts validate (review) the requirements specification with the customer to make sure this is what the customer wants. The requirements are not supposed to change later. But if they do change significantly, the process needs to start over (with some reuse of previous work).

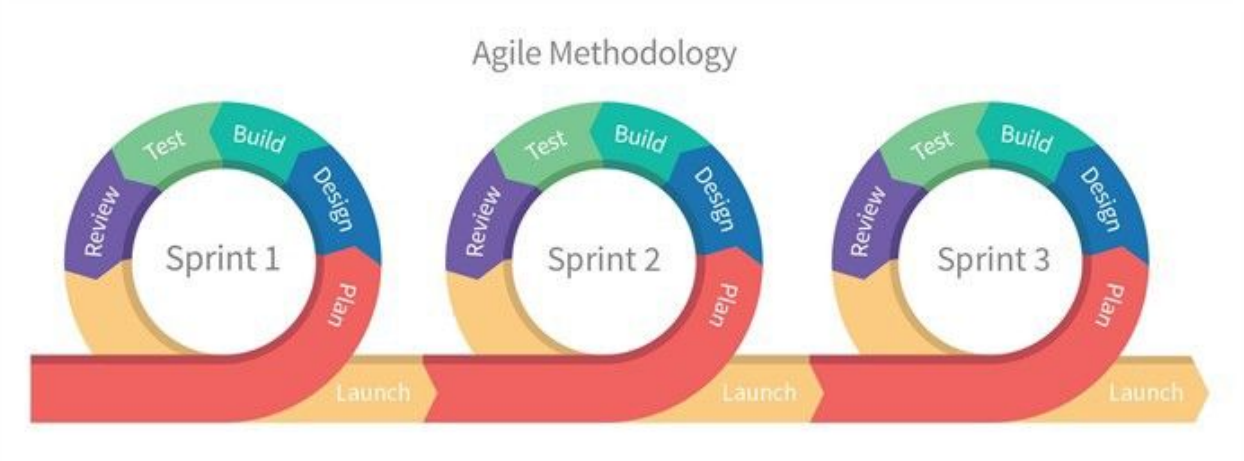
Do the entire second step: designers develop the entire design, planning all the modules, data structures, subroutines, etc. Designers validate (review) the design with requirements analysts to make sure it fulfills the requirements. Afterwards, the design is not supposed to change.

Do the entire third step: programmers write all the code, and make sure it compiles and builds. Programmers validate (review) the code with the designers to make sure it fulfills the design. Afterwards, the structure of the code (modules, classes, data structures, methods) is not supposed to change - assumes bug fixes will be small and localized.

Testers plan the test suite, and validate (review) the test suite with requirements analysts, designers and programmers to make sure it covers 1. the original requirements, 2. the design, 3. the code. Portions of test planning may run in parallel with these stages.

Do the entire fourth step: testers repeatedly run the full test suite until all tests pass. When bugs are found, testing cycles between programmers patching the code and testers rerunning tests. The patches are supposed to be modest bug fixes, not major rewrites, with no changes to design or requirements.

Finally deploy to the customer. All changes after that are considered “maintenance”, which is not really addressed by waterfall - except by repeating the entire waterfall process for version two.



In contrast, agile considers a small piece of the software project at a time in a “sprint” (or iteration), typically two weeks, with periodic releases to the customer (e.g., quarterly) or even after every sprint.

An agile sprint goes through all the waterfall-like stages, but for only that small piece of the project, and tries to parallelize as much effort as possible - the roles of requirements analysts, designers, programmers, testers are not necessarily separated.

Since relatively little time and effort has been invested in an individual sprint, it is feasible to “change everything” during the following sprint.

Waterfall is not “bad”, agile is not “good”

What preceded waterfall? Chaos (1940s and 1950s)

- Little or no structure, discipline, planning
- Little predictability over schedules and costs
- Didn’t scale to large projects with many developers

Waterfall was a huge improvement! (1960s - 1980s)

- “Road map” to coordinate multiple developers
- Control over scope of projects to improve predictability of schedule and costs

Why am I telling you about a 1960s-1980s process?

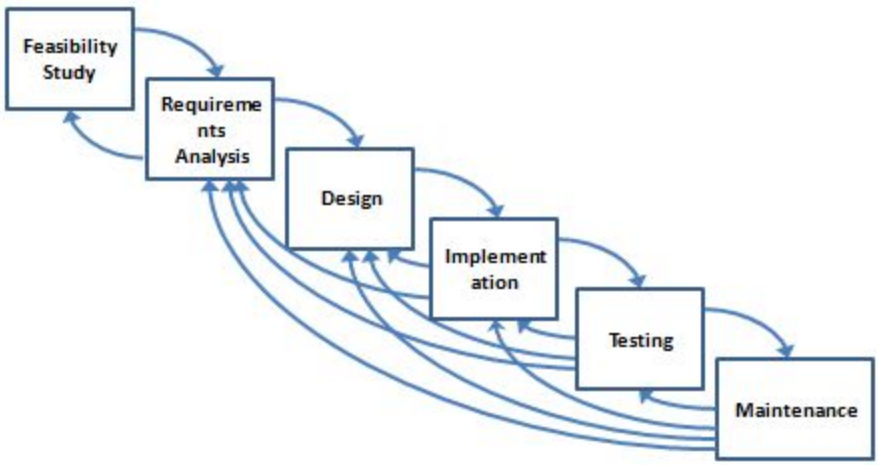
Waterfall, or some variant of waterfall, is still necessary for certain kinds of software projects, e.g., safety-critical systems

- Cannot easily be changed after deployment so requirements must be fixed in advance
- Each step must be carefully verified correct and complete before beginning the next step
- Examples: software controlling aircraft, cars, medical devices, nuclear power plants

Still widely used in government even for not safety-critical … sometimes agile comes to the rescue

Basic waterfall extended by Win Royce and others since ~1970

- Iterative/incremental, rapid prototyping (“do it twice”), involve customer
- But still phase-driven and document-intensive, cannot exit a phase the first time until the documentation required for that phase has been completed, reviewed, and approved



Contrast to projects that can or should evolve gradually, to allow for mistakes and mid-course corrections, continuing introduction of new features, relatively easy to re-deploy - most business and consumer applications

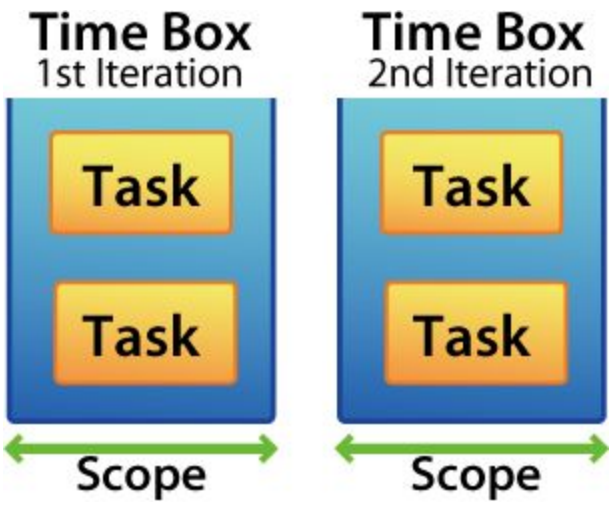
So today we have agile (since 1990s, although the term “agile” was coined in 2001)

- Most software organizations claim to follow an agile software process
- If you eventually work for two or more different “agile” organizations, you will learn there is not a single well-defined agile process used by everyone

Some well-known agile variants:

- Scrum (backlog)
- Kanban (work-in-progress limits)
- Extreme Programming (test-driven development)

Time box vs. Scope box



Fixed-length sprint means the scope is limited by what can be accomplished during that time period

If more features (scope) are added to a iteration, then the length of the iteration necessarily increases

If less time is allocated to an iteration, then fewer features (scope) can be completed

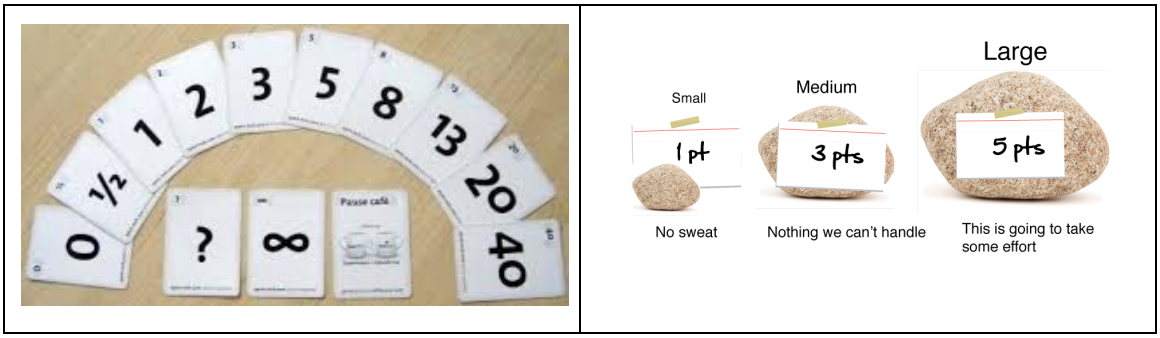
If it turns out that the originally planned features will take longer than scheduled, which happens often on new projects, then the scope must be reduced with some features pushed off to later sprint (or dropped)

Choosing time boxing over scope boxing (agile) controls costs per time period, because management knows how much the iteration will cost ~= how much the staff is paid for the time-period

But how do you predict how many iterations?

Many methods have been developed to try to predict costs up front, e.g., “function points”, COCOMO (COnstructive COst MOdel), “planning poker”, “story points”.

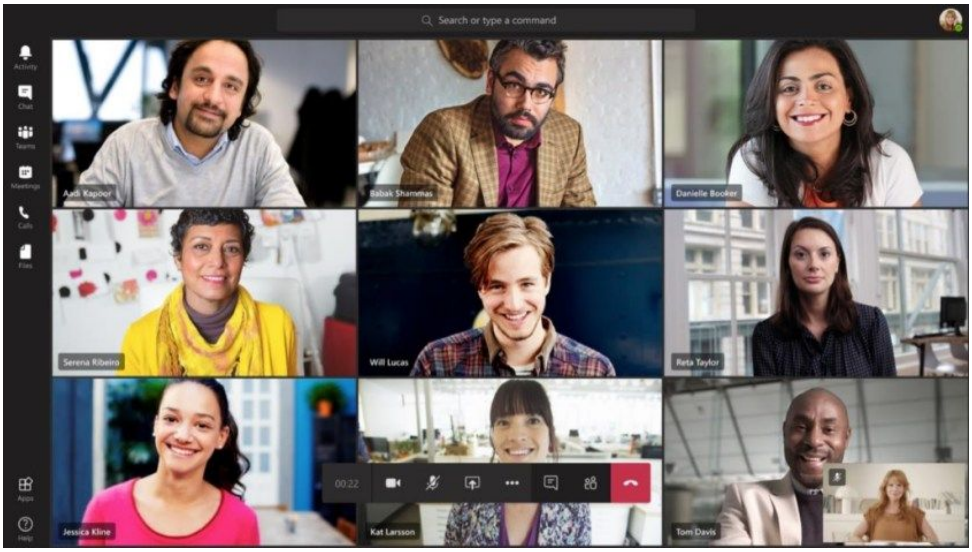
- Function points and COCOMO are associated with waterfall, planning poker and story points with agile.
- Same idea: How much time with how many engineers did it take to develop similar software last time?



Two things every customer wants to know: How long will it take and How much will it cost?

“Daily standup meetings” (no more than 15 minutes)

- ➢What did you do yesterday?
- ➢What will you do today?
- ➢Are there any impediments in your way?



Pair programming should be called “pair software engineering”, but “pair programming” sounds better - Applies to all aspects of software products, not just coding

Some use pair-programming interviews between a job candidate and an evaluator (current employee) even if their employees do not pair-program routinely

Pair programming is when two people sit side by side at same computer - or use desktop sharing or cloud IDE

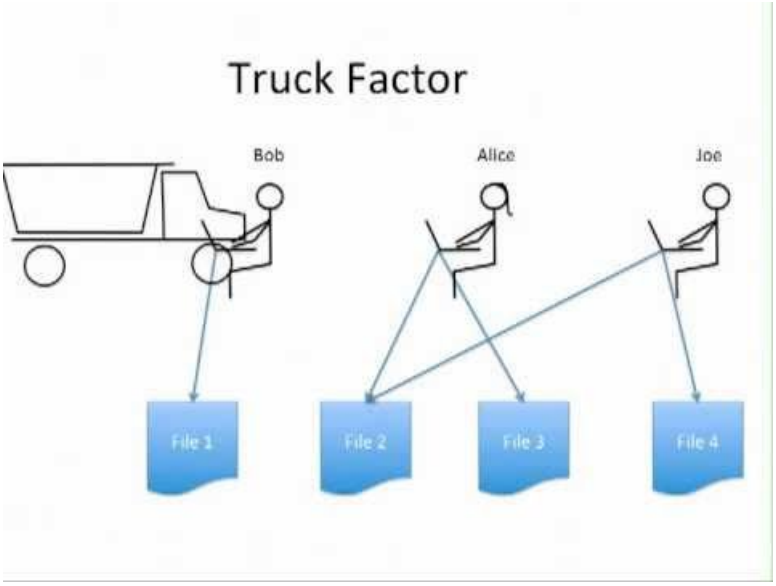
- Take turns “driving” (typing) vs. “navigating” (continuous review of code or whatever working on)
- Slide keyboard and mouse back and forth
- Switch roles at frequent intervals (e.g., 25 minutes)
- The pair does **not** divide up the work, they do everything **together**
- If necessary to work separately on something, e.g., when one partner is absent, they review together

Higher quality, but mixed productivity - often takes more total time to do the initial work (when you consider time-spent x 2), but less time later on bug repair

- Shared responsibility to complete tasks on time
- Stay focused and on task -
  Shared time treated as more valuable
  Less likely to read email, surf web, etc.
  Less likely to be interrupted
- Partners expect “best practices” from each other
- Two people can solve problems that one couldn’t do alone and/or produce better solutions
- Pool knowledge resources, improve skills

Organizations that use pair programming typically switch pairs often to reduce risk

- *Collective code ownership*
- Maximize "**truck factor**" = number of engineers that would have to disappear before project would be in serious jeopardy



“Assignment I2”
https://courseworks2.columbia.edu/courses/104335/assignments/482608
Assignment 2 builds on Assignment 1. Please make sure you have completed Assignment 1 before starting this assignment. In addition to using unit testing (JUnit), coverage tracking (Emma), and a static analysis bug finder (SpotBugs) to identify bugs in your application, you need to fix the bugs!

There will be one more assignment for this individual project, to add persistent storage of an in-progress game, with one more corresponding tutorial.