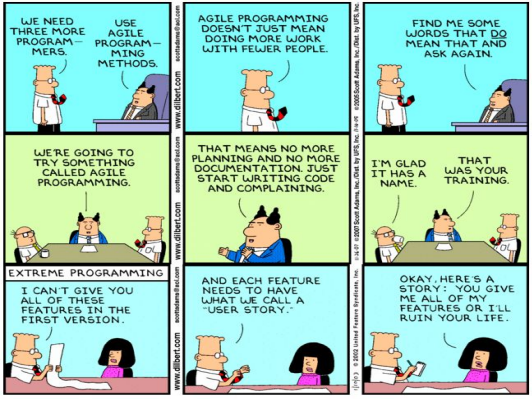
Module 3: Software Process
CPSC 310



C0
229
112
38
33*
Beginning
Acquiring
Developing
Pro■cient
●AutoTest latency in ■rst 7 days never exceeded 5 mins.
●AutoTest latency in last 3 days ranged between 1h and 22h.
■
Longest non-requested builds took 68h.
●Classy should re■ect your c0 bucket.
■
Your <highest,earliest> commit.

J0 & C1

- C1: Your main goal this week is to form a group in your lab, sign it up on Classy, and start planning (spec already posted).
- J0: This is due at 1800 on Sept 20.

Details are online: use this as a lens to re■ect on your C0.

One part is with your C1 partner, so ■gure that out ■rst!

You will talk 1:1 with your TA about this the following week.

WARNING: Do NOT leave your lab this week without a partner!

Dire warning!
You MUST have a partner this week by the end of your lab to do the project.

• Describe risks teams face and mechanisms for reducing these risks.
• Be able to explain why software is built in teams, and why larger teams face di■erent challenges than smaller teams (e.g., Brooks Law).
• Know what teamwork agreements / codes of conduct are, what they may contain, and why e■ective teams often have them explicitly recorded.
• Explain why processes are important, and why they should be adopted early.
• Provide some reasons that Waterfall was necessary at the time, what it involved, and why some projects today still rely on this process.
• Explain how extreme and agile processes arose, why they were possible, and what bene■ts and tradeo■s they involved compared to prior processes.
• Explain all the terms and practices used in Scrum.
• Explain the high-level di■erences between Scrum and Kanban.
• Be able to decompose high-level requirements into user stories and be able to evaluate user stories in terms of INVEST attributes.
• Elaborate on software process risks, explaining how they may a■ect project outcomes.
• Be able to recall high-level concepts from past requirements failures and be able to use this knowledge to reason about process risks.

Process

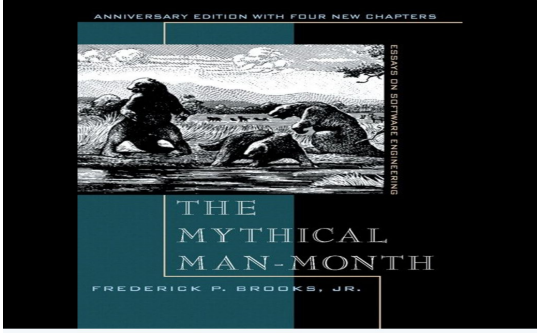
Examinable skills

Teamwork


GROWTH
TOGETHER
HELP!
TEAMWORK
SUCCESS
Ideas!
30%
10%

What is a team?
A group of people
… working together
… on a common goal.

What is teamwork?
The ability to cooperate and communicate e■ectively with others to achieve a common goal.



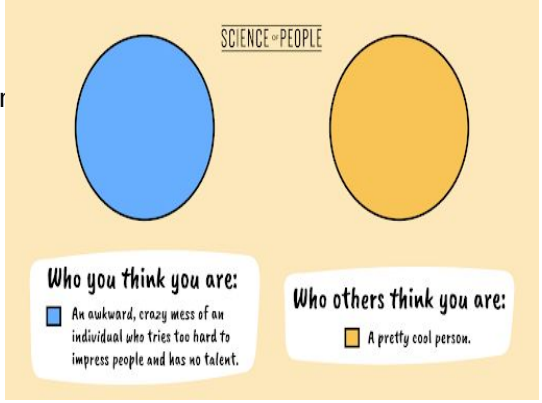
Rich Rogers @RichRogersIoT · Jan 9
"I bought my boss two copies of The Mythical Man Month so that he could read it twice as fast." - @rkoutnik

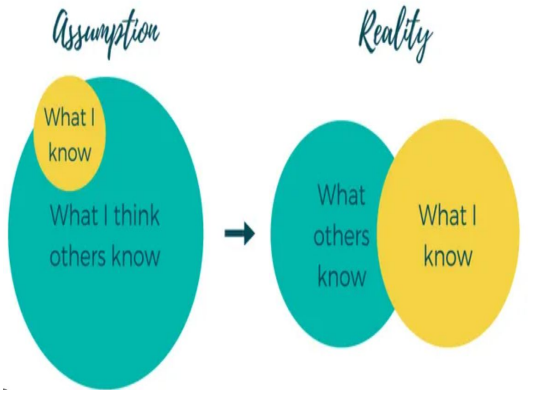
11 646 759

Stages of team development
• Forming (creating team)
• Storming (dealing with criticism)
• Norming (■guring out procedures)
• Performing (cooperating on task)

Dependence
Independence
Interdependence
Interpersonal maturity

[https://gailgazelle.com/the-imposter-syndrome-common-ar
[https://www.scienceofpeople.com/impostor-syndrome]
Have you ever felt like you are
tricking others into thinking you are
more successful and capable than
you actually are?
Imposter syndrome





[https://gailgazelle.com/the-imposter-syndrome-common-among-physicians]
Imposter syndrome
Seven true/false questions:
____ Do you ever feel you don’t deserve your achievements?
____ Do you ever worry that people will ■nd out you are secretly not worthy?
____ After a success, do you dismiss it as just good luck or timing?
____ Do you think you have tricked others into thinking you are more successful than you actually are?
____ Do you apologize for yourself even if you didn’t do anything wrong?
____ Do you think others overvalue your success?

Teamwork

- Properties of e■ective teamwork:
- Consistency
- Respect
- Inclusion
- Honesty
- E■ective dev teams are usually < 10 people.
- Feature teams are typically 2-3 people.

The ability to cooperate
and communicate
e■ectively with others to
achieve a common goal.

E■ective communication
• Communicate supportively.
• Listen openly.
• Interpret nonverbal messages.
• Give useful feedback.
• Receive feedback e■ectively.

Psychological safety: Project Aristotle

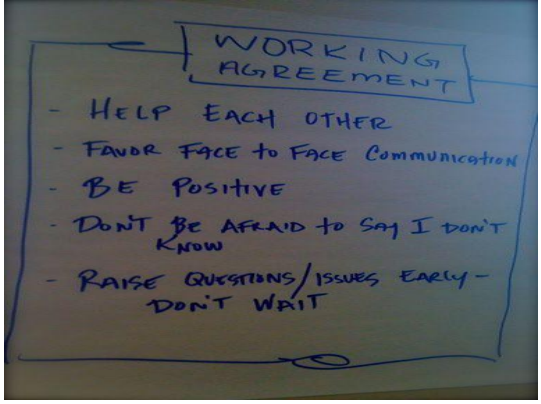
- 2012 - Google studied hundreds of teams to determine what made a high-performing team.
- Not who is on the team, but how they worked together.

Psychological Safety: perception of consequences of taking risks.

[https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html]

Team con■ict

• A contract which all members of the team agrees to abide by.
• Created by the team based on retrospective meetings.
• Details team expectations:
• Meeting frequency, email / text responsiveness.
• Git / issue tracker / test strategies.
• Everyone accountable.
• Write it down! (Required part of J0)



Teamwork agreement [https://agilefaq.wordpress.com/2007/11/21/what-is-a-team-ground-rule-or-team-working-agreement]

Project sprint planning
• Explicitly break up checkpoint into individual tasks.
• Add tasks in issue tracker. Assign to individuals.
• Ensure each task has a de■nition-of-done or sample unit test.
(tests probably exist from C0 already)
• Consider project board to track tasks.
• Generate a sprint plan in J0!

Working e■ectively as a team
• Important to be clear about roles and responsibilities.
• Scheduling co-located time: lecture x 2 + lab at minimum!
• Who is in charge of what aspects of each sprint?
• Sprint planning (identifying and assigning tasks).
• Working through tasks.
• Verifying tasks with unit tests.
• Re■ecting on the tasks and process. (J0)
• Pull requests encourage collaboration. (J1)

C1, Q0, & Labs

- C1 is in full swing.
  - All groups should have <project_team###> repositories.
  - If you do not have one, talk to your TA right away!
- Q0 is running this week in the CBTF.
  - If you haven't signed up for a slot yet, please do so.
- Labs:
  - Please remember that lab attendance is mandatory. This week you'll talk with your TA & partner together and TA alone (about J0).

Quiz overview
• Content will be everything up to the end of the prior week.
• Quizzes are cumulative.
• No notes / electronics.
• Accommodations:
• Quizzes cannot be made up.

Quiz mechanics

- 50 minute slot, designed to take < 30 minutes.
- 100% True/False.
- 1 Mark for every correct answer.
- 0 Marks for every blank answer.
- -0.5 Marks for every wrong answer (guess penalty).
- Learning areas progress will appear in Classy for any learning area that has su■cient coverage.
- This will update throughout the term after quizzes/journals.

Quiz practice / viewing

- There will be a short practice quiz available on PrairieLearn.
- This is _NOT_ meant to be comprehensive.
- Mainly there so you can get a feel for question format.
- Quiz viewing session in the CBTF next week.
- Same as test protocol: no notes, no devices.
- Viewings after every quiz (and at least one before ■nal exam).

After quiz starts: No Piazza

- Once the quiz begins (Wednesday Morning) - Please no quiz questions on Piazza.

→ Study early!

- Instead, go to OH and Labs, ask other 310 students.

Quiz sample question

Quiz sample question

Quiz sample question

Quiz sample question

Process

Software is built by teams of people

- Large systems require large teams.
- Coordinating teams of people can be challenging.
- Teams use process to provide clarity about how decisions get made and how software will evolve.
- Processes are essential de-risking mechanisms to help everyone on a team work together e■ectively.
- Requirements gathering and re■nement is a core activity for all engineers to help build consensus about is being built.

■

Managing the abstraction gap between high-level requirements and low-level implementation is a constant tension.

Software project risks

- 6 dimensions of risks:
- User: Resistance to change; con■icts between them; negative attitudes towards the project; lack of commitment; lack of cooperation.
- Requirements: Changing; inadequately identi■ed; unclear; incorrect.
- Project Complexity: New technology; high complexity; immature technology; ■rst use of technology.
- Planning & Control: Poor process; inadequate estimation; poor planning; unclear milestones; inexperienced PM's; ine■ective communication.
- Team: Lack of experience; lack of training; lack of specialised skills; lack of experience working as a team.
- Organisational Environment: Change of management during the project; unstable organisation; ongoing restructuring.

When you take on a project,
consider these risks, and
whether you can mitigate them!

Why process?
Denver Baggage
(mis)Handling
Underestimation of complexity. Complex architecture.
Changes in requirements. Underestimation of schedule an
Dismissal of advice from experts.
Failure to build in backup or recovery process to handle fai
http://calleam.com/WTPF/?page_id=2086

The story...part 1

Nov 1989	Work starts on the construction of the airport	
Oct 1990	City of Denver engages Breier Neidle Patrone Associates to analyse feasibility of building an integrated baggage system. Reports advises that complexity makes the proposition unfeasible	risk flagged; ignored.
Feb 1991	Continental Airlines signs on and plans on using Denver as a hub	scale changed
Jun 1991	United Airlines signs on and plans on using Concourse A as a hub	
Jun 1991	United Airlines engages BAE Systems to build an automated baggage system for Concourse A. BAE was a world leader in the supply, installation and operation of baggage handling equipment	
Summer 1991	Airport's Project Management team recognizes that a baggage handling solution for the complete airport was required. Bids for an airport wide solution are requested	
Fall 1991	Of the 16 companies included in the bidding process only 3 respond and review of proposals indicate none could be ready in time for the Oct 1993 opening. The 3 bids are all rejected	risks flagged; ignored
Early 1992	Denver Airport Project Management team approach BAE directly requesting a bid for the project	
Apr 1992	Denver Airport contracts with BAE to expand the United Airlines baggage handling system into an integrated system handling all 3 concourses, all airlines, departing as well as arriving flights. In addition system is to handle transfer baggage automatically. Contract is hammered out in 3 intense working sessions	hasty contract
Aug 1992	United Airlines changes their plans and cuts out plans for the system to transfer bags between aircraft. Resulting changes save $20m, but result in a major redesign of the United Airlines portion of the system. Change requests are raised to add automated handling of oversized baggage and for the creation of a dedicated ski equipment handling area	requirements change
Sep 1992	Continental requests ski equipment handling facilities be added to their concourse as well	requirements change
Oct 1992	Chief Airport Engineer, Walter Singer dies. Mr Singer had been one of the driving forces behind the creation of the automated baggage system	guru dies

The story...part 2

Jan 1993	Change orders raised altering size of ski equipment claim area and adding maintenance tracks so carts could be serviced without having to be removed from the rails	requirements change
Feb 1993	Target opening date shifted from 31 Oct 93 to 19 Dec 93 and soon thereafter to 9 Mar 94	
Sep 1993	Target opening date is shifted again, new target date is 15 May 1994	
31 Oct 1993	Original target for opening	
19 Dec 1993	Second target for opening	delays
Jan 1994	United Airlines requests further changes to the oversize baggage input area	
9 Mar 1994	Third target for opening	
Mar 1994	Problems establishing a clean electrical supply results in continual power outages that disrupt testing and development. Solution requires installation of industrial filters into the electrical system. Ordering and installation of the filters takes several months	technical challenges
Apr 1994	Airport authorities arrange a demonstration for the system for the media (without first informing BAE). Demonstration is a disaster as clothes are disgorged from crushed bags	
Apr 1994	Denver Mayor cancels 15 May target date and announces an indefinite delay in opening	more delays
May 1994	Logplan Consulting engaged to evaluate the project	
15 May 1994	Fourth target for opening	
May 1994	BAE Systems denies system is malfunctioning. Instead they say many of the issues reported to date had been caused by the airport staff using the system incorrectly	BAE blames the user
Aug 1994	System testing continues to flounder. Scope of work is radically trimmed back a Logplan's recommendation airport builds a manual tug and trolley system instea	manual system used instead
Aug 1994	City of Denver starts fining BAE $12K per day for further delays	BAE fined for overtime
28 Feb 1995	Actual opening	
Aug 2005	In order to save costs the system is scrapped in favour of a fully manual system. Mainte costs were running at $1M per month at the time.	done! but badly scrapped

What happened?

- Bad planning: Software design began only 17 months before launch.
- In Munich, engineers spent two years testing a smaller system.
- Physical problems: Most buildings were built before the baggage system was designed, meaning the baggage system had to adapt to an architecture that wasn't a good ■t (sharp turns, narrow corridors).
- Change of management: Death of the driving force of the project (you'd think this was atypical, but gurus often leave a company mid-way through a project, leaving the project somewhat stranded if planning isn't good).
- Accepting changing requirements: Alterations to baggage sizes, types, paths, etc — and the contractor said a ■rm "yes" to all these changes!
- Lack of experience: This was the ■rst time BAE had built a system like this. But for some reason they didn't choose to ask for outside advice from the Munich baggage system engineers who could have helped mitigate risk.

Projects need good plans
A software process is a structured set of activities for developing a software system.
De■nes who, what, when and how for attaining goals.

Software process

A software process is a structured set of activities to develop a software system.

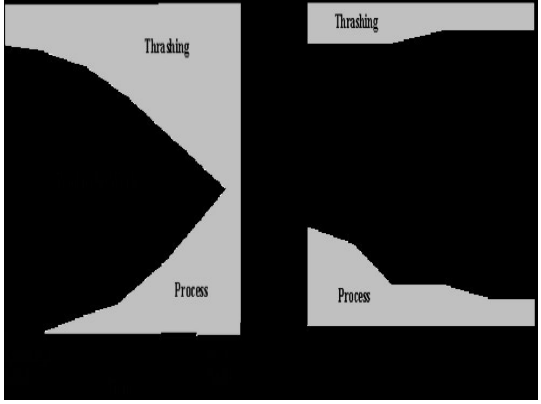
It de■nes who does what, when, and how, to reach a goal.

Processes have descriptions that discuss:

Products: the outcome of a process activity.

Stakeholders: people who care about the outcome.

Is process worth it?
“When a project has paid too little early attention to the processes it will use, by the end of a project developers feel they are spending all of their time in meetings and correcting defects and little or no time extending the software.”
“During the first few weeks of the project, the process-oriented team will seem less productive than the process-phobic team… By the end of the project, the process-oriented team will be operating at a high-speed hum, with little thrashing, and performing its processes with little conscious effort.”
[http://www.stevemcconnell.com/articles/art09.htm]



Process phases

• Many di■erent software processes.

•

Each with their own strengths and weaknesses.

• All include:

•

Requirements elicitation / gathering.

•

Architectural design.

•

Detailed design / speci■cation.

•

Implementation.

•

Integration.

•

Testing.

•

Deployment.

•

Maintenance.

The goal for each of these activities is to:

●

Mark out a clear set of steps

●

Produce tangible item(s)

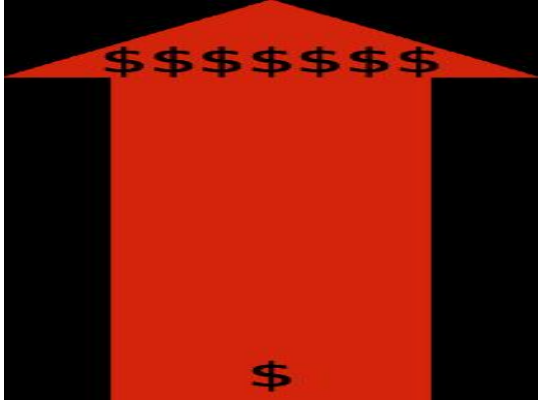
●

Allow for review of work

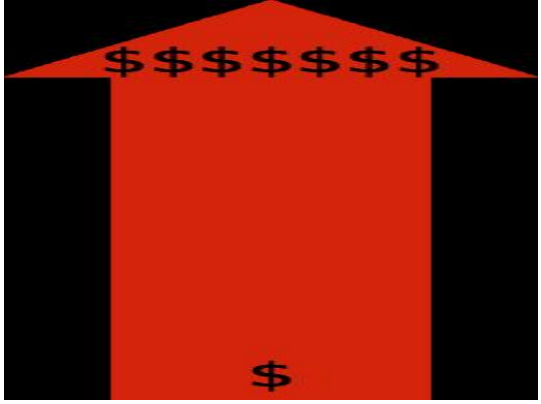
●

Specify actions to perform next

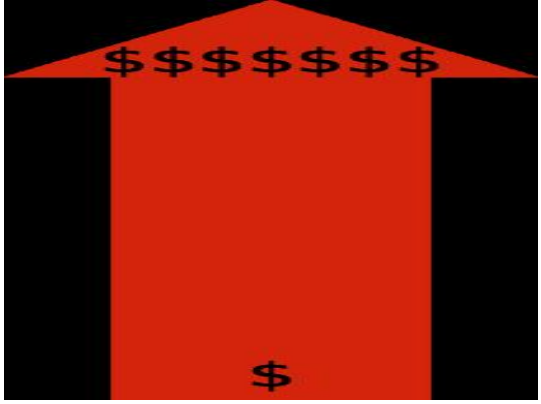
E■ciency progress a■ected process
Failure
Shipping
Coding
Compiling
Planning
Testing

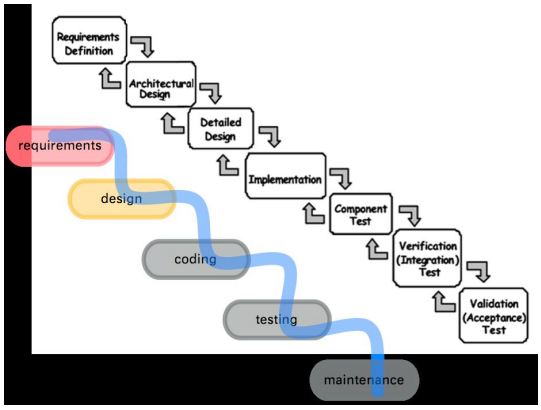


Waterfall
Failure Shipping
Coding
Compiling
Planning
Testing



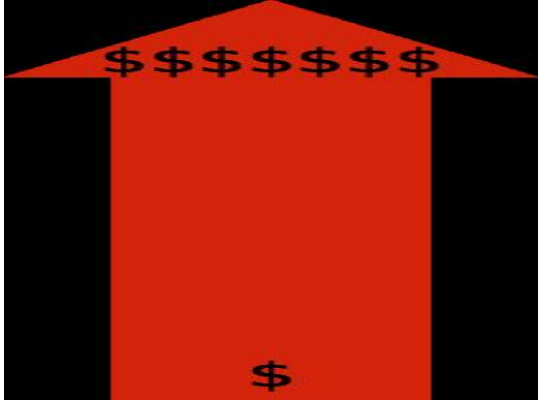
Waterfall
Failure Shipping
Coding
Compiling
Planning
Testing

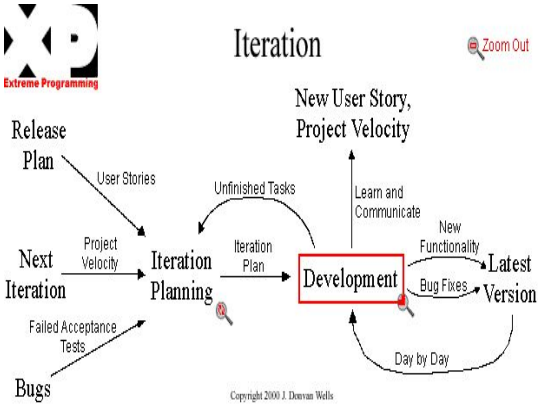




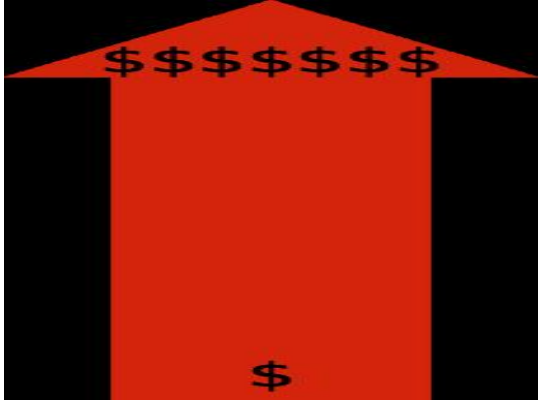
Waterfall scenario (not unusual timeline)
Requirements: 18 months. Soliciting customer feedback, creating high ■delity mockups, validating requirements.
Design: 12 months. Deriving high-level architectural description and all design info and documentation.
Implementation: 18 months. Constructing system components matching design that performs as in the requirements.
Veri■cation: 12 months. Ensure overall system performs as speci■ed in the requirements. Re■ne implementation as needed.
Maintenance: 15+ years after v1. Keeping the system going amid OS / security / device changes.

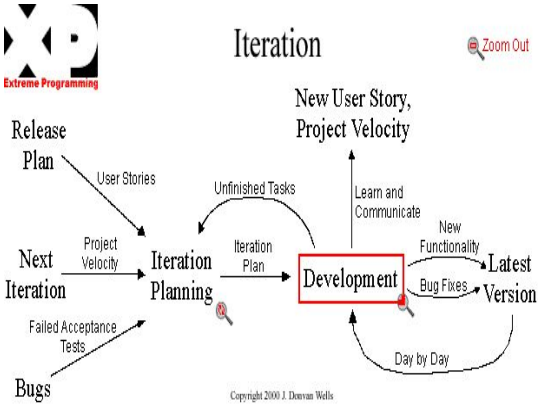
Dawn of Extreme Programming
Failure
(used in less critical
applications)
Shipping
Coding
Compiling
Planning
Testing



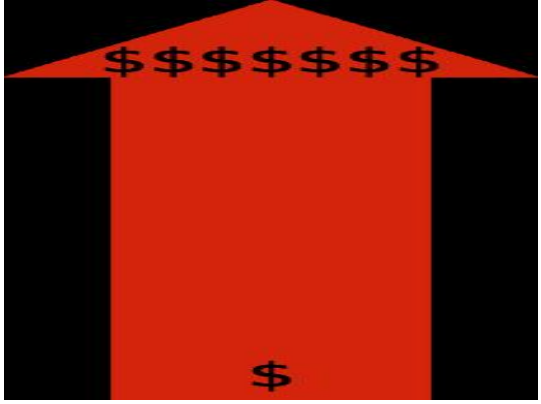


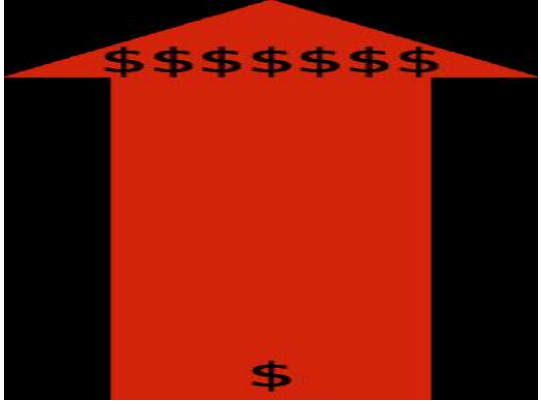
Dawn of Extreme Programming
Failure
(used in less critical applications)
Shipping
Coding
Compiling
Planning
Testing
COMMUNICATION
SIMPLICITY
FEEDBACK
COURAGE
Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes when ever [sic] they happen.





Evolution into Agile development
Failure (in non-critical systems)
Shipping
Coding
Compiling
Planning
Testing





Evolution into Agile development
Failure (in non-critical systems)
Shipping
Coding
Compiling
Planning
Testing
Individuals and interactions over processes and tools.
Working software over comprehensive documentation.
Customer collaboration over contract negotiation.
Responding to change over following a plan.
New approaches:
●User Stories
(lightweight specs)
●Test Driven Dev

Agile software process: A deeper look
Test driven design (write tests ■rst).
Emergent design: Start at the Minimum Viable Product, then grow from there, identifying duplication and introducing abstractions as needed so solve duplication.
Refactor code: rather than doing big design ■rst (make pragmatic choices along the way -- this is not a license to write terrible code) AND the architectural spike must come ■rst.

Spikes

XP introduced the concept of Spikes: short, intense, activities preceding development iterations (or sometimes ■t in between, but not typically). Used to provide insight for decision making.

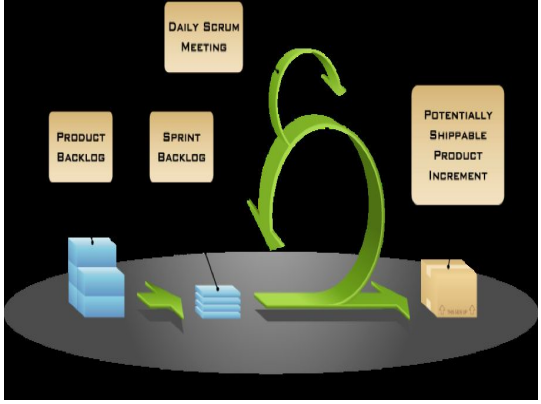
Architectural Spike: when product is being devised, decide on high- and medium-level architectures. E.g. What tech stack? What are the collaborators and subcomponents of major components (client, server)?

User Interface Spike: Before any UI development, decide on look and feel, UI framework, plan for UI expansion. These can involve initial lightweight prototyping and set the stage for design from a user perspective.

[https://www.scaledagileframework.com/spikes/]

Scrum

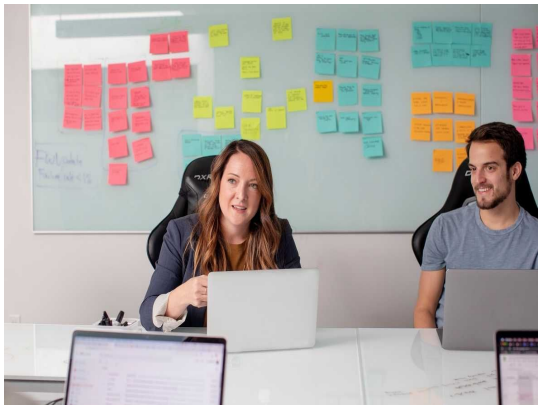
Scrum timeline
Note that you are always iterating from working product to working product, even if only some of the features are present.
SPRINT



Scrum scenario
Plan
Code & Test
Ceremonies
3 weeks

Scrum roles
Product Owner
Scrum Lead
Team
De■nes features of the product.
Prioritizes features according to market value.
Adjust features and priorities every iteration, as needed.
Facilitates Scrum process.
Helps resolve problems.
Shields team from external interferences.
NOT the manager.
Self-organizing, self-managing, cross-functional.
Developers, designers, managers, clients, etc…
7 (+/- 2) people.

Scrum ceremony: Sprint planning
What user stories will be included in the sprint?
What's the complexity of each story?
[https://letsscrumit.com/scrum-events-2-sprint-planning]



Product backlog
Sprint backlog
Burndown chart
Prioritized list of backlog items (PBIs).
PBIs specify a customer-centric feature (User Story form).
E■ort estimated by Team, priority estimated by Product Owner.
Contains list of user stories that are negotiated by team and product owner from the Product Backlog.
Negotiated PBIs broken down into speci■c tasks.
Total remaining team task hours within one sprint.
Scrum artifacts

Scrum ceremony: Daily standup
• ~15 minutes where everyone talks about:
•
What did you do yesterday?
•
What will you do today?
•
What are you blocked on?
• Team members make commitments to each other.
• Not a problem solving session.
• Not designed to blame who is behind schedule.
• E■ective way for a Scrum Master to track team progress.
• Task estimates may need to be adjusted.
You should plan to do
these either daily, or
every couple of days.

Scrum ceremony: Review + retrospective
• Team presents what it accomplished during the sprint.
• Typically takes the form of a demo of new features or underlying architecture.
• Informal; normally 2-hour prep time rule.
• As in the Daily Scrum Standup, everyone is invited.
• Review and retrospective can be one or separate meetings.
[https://illustratedagile.com/2015/01/16/3-things-to-observe-in-a-sprint-review/]

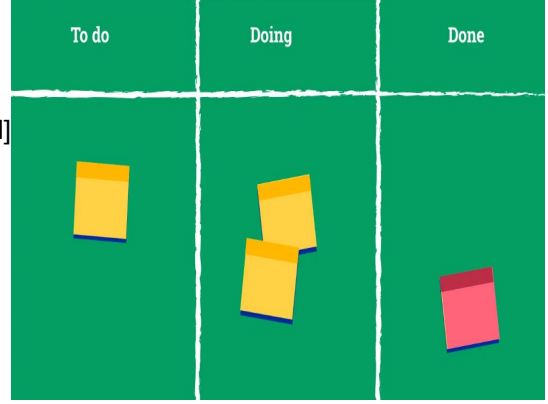
Quick re■ection on Scrum

- Sprint Planning: Work items are pulled from the product backlog.
- Scrum teams work in series of sprints → no new requirements during sprint.
- Dev team commits to implementing the work items in one sprint.
- Sprint Backlog: work items for sprint implemented by developers.
- Teams usually use a scrum/agile board to track progress.
- Result shipped to customer at end of the sprint.
- Unresolved work items are moved back to the product backlog.
- 15min daily standup meeting.
- Review / retrospective meeting at end of sprint.

From Scrum to Kanban

- Kanban: other common agile process methodology
- “You talk about work; we DO work”.
- No sprints but a continuous process without a sprint backlog.
- Pull system implemented di■erently: each column on the board has a work-in-progress limit related to the team’s capacity:
- “Keep WIP under control”
- Space freed on the board only when items move between columns.
- No speci■c release dates → up to the team to decide when to release.
- No sprint planning, or sprint reviews:
- Keeps daily standup.
- Often used for more rapid releases than scrum.
- Less rigid ceremonies.

Agile board (Kanban)
[https://www.zoho.com/sprints/what-is-a-scrum-board.html]
WIP Limit
Product
Backlog
pull



Automation

Examinable skills

- Explain the rationale behind the use of automation in the software development process.
- Identify key ideas and common actions that are automated.
- Describe the purpose and bene■ts of continuous integration, continuous development, and continuous release.
- Describe the purpose of continuous development.
- Understand why branches are used in development and the bene■ts and drawbacks of branch-based development.
- Explain how semantic versioning works.

Automation

@310-bot #c0
@310-bot #c1
@310-bot #c2
@310-bot #c3
@310-bot #check

- The 310 bots are examples of continuous integration and rely on automation to check and evaluate your systems. These are key to DevOps (described later).
- The cornerstone to these analyses is automation: they run automatically and continually to provide crucial development feedback.

Automation in the project
Automated Targets:
Build
Style
Lint
Violations
Test (public)
Test (private)
Cover

Change code
Change code
Change code
Fix DRY violation
Use Factory pattern
Make API more RESTful
Change code
310 lectures so far focused on the code dimension: processes/design/implementation/maintenance of.. code.
Coding and deploying
Code

Change code
Change code
Change code
Fix DRY violation
Use Factory pattern
Make API more RESTful
Change code
Deploy
Code on its own is just half the story. Code must be built into a release and deployed into production for users to bene■t from it.
Code
Coding and deploying

Change code
→ version’
Actions →
Continuous Integration loop

Change code
→ version’
Get code &
dependencies
Actions →
Continuous Integration loop

Change code
→ version’
Get code &
dependencies
Actions →
← Feedback
Continuous Integration loop

Change code
→ version’
Get code &
dependencies
Build
Continuous Integration loop
Actions →
← Feedback

Change code
→ version’
Get code &
dependencies
Build
Continuous Integration loop
Actions →
← Feedback

Change code
→ version’
Get code &
dependencies
Build
Run tests
Continuous Integration loop
Actions →
← Feedback

Change code
→ version’
Get code &
dependencies
Build
Run tests
Continuous Integration loop
Actions →
← Feedback

Change code
→ version’
Get code &
dependencies
Build
Run tests
…
●Repeatable
●Reliable
Continuous Integration loop
Actions →
← Feedback

• Automated pipeline:
•
Triggered by code change events.
• Ensures absence of obvious build issues and con■guration.
issues (e.g., dependencies all checked in).
• Ensures tests are executed.
• May encourage more tests.
• Can run checks on di■erent platforms, security checks…
Continuous Integration

Continuous Integration

• Automated pipeline:

•

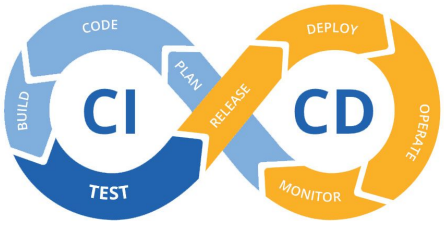
Triggered by code change events.

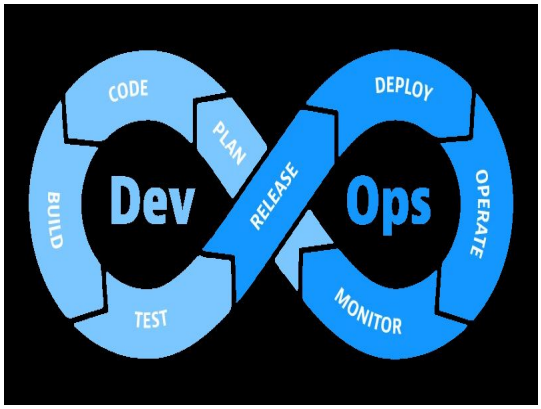
• Ensures absence of obvious build issues and con■guratic
issues (e.g., dependencies all checked in).

• Ensures tests are executed.

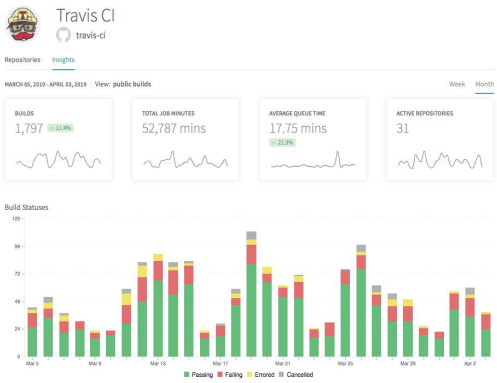
• May encourage more tests.

• Can run checks on di■erent platforms, security checks…

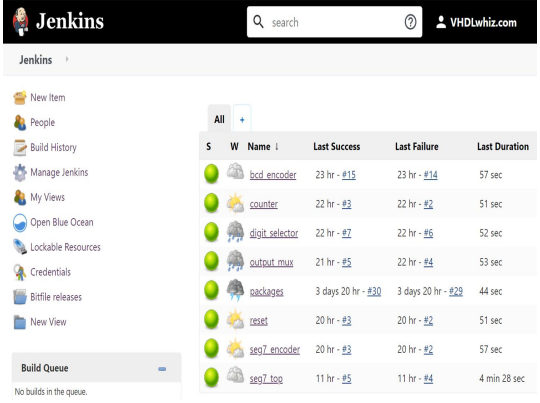




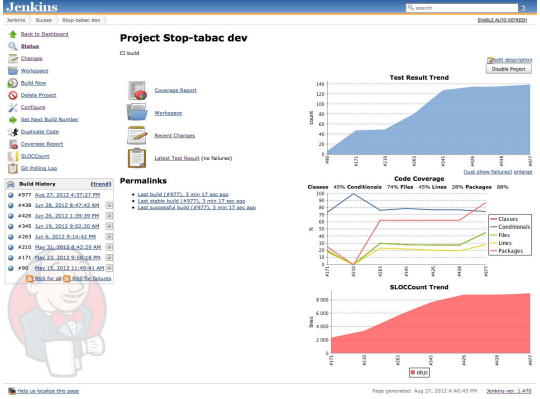
CI Tools: Travis CI
[https://blog.travis-ci.com/2019-04-03-travis-ci-insights]
[https://app.circleci.com/pipelines/github/ubccpsc/classy]



CI Tools: Jenkins
[https://vhdlwhiz.com/jenkins-for-fpga/]



CI Tools: Jenkins
[https://coveralls.io/github/ubccpsc/classy?branch=master]



From integration to release

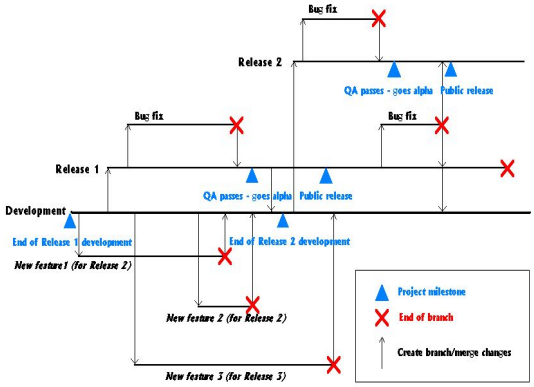
- Release management:
- Versioning
- Branches
- Goal: Continuous releases.
- Every merge-to-main should be a release.

Semantic versioning
• Associated with some public API.
• For a version number MAJOR.MINOR.PATCH, increment the:
•
MAJOR when you make incompatible API changes,
•
MINOR when you add functionality in a backwards-compatible manner, and
•
PATCH when you make backwards-compatible bug ■xes.
• Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.
•
~1.2.3 → 1.2.*; ^5.4.3 → 5.*.*
[https://semver.org/]

Why semantic versioning?

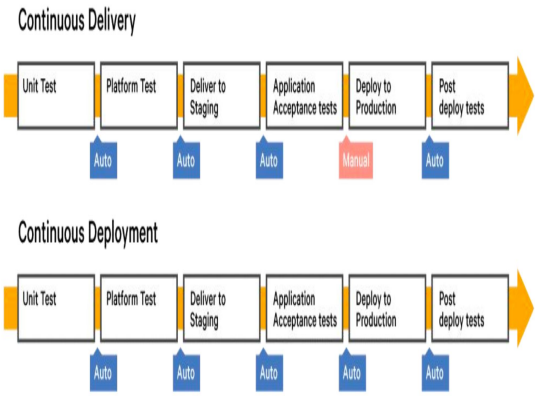
- Consider a library called “Firetruck.” It requires a Semantically Versioned package named “Ladder.”
- At the time that Firetruck is created, Ladder is at version 3.1.0. Since Firetruck uses some functionality that was ■rst introduced in 3.1.0, you can safely specify the Ladder dependency as greater than or equal to 3.1.0 but less than 4.0.0.
- Now, when Ladder version 3.1.1 and 3.2.0 become available, you can release them to your package management system and know that they will be compatible with existing dependent software.

Branch management

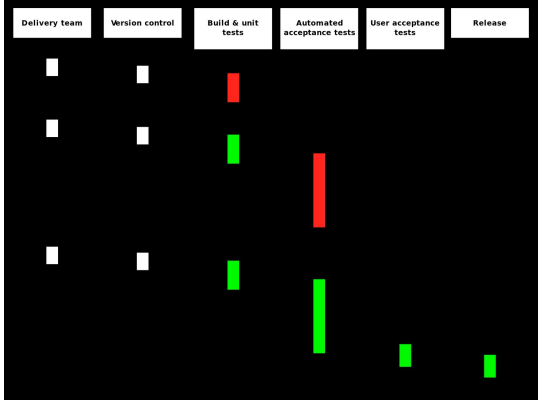


●
Traditional View (Boxed Software):
•
Working toward ■xed release date, QA heavy before release.
•
Release and move on.
•
Fix post-release defects in next release or through expensive patches.
●
Frequent releases:
•
Incremental updates delivered frequently (weeks, days, ...), e.g. Browsers.
•
Automated updates (“patch culture”; “updater done? ship it”).
●
Hosted software (SaaS : software as a service):
•
Frequent incremental releases, hot patches, di■erent versions for di■erent customers, customer may not even notice update.
From release to continuous release

Continuous delivery/deployment



[https://commons.wikimedia.org/wiki/File:Continuous_Delivery_process_diagram.svg]
Continuous delivery process



Automation facilitates rapid progress

- All mature teams use automation to improve their process.
- Automation:
  - De-risks development through continual evaluation.
  - Facilitates rapid problem identi■cation.
  - Eases rollback: past states are well understood.
  - Decreases resistance to release (individual steps well understood, possibility of rollback known).
- Increases team trust; results are visible, and often must meet prede■ned measures to qualify for code review.
- A small investment in automation almost always pays dividends in future time savings.

Requirements

Examinable skills
Write user stories with the correct format, and meaning.
Write unambiguous de■nitions of done.
Identify why some user stories are not good user stories (because they are ambiguous, or too large, or not testable, for instance -- INVEST principles).
Explain why, as professionals, de■nitions of done are important.
Explain how de■nitions of done drive test suites.
Explain the di■erence between a story and a task, and identify which is which.
Explain how requirement lengths are motivated to some extent by technological and economic concerns.
Be able to di■erentiate and elicit functional requirements and quality attributes.
Explain the di■erence between a requirement and a speci■cation.
Explain how user stories relate the problem domain to the solution domain to capture client expectations.
Requirements and
Specifications

A ‘wicked problem’ is one that can only be clearly de■ned by solving it.

‘This paradox implies that one must solve a problem once to de■ne it and then solve it again to create a solution that works.’
—Peters and Tripp

Wicked characteristics
• No de■nitive formulation.
• No stopping rule.
• Solutions not true-or-false, just good-or-bad.
• No ultimate test of a solution.

●Four main stakeholders: students, TAs, instructors, UBC.
●Di■ering goals:
■
Students: Maximize grade while developing marketable skills.
■
TAs: Help students develop self mastery of material.
■
Instructor: Build deep understanding of core concepts.
■
UBC: Maintain integrity of the learning environment.
Curriculum design as a wicked problem

●Four main stakeholders: students, TAs, instructors, UBC.
●Di■ering goals:
■
Students: Maximize grade while developing marketable skills.
■
TAs: Help students develop self mastery of material.
■
Instructor: Build deep understanding of core concepts.
■
UBC: Maintain integrity of the learning environment.
●Di■ering constraints:
■
Students: Finite time budget; not all courses are of core interest.
■
TAs: Fixed availability; restrictions from instructor.
■
Instructor: Program learning outcomes; course scalability.
■
UBC: Instructor / TA availability, student abilities in the workplace.
Curriculum design as a wicked problem

Curriculum design as a wicked problem

- Four main stakeholders: students, TAs, instructors, UBC.
- Di■ering goals:
  - ■ Students: Maximize grade while developing marketable skills.
  - ■ TAs: Help students develop self mastery of material.
  - ■ Instructor: Build deep understanding of core concepts.
  - ■ UBC: Maintain integrity of the learning environment.
- Di■ering constraints:
  - ■ Students: Finite time budget; not all courses are of core interest.
  - ■ TAs: Fixed availability; restrictions from instructor.
  - ■ Instructor: Program learning outcomes; course scalability.
  - ■ UBC: Instructor / TA availability, student abilities in the workplace.
- Goal/constraint mismatch leads to obvious tension:
  - ■ Students: Maximize alignment of material with assessment.
  - ■ TAs: Minimize marking to maximize student contact.
  - ■ Instructor: Validate understanding and synthesis over memorization.
  - ■ UBC: Ensure an equitable and consistent process.

‘The appropriate way to tackle wicked problems is to discuss them. Consensus emerges through the process of laying out alternative understandings, competing interests, priorities, and constraints’
—Mary Poppendieck

Requirement engineering lifecycle

- Elicitation:
  - The process by which requirements are gathered.
  - Using whatever sources of information are available: client, users, observation, videos, documents, interviews, etc.
- Validation:
  - Have we elicited and documented the right reqs?
  - Are they consistent?

[https://cs.uwaterloo.ca/~dberry/ATRE/Slides/IcebergSlides.pdf]

[https://cs.uwaterloo.ca/~dberry/ATRE/Slides/IcebergSlides.pdf]
Requirement engineering lifecycle
Idealized RE process

[https://cs.uwaterloo.ca/~dberry/ATRE/Slides/IcebergSlides.pdf]
Idealized RE process
Requirement engineering lifecycle
Reality
Req. engineering

Software will always be hard because…
complexity
conformity
changeability
invisibility

●Functional requirements:
■Speci■es what the system should do.
●Quality attributes (also known as non-functional reqs):
■Properties that the product must have.
■Usually described using adjectives.
■Often strongly impact system success.
Two main kinds of requirements

●Functional requirements:
■Speci■es what the system should do.
●Quality attributes (also known as non-functional reqs):
■Properties that the product must have.
■Usually described using adjectives.
■Often strongly impact system success.
●Quality attribute examples:
■Security: e.g., con■dentiality, integrity, availability, privacy.
■Reliability: e.g., durability, recoverability, safety.
■Performance: scalability, elasticity, capacity.
■Legal: e.g., compliance, regulatory, auditability.
■Usability: e.g., learnability, discoverability, accessibility.
■Others: e.g., a■ordability, debuggability, evolvability.
Two main kinds of requirements

●Functional requirements:
■Speci■es what the system should do.
●Quality attributes (also known as non-functional reqs):
■Properties that the product must have.
■Usually described using adjectives.
■Often strongly impact system success.
●Quality attribute examples:
■Security: e.g., con■dentiality, integrity, availability, privacy.
■Reliability: e.g., durability, recoverability, safety.
■Performance: scalability, elasticity, capacity.
■Legal: e.g., compliance, regulatory, auditability.
■Usability: e.g., learnability, discoverability, accessibility.
■Others: e.g., a■ordability, debuggability, evolvability.
Two main kinds of requirements
Quality attributes are
only useful if they can
be measured.

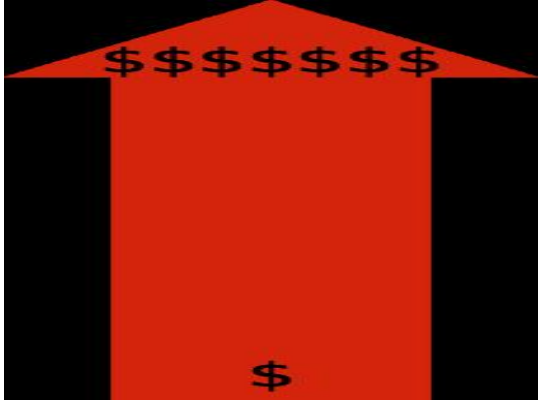
●Functional requirements:
■Speci■es what the system should do.
●Quality attributes (also known as non-functional reqs):
■Properties that the product must have.
■Usually described using adjectives.
■Often strongly impact system success.
●Quality attribute examples:
■Security: e.g., con■dentiality, integrity, availability, privacy.
■Reliability: e.g., durability, recoverability, safety.
■Performance: scalability, elasticity, capacity.
■Legal: e.g., compliance, regulatory, auditability.
■Usability: e.g., learnability, discoverability, accessibility.
■Others: e.g., a■ordability, debuggability, evolvability.
Two main kinds of requirements
There is often tension
between di■erent
Quality attributes.

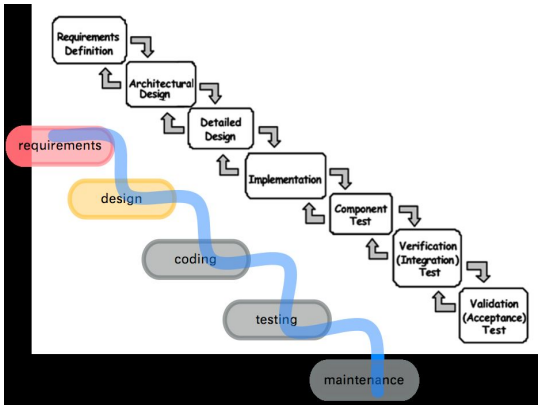
Requirements
Evolution

Requirements size alignment

- Large requirements (comprehensive reqs / formal specs): Needed for situations where coding, compiling, and testing are expensive. (still true for e.g., aircraft, medical devices).
- Medium requirements (use cases): Arose when coding, compiling, and shipping became cheaper, but were NOT free.
- Small requirements (user stories): Now an option when coding, testing, shipping are e■ectively free (now, humans are the most expensive element).

Waterfall process
Failure Shipping
Coding
Compiling
Planning
Testing
Cost

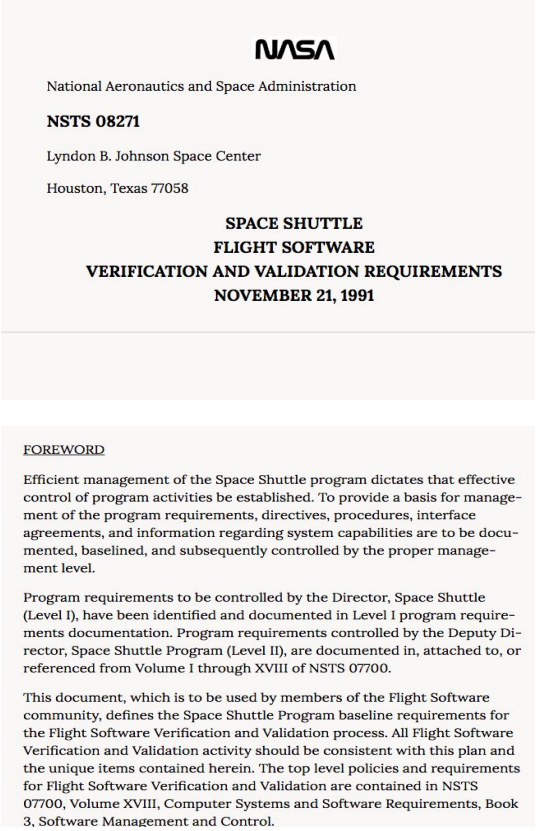




Requirements: Starting point

- When computation was expensive, people took a long time planning their implementation, and that involved writing lengthy requirements.
- Another reason was that systems were applied to high-risk problems, so fully specifying the requirements was important.
- Hence:
  - Requirements used to be very very large documents.
  - And used to be written very formally.
  - Actually they still are, in situations where life is at risk!

First came large requirements
[https://www.nap.edu/read/2222/chapter/16#160]



NASA

National Aeronautics and Space Administration

NSTS 08271

Lyndon B. Johnson Space Center

Houston, Texas 77058

SPACE SHUTTLE
FLIGHT SOFTWARE
VERIFICATION AND VALIDATION REQUIREMENTS
NOVEMBER 21, 1991

FOREWORD

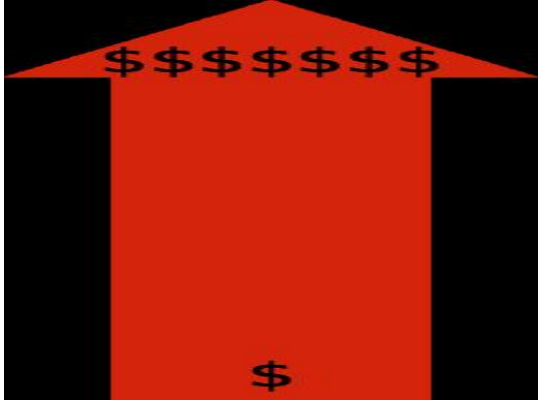
Efficient management of the Space Shuttle program dictates that effective control of program activities be established. To provide a basis for management of the program requirements, directives, procedures, interface agreements, and information regarding system capabilities are to be documented, baselined, and subsequently controlled by the proper management level.

Program requirements to be controlled by the Director, Space Shuttle (Level I), have been identified and documented in Level I program requirements documentation. Program requirements controlled by the Deputy Director, Space Shuttle Program (Level II), are documented in, attached to, or referenced from Volume I through XVIII of NSTS 07700.

This document, which is to be used by members of the Flight Software community, defines the Space Shuttle Program baseline requirements for the Flight Software Verification and Validation process. All Flight Software Verification and Validation activity should be consistent with this plan and the unique items contained herein. The top level policies and requirements for Flight Software Verification and Validation are contained in NSTS 07700, Volume XVIII, Computer Systems and Software Requirements, Book 3, Software Management and Control.

Problems with large requirements

- Very difficult for a client to play out the behaviour based on the description because the description is so in-depth.
- These are long, almost legalese documents that take a long time to convert into a specification or detailed design.



Extreme Programming shifts goals
Failure
(used in less critical
applications)
Shipping
Coding
Compiling
Planning
Testing
COMMUNICATION
SIMPLICITY
FEEDBACK
COURAGE
Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes when ever [sic] they happen.

Medium requirements: Use cases

Use Case 1: Buy something

Context of use: Requestor buys something through the system, gets it.

Scope: Corporate - The overall purchasing mechanism, electronic and non-electronic, as seen by the people in the company.

Level: Summary

Preconditions: none

Success End Condition: Requestor has goods, correct budget ready to be debited.

Failed End Protection: Either order not sent or goods not being billed for.

Primary Actor: Requestor

Trigger: Requestor decides to buy something.

Main Success Scenario

1. **Requestor**: initiate a request
2. **Approver**: check money in the budget, check price of goods, *complete request for submission*
3. **Buyer**: check contents of storage, find best vendor for goods
4. **Authorizer**: *validate approver's signature*
5. **Buyer**: *complete request for ordering, initiate PO with Vendor*

And on...

6. Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design)
7. Receiver: *register delivery*, send goods to Requestor
8. Requestor: *mark request delivered*.

Extensions
1a. Requestor does not know vendor or price: leave those parts blank and continue.
1b. At any time prior to receiving goods, Requestor can change or cancel the request.
Canceling it removes it from any active processing. (delete from system?)
Reducing price leaves it intact in process.
Raising price sends it back to Approver.
2a. Approver does not know vendor or price: leave blank and let Buyer fill in or call back.
2b. Approver is not Requestor's manager: still ok, as long as approver signs
2c. Approver declines: send back to Requestor for change or deletion
3a. Buyer finds goods in storage: send those up, reduce request by that amount and carry on.
3b. Buyer fills in Vendor and price, which were missing: gets resent to Approver.
4a. Authorizer declines Approver: send back to Requestor and remove from active processing.
5a. Request involves multiple Vendors: Buyer generates multiple POs.
5b. Buyer merges multiple requests: same process, but mark PO with the requests being merged.
6a. Vendor does not deliver on time: System does *alert of non-delivery*
7a. Partial delivery: Receiver marks partial delivery on PO and continues
7b. Partial delivery of multiple-request PO: Receiver assigns quantities to requests and continues.
8a. Goods are incorrect or improper quality: Requestor does *refuse delivered goods*. (what does this mean?)
8b. Requestor has quit the company: Buyer checks with Requestor's manager, either *reassign Requestor*, or return goods and *cancel request*.

And on!

Deferred Variations
none

Project Information

Priority	Release Due	Response time	Freq of use	
Various	Several		Various	3/day

Calling Use Case: none
Subordinate Use Cases: see text
Channel to primary actor: Internet browser, mail system, or equivalent
Secondary Actors: Vendor
Channels to Secondary Actors: fax, phone, car
Open issues

When is a canceled request deleted from the system?
What authorization is needed to cancel a request?
Who can alter a request's contents?
What change history must be maintained on requests?
What happens when Requestor refuses delivered goods?

Problems with medium requirements

- Still di■cult for a client to play out the behaviour based on the description because the description is so in-depth.

■

Fairly formal descriptions.

- Still interconnected -- they would refer to one another! “Buy Something” might refer to the “Procure goods” use case.
- Would weave together multiple stakeholders: requestor/ buyer/ vendor/…

Smaller requirements→SMALLER use cases

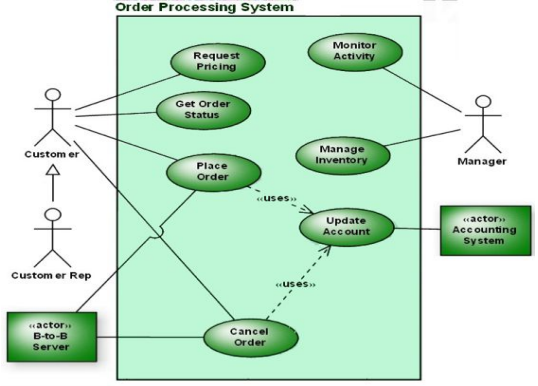
Use Case 1: Buy something

The Requestor initiates a request and sends it to her or his Approver. The Approver checks that there is money in the budget, check the price of the goods, completes the request for submission, and sends it to the Buyer. The Buyer checks the contents of storage, finding best vendor for goods. Authorizer: validate approver's signature . Buyer: complete request for ordering, initiate PO with Vendor. Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design). Receiver: register delivery, send goods to Requestor. Requestor: mark request delivered..

At any time prior to receiving goods, Requestor can change or cancel the request. Canceling it removes it from any active processing. (delete from system?) Reducing the price leaves it intact in process. Raising the price sends it back to Approver.

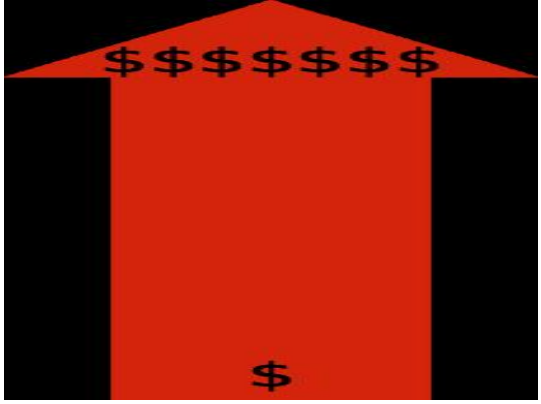
Then PICTORIAL requirements

- Use case diagrams show packaging and decomposition of use cases not their content.
- Each ellipse is a use case:
  - Only top-level services should be shown.
  - Not their internal behaviour.
- Actors can be other systems.
- The system (black outline) can be an actor in other use case diagrams.
- Are not enough by themselves:
  - Must individually document use cases.



Problems with use cases

- STILL di■cult for a client to play out the behaviour based on the description because it is so in-depth.
- This contributes to a mismatch between client expectations and what the developer does.
- Intermingles the client and solution domains:
  - Problem domain: The needs of the client.
  - Solution domain: How it will be implemented.

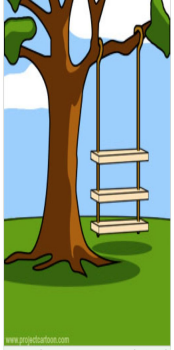


Agile development
Failure (in non-critical systems)
Shipping
Coding
Compiling
Planning
Testing
Individuals and interactions over processes and tools.
Working software over comprehensive documentation.
Customer collaboration over contract negotiation.
Responding to change over following a plan.
New approaches:
●User Stories
(lightweight specs)
●Test Driven Dev

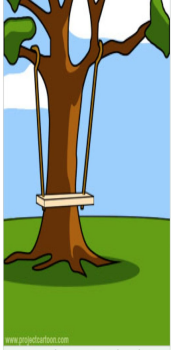
Specs

Functional
requirements
(what the system
should do).
Quality attributes
(constraints the
system should meet).
Software speci■cations describe system behaviour

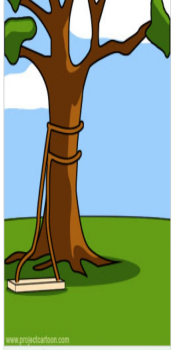
• A perfect system is worthless if it solves the wrong problem.
• It is di■cult to create specs that are:
• Complete
• Consistent
• Precise
• Concise
Specs describe what to do (but not how)



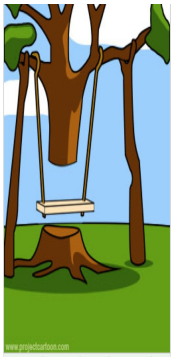
How the customer explained it



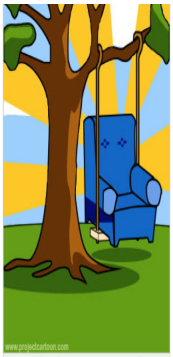
How the project leader understood it



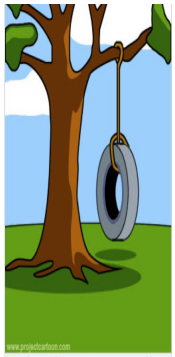
How the programmer wrote it



How the analyst designed it



How the business consultant described it



What the customer really needed

• A speci■cation:
• Connects customer and engineer.
• Ensures parts of the implementation work together.
• De■nes the correctness of the implementation.
• Therefore, everyone must understand the spec
• Designers, developers, testers, managers, ops, customers…
• Good speci■cations are essential for a project to be successful.

Speci■cations matter

Spec
Discussion

• A speci■cation:
• Connects customer and engineer.
• Ensures parts of the implementation work together.
• De■nes the correctness of the implementation.
• Therefore, everyone must understand the spec
• Designers, developers, testers, managers, ops, customers…
• Good speci■cations are essential for a project to be successful.

Speci■cations matter

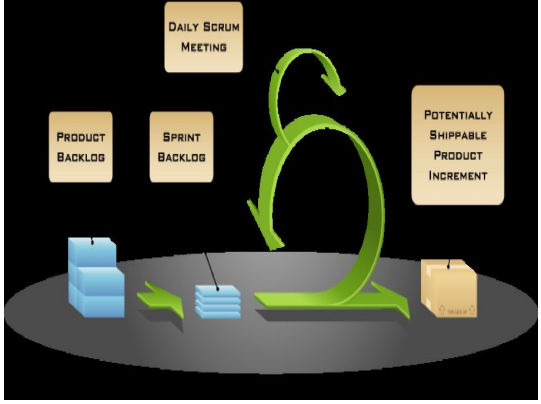
Is it complete?
Is it concise?
Is it consistent?
Is it precise?
Writing speci■cations is hard.

●Complete:
■Is the number memory changed by TALK during a call?
●Consistent:
■What do you show if a call arrives when reviewing?
●Precise:
■Does send(string) send tones to the audio bus?
●Concise:
■Did everyone understand it?
Why? The technical view

• This is where the hard decisions, that balance between what you want to have and what you can have in a system, need to happen.
• Challenges:
• Include all stakeholders.
•
Make decisions smoothly and rapidly.
• Satisfy as many constraints as possible.
Why? The management view

User
Stories
(small requirements)

Scrum timeline
Note that you are always iterating from working product, to working product, even if only some of the features are present.
SPRINT



USER
STORIES
Role Goal Bene■t
Sometimes just called the “User Story”.
Engineering Tasks
De■nitions of Done
Sometimes called Acceptance Criteria.
(Solution to Role-Goal-Bene■t)

USER
STORIES
Role Goal Bene■t
Sometimes just called the “User Story”.
De■nitions of Done
Sometimes called Acceptance Criteria.
(Solution to Role-Goal-Bene■t)
Engineering Tasks
Contract:

USER
STORIES
Role Goal Bene■t
Sometimes just called the “User Story”.
De■nitions of Done
Sometimes called Acceptance Criteria.
(Solution to Role-Goal-Bene■t)
Engineering Tasks
Engineering
details:

ROLE
GOAL
BENEFIT
DEFINITIONS
OF DONE
ENGINEERING
TASKS
Which users
to test with
What
behaviour to
build
What
tests to
specify
Informs
What
needs to
be done
STORY
POINTS
Informs
# of hours
it will take
Helps prioritize
user stories
Connecting user stories + soft. engineering

Role Goal Bene■t statement
●Have a Role (the speci■c type of user expressing the need).
●Have a Goal (the desired behaviour).
●Have a Bene■t (the outcome of the behaviour).
RGB statement:
As a customer,
… I want to be able
to buy something
… and then get it.
●
As a user, I want to search for contacts so I can message them.
●
As a customer, I want to search for product items, so I can buy them.
●
As an employer, I want to post a job on the website so people can apply for it.

RGB: As a shopper, I want to be able to buy something and then view past purchases so that I can spend money on the site.
De■nition of done: User clicks the button buy, and it appears in their purchased items, and is shipped to their home and the user will see the money deducted from their account.
User story examples

User story examples

RGB: As a shopper, I want to be able to buy something and then view past purchases so that I can spend money on the site.

De■nition of done: User clicks the button buy, and it appears in their purchased items, and is shipped to their home and the user will see the money deducted from their account.

Bad User story: As a buyer, When I'm told that I'm not approved for purchase by the system, I want to be able to click "request approval" button (solution domain!) and then receive con■rmation that the approval request has been sent. (no bene■t! How valuable is this?)

De■nition of done: User is seeing "not approved, and clicks "request approval", and this triggers a react function which makes user's ID appear in the list of approval requests, and an email is sent back to the user that their request is in processing

(DoD is user-level: should not mention code; DoD is client-oriented solution domain)

Not Role Goal Bene■t statements:
• Implement contact list view ContactListView.ts.
• De■ne the product table database schema.
• Automate the job posting algorithm.
• Refactor the code to make it more readable.
These are engineering tasks.

Good de■nitions of done

- These are your contracts with your clients.
- This is how you know you have completed a user story, and can mark it resolved.
- It’s like a sequence diagram that explains how the features work.
- This is how you know whether you can test your user story (if you can’t, you need a di■erent user story!!).

De■nition of done: User clicks the button buy, and the item appears in their purchased items, and is shipped to their home.

Engineering tasks

- Developers break down user stories into engineering tasks.
- These are NOT from a user perspective -- they are just things that need to be done to get the work completed (■nish the parser; investigate the JSON library; set up the database; setup mocks for testing; etc).
- Based on those tasks, the developers estimate how much time the story will take.

[https://www.mountaingoatsoftware.com/blog/the-di■erence-between-a-story-and-a-task]

Estimating story points

●A story point basically corresponds to developer-hour of w

●Estimation traditionally was made by a developer, guessi (based on experience) how long a story would take them.

●Estimates are moving more towards an ontology:

■

E.g., using classi■cations rather than opaque experience.

■

Key especially for young teams or new domain.

Single location

Multiple locations

Simple change

1

2

Complex change

3

5

Where each of these ratings would have some standard number of hours associated

Burndown chart

Why Fibonacci?
“I think if it was linear, it wouldn't represent how complex so
think it's more important for the team to have a uni■ed de■
represents. Generally we never pull in [to the sprint] things
pull in research tickets [AKA: a research spike] or break do
an indicator that too much is unknown”
Case of a local estimation approach

1 - Trivial cosmetic change in very few places

2 - Many trivial changes across a project (following existing patterns, adding properties to existing objects etc.)

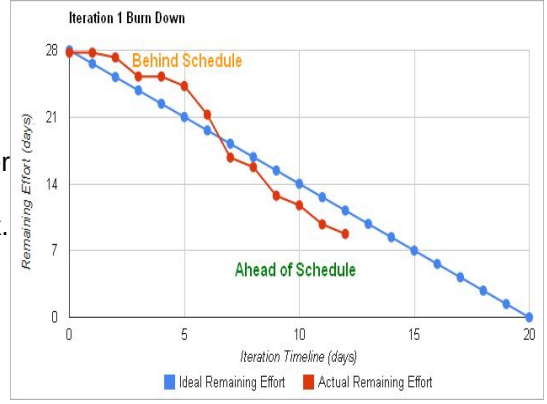
3 - Changes that require a new design or concept and are straight forward to implement in the existing design

5 - Changes that require a new design or concept and require some rework of the existing design OR are very complicated.

8 - Changes that require a new design or concept and require some rework of the existing design AND are very complicated.

Estimating story points

- A story point basically corresponds to an hour of developer
- Estimation is important to ensure team can complete work.
- Also widely used to see if team is ahead/behind schedule.
- Enables greater responsiveness / team awareness.



From user stories to epics to themes

• User stories are often related, but independent.

• Epics group together related user stories:

•

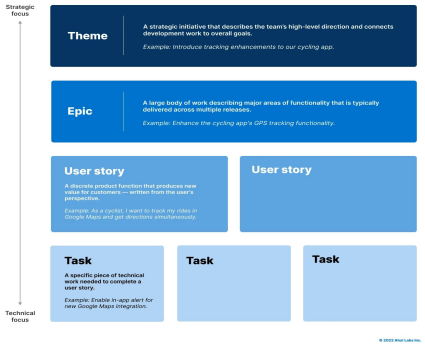
Usually delivered over multiple sprints.

•

Grouping of stories that share an overall goal.

• Themes group epics and describe even higher-level objectives.

https://www.aha.io/roadmapping/guide/agile/themes-vs-epics-vs-stories-vs-tasks



Good things about small requirements

- Clear linkage between problem domain and solution domain because de■nitions of done and succinct descriptions.
- Decrease risk:
  - ■DoD makes sure we are building the right thing.
  - ■Small increments mean failure detected quicker.
- These are still legal documents!
- They do not have hierarchy the way prior requirements did (though clusters of them might make sense).
- Great way to distribute work between team members!

Assessing user stories: INVEST

• Agile approaches depend on user stories.

• If they are ill de■ned or incomplete, the story will be built poorly and any time estimates are meaningless.

• INVEST provides a user story checklist:

•
Independent

•
Negotiable

•
Valuable to users or customers

•
Estimatable

•
Small

•
Testable

INVEST: Independent

User stories should be independent of one another, but obviously they're not implemented in a vacuum.

VARIANT: In the some cases they do have to be grouped, but that is not the norm. It's best when user stories are not blocking each other, in terms of development or testing.

User requirements used to be hierarchical like this:

Edit Text
Put red squiggles under misspelled words
Change text font
Copy and Paste Text

To implement the "Edit Text" feature, you would have to implement all the sub features. Only then would the product be done and ready to ship.

INVEST: Independent

User stories should be independent of one another, but obviously they're not implemented in a vacuum.

VARIANT: In the some cases they do have to be grouped, but that is not the norm. It's best when user stories are not blocking each other, in terms of development or testing.

User requirements used to be hierarchical like this:

Edit Text
Put red squiggles under misspelled words
Change text font
Copy and Paste Text

To implement the "Edit Text" feature, you would have to implement all the sub features. Only then would the product be done and ready to ship.

That hierarchy vanished, leaving independent user stories

Put red squiggles under misspelled words
Change Text Font
Copy and Paste Text

Each of these user stories can be implemented independently. They may be prioritised di■erently, and their engineering tasks may build on each other, but they can be thought of without thinking of them as a collection. The product can ship with one of these, without having to have the rest.

Independent does not mean unsequenced
User stories do need to build upon one another, but within a
two user stories should ideally not be dependent on each o
blocked by one another.
Sequencing the user stories

INVEST: Negotiable

Users didn’t used to have a lot of bargaining power when talking about the requirements speci■cations for their systems. This was mainly down to the fact that the requirements were di■cult to read. Users would not be able to visualize how a feature would work -- they would assume they knew, and they might well be wrong. At the time the ■nished product was released, there were often unpleasant surprises.

Negotiable means that clients have to be able to fully understand and critique how a feature will work. This means that they are clearly written. The time estimates also have to be present so a customer can decide whether the amount of time spent on a particular story is worth it to them.

INVEST: Valuable
De■nitely best if it is providing value to a customer. They are the ones paying for the development e■ort.
This forces thinking from the client's perspective. For example, refactoring needs to be put into a story that has client value. (OR it could be put into a spike!)
VARIANT: In practice, this is mostly the case, but exceptions do have to be made sometimes.

INVEST: Estimatable

Estimatable means that the user story should be written precisely enough that a developer can estimate how long it will take. This has huge value to the developer because projects can run over time if requirements are vague. “Not realising what they really wanted” is not an option here, because the work must be estimatable.

INVEST: Small

This is in direct contrast to “Big”. Which is how requirements used to be. They used to be WHOLE FEATURES that spanned the entire lifetime of the product. So ALL of the “Edit” feature, for instance, would chunk in at once from the user’s perspective, and so “Edit” was one massive requirements document, with a thousand sub-requirements that might take a year to complete. Now, stories are added incrementally to products (in clusters, but you get the idea), and so they are small enough to ■t within the work span of a couple of weeks (a sprint). If it’s larger than that, you need to split it up.

Some teams strive to get size down to a single day -- but overly granulating can lead to undesired dependencies between stories.

INVEST: Testable

This is how you know you're done, and that you've built what you said you were going to build. You need to know how to test, what to test, and what it means when all those tests pass (and this refers to either user or unit tests).

Some things are NOT testable:

•

For instance "Make the user happy" is not testable. You can't actually measure whether someone is happy.

• "Help the user select x things in y seconds" is testable. You can see whether, on average, over a certain population, users are generally able to be able to ■nish that quickly.

User story INVEST walkthrough
Architectural context: Company email system with a database of contacts.
User story:

User story INVEST walkthrough
Architectural context: Company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
De■nitions of done:

User story INVEST walkthrough
Architectural context: Company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
De■nitions of done: Employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent?
- Negotiable?
- Valuable?
- Estimatable?
- Small?
- Testable?

Architectural context: company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent: Yes? Dependency on previous functionality to exist – what search fn exists, need message composing, need a DB. Needs more clarity on these dependencies.
- Architectural context: company email system with a database of contacts.
- User story: As an employee, I want to search for contacts by name so I can message them.
- Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent: Yes? Dependency on previous functionality to exist – what search fn exists, need message composing, need a DB. Needs more clarity on these dependencies.
- Negotiable: Yes. Easy to interpret. Concise. DoD is from a client's point of view.

Architectural context: company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent: Yes? Dependency on previous functionality to exist – what search fn exists, need message composing, need a DB. Needs more clarity on these dependencies.
- Negotiable: Yes. Easy to interpret. Concise. DoD is from a client's point of view.
- Valuable: Bene■t to users explicitly stated. Able to message.

Architectural context: company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent: Yes? Dependency on previous functionality to exist – what search fn exists, need message composing, need a DB. Needs more clarity on these dependencies.
- Negotiable: Yes. Easy to interpret. Concise. DoD is from a client's point of view.
- Valuable: Bene■t to users explicitly stated. Able to message.
- Estimatable: ??? Tricky to estimate since existing features unclear.
- Lack of clarity around DB. Does the DB support partial searches? This may radically change the estimate for this story.

Architectural context: company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent: Yes? Dependency on previous functionality to exist – what search fn exists, need message composing, need a DB. Needs more clarity on these dependencies.
- Negotiable: Yes. Easy to interpret. Concise. DoD is from a client's point of view.
- Valuable: Bene■t to users explicitly stated. Able to message.
- Estimatable: ??? Tricky to estimate since existing features unclear.
- Lack of clarity around DB. Does the DB support partial searches? This may radically change the estimate for this story.
- Small: Feature is self-contained, certainly doesn't feel large.

Architectural context: company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

User story INVEST walkthrough

- Independent: Yes? Dependency on previous functionality to exist – what search fn exists, need message composing, need a DB. Needs more clarity on these dependencies.
- Negotiable: Yes. Easy to interpret. Concise. DoD is from a client's point of view.
- Valuable: Bene■t to users explicitly stated. Able to message.
- Estimatable: ??? Tricky to estimate since existing features unclear.
- Lack of clarity around DB. Does the DB support partial searches? This may radically change the estimate for this story.
- Small: Feature is self-contained, certainly doesn't feel large.
- Testable: Yes. End-to-end test formulated in the DoD.
- Some ambiguity – "contacts included" means. What exactly?

Architectural context: company email system with a database of contacts.
User story: As an employee, I want to search for contacts by name so I can message them.
Definitions of done: employee can enter a name or partial name into a text box, click search, will see the list of contacts matching the search with check box next to each one, and they can then press a message button once at least one [contact] is selected and then a compose email with contacts included appears.

C1

- C1 Due Friday @ 1800. J1 Posted (Due the following Friday).
- 81 teams at Developing/Pro■cient (50+ teams not started).
- AutoTest latency < 5 mins for the past two weeks.
- #c1 and #check have been updated.

■

Both provide better feedback than before.

■

Most common problems we’re seeing:

■

Using the *sync APIs from fs-extra.

■

Referencing ■les not committed to version control.

- 14h OO this week; expect delays later in the week.
- Q0: The rubric for one question was wrong (on mocking).

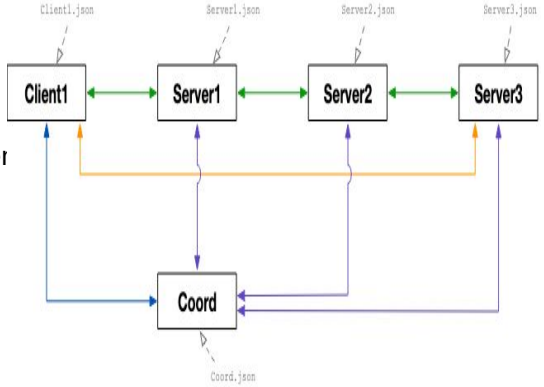
■

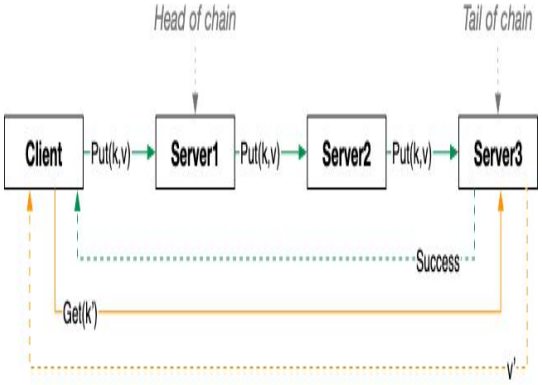
We will mark this correctly on the backend, but might not be able to update the scoring shown in PrairieLearn.

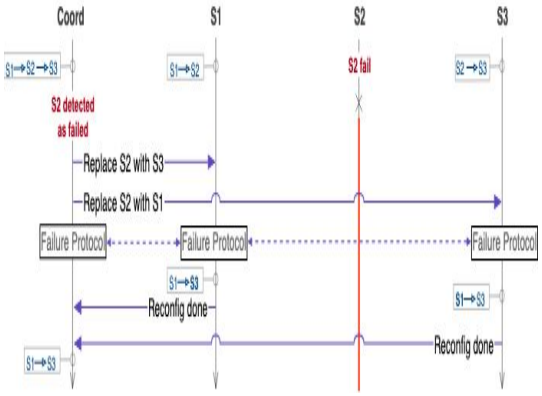
• Requirements must be comprehensible for the client.
• Speci■cations are written for the engineering team.
• They communicate the same thing, but to di■erent audiences.
• Unlike reqs, specs can vary in their degree of formalism.
• Key challenge: Translating informal (often vague) requirements into the SE domain.
• This is hard due to the gap between the abstraction of natural languages and the precision required by an implementation.
• Many tools: FSM/UML/diagrams, formal logic, pseudocode, …
From requirements to speci■cations

• Requirements must be comprehensible for the client.
• Speci■cations are written for the engineering team.
• They communicate the same thing, but to di■erent audier
• Unlike reqs, specs can vary in their degree of formalism.

From requirements to speci■cations



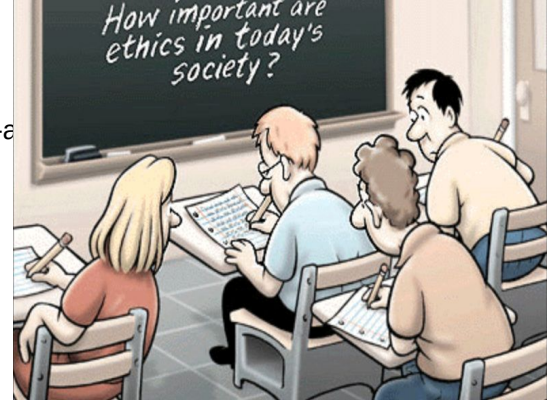




Process & speci■cations
●All software is built by teams.
●Every team follows its own process.
■
Waterfall, agile, & scrum most common today.
●Changes need requirements to be gathered and prioritized.
■
Abstraction gaps between text and code a constant challenge.
■
The more complete a speci■cation, the more overwhelming it is.
■
The less complete a speci■cation, the more likely it is to be incorrect.
●Many ethical concerns will be evaluated during the
requirements elicitation process.
●User stories often provide a useful middle ground.
■
Good for individual features and ■xes.
■
Can inhibit long-range planning unless care is taken.

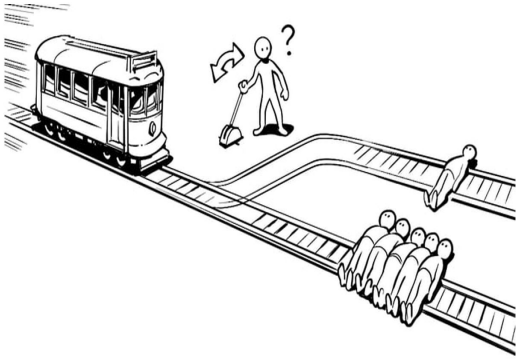
Ethics & Intellectual Property (IP)
[https://kenscourses.com/tc1019fall2016/syndicated/what-a

Reid Holmes & Nick Bradley



Examinable skills
Reason about moral actions in software engineering situations.
Be able to recognise ethical responsibilities and principles.
Be able to reason/rationalise how software design relates to ethical responsibilities and principles.
Understand how intellectual property rights a■ect consumers.
Be able to describe how intellectual property is used by producers to protect their intellectual investments.
Justify the tradeo■s between individual and business needs.
Be able to reason about how the most common source code licenses apply to speci■c clear examples.
Ethics & IP

Try a bunch!
[https://neal.fun/absurd-trolley-problems/]



[https://techcrunch.com/2019/11/13/the-math-of-sisyphus]

AUTONOMOUS CARS

Uber Got Off Easy

Aaron Gordon
Yesterday 12:25PM • Filed to: UBER

Elon Musk joins call for pause in creation of giant AI 'digital minds'

More than 1,000 artificial intelligence experts urge delay until world can be confident 'effects will be positive and risks manageable'


Elon Musk @elonmusk · 23h
Summary of argument against AI safety
but the side track is less direct and will thus slow down progress
10.4K
17.2K
178.9K
23.3M

Volkswagen emissions (Dieselgate)

```
if (isSteeringMoving()) {
disableEco();
} else {

enableEco();
}
```

OUTCOME

Europe misses its emissions targets (and VW pays > $33 billion in ■nes).

Dieselgate: millions of 'extremely' polluting cars still on Europe's roads, says report

The research group that first exposed the scandal say 'it's not over' and that governments must act

Names

- We've all created some kind of user object at some point.
- And users have names.
- And they have ■rst names and last names, right?
- Here are assumptions that are not true about names:

Names

- We've all created some kind of user object at some point.
- And users have names.
- And they have ■rst names and last names, right?
- Here are assumptions that are not true about names:

All people have exactly one full name.
All names consist of more than one letter.
All names are case sensitive.
All names are case insensitive.
Names are written in ASCII characters.
Names do not change.
Names are set at birth.
Names do not contain numbers.
Names are not in ALL CAPS.
Names have speci■c ordering.
People's names have a 'correct' ordering.
First names and last names are not the same.
[https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/]

Facebook among 30 organisations in UK political data inquiry

Information commissioner is investigating use of personal information in political campaigns

Facebook Privacy Scandal Unleashes Nationwide 'Litigation Swarm'

By **Christie Smythe**

April 4, 2018, 4:00 AM PDT *Updated on* April 4, 2018, 6:23 AM PDT

Australia latest to open probe into Facebook data scandal

Natasha Lomas @riptari / 6 hours ago

Comment

Dealing With Data

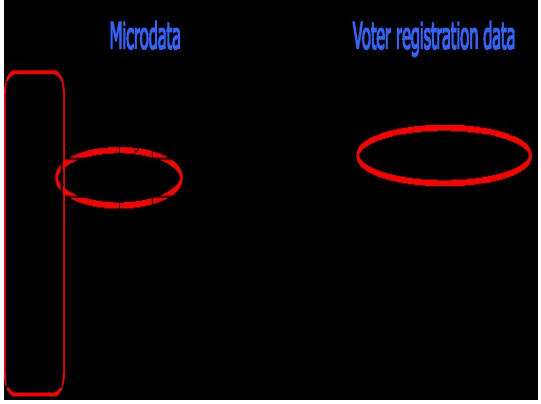
- Modern organizations collect vast amounts of data.
- This data is often analyzed or sold to third parties.
- Anonymization is frequently used to avoid reputational harm to the collecting organization.
- ■

As well as harm to those whose data is collected.

- Anonymization is simple to perform, but has many shortcomings.

Dealing With Data
Name
Zipcode
Age
Sex
Alice
47677
29
F
Bob
47983
65
M
Carol
47677
22
F
Dan
47532
23
M
Ellen
46789
43
F

Dealing With Data k-anonymity ensures information for a person cannot be distinguished from
at least k-1 other people in a dataset.
k = 1



Algorithmic Bias

Systematic and repeatable errors in a computer system that create "unfair" outcomes, such as "privileging" one category over another in ways di■erent from the intended function of the algorithm.

Examples:

- Assigning prices.
- Making shipping decisions.
- Approving loans & mortgages.
- Admitting people to schools.
- Making parole decisions.

Detecting Algorithmic Bias

Fairness Testing: Testing Software for Discrimination

Sainyam Galhotra Yuriy Brun Alexandra Meliou
University of Massachusetts, Amherst
Amherst, Massachusetts 01003-9264, USA
{sainyam, brun, ameli}@cs.umass.edu

• A wide variety of behaviours are societally negative.
• But not all of these can (or should) be legislated.
• Considering only legal restrictions is a pretty low bar.
• As software grows in importance, the software engineering community needs to consider the legal, moral, and societal impact of the work that we do.

Can’t the law take care of this?

• But unfortunately often don’t.
• Research shows we use common sense before considering codes of ethics, which can be problematic. [McNamara et. al.]
• So let’s try some of these ourselves!
• Go through some scenarios, considering the code of ethics.
• Who is at risk? What are the risks? What action is appropriate? What is the morally right choice?

Engineers Need to Consider Ethics

Intellectual Property

In order to protect against unauthorized use of your software, your company has built in an automatic kill switch that prevents it from running after a speci■c amount of time. An intensive care unit at the local hospital started using your software a year ago. The unit hasn't paid any of their bills and the kill switch is about to trigger. If the kill switch remains in place, the hospital will not be able to function as critical equipment will be disabled. You are capable of removing the kill switch for the hospital. What do you do?

A)

Edit the software to remove the kill switch.

B)

Do nothing, allowing the kill switch to activate.

ethics vignette 3

ACM Code of Ethics

https://www.acm.org/code-of-ethics

Code Quality

In going over a software speci■cation that your company has just been hired to create, your team discovers a large ■aw that could potentially a■ect the customer. Your company has spent the last year trying to negotiate this lucrative contract and your managers do not want to tell the customers about the issue because it might extend the negotiations even further. What do you do?

A)

Tell the customer about the issue.

B)

Do not tell the customer about the issue.

ACM Code of Ethics

https://www.acm.org/code-of-ethics

ethics vignette 4

Consider Cambridge Analytica
Cambridge Analytica collected users' data from Facebook without explicitly gaining consent. They claimed that informed consent was given because users clicked "I agree" when answering a survey. However, in reality, the extent of data collection was more than would have been assumed from a lay reading of the consent form. If you were a developer of the data collection module, what would you do?

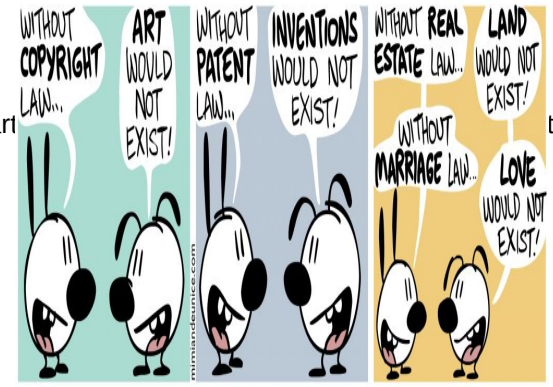
- Formally raise the issue with your team's leadership.
- Build the module, that's what you're being paid for.

https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal
ethics vignette 5
ACM Code of Ethics
https://www.acm.org/code-of-ethics

Work Inexperience
The startup you work for has obtained a contract to build an application. The owner of the startup is not a developer. You've been tasked with building the application together with your small team, but neither you nor your teammates have ever built this kind of application before, or even worked in this domain. The salary is very high, and the company stands to make a lot of money if the implementation is successful. What do you do?
A)
Warn your customers that there are risks associated with the contract.
B)
Do your best and hope it will work out through hard work.

ACM Code of Ethics
https://www.acm.org/code-of-ethics
ethics vignette 6

Intellectual Property
[https://www.ipdigit.eu/2011/09/what-can-you-■nd-in-a-cart ty/]
Reid Holmes & Nick Bradley



Intellectual Property (IP)
Intellectual property is a phrase used to de■ne "a concept in which tangible expressions of intellectual/creative pursuits — such as inventions, designs, creative works, etc. — are treated in legal and social spheres as property, with all its attendant implications (e.g., ownership, use, economic transactions, etc.). It is governed by the legal regimes of copyright, trademark, patent, and trade secret law.
— [https://www.canada.ca/]

●As an IP user, I need to honour creators’ rights and have limited rights to use existing work.
●As an IP creator, I have right to pro■t and must honour rights of other creators when I use their work.
●As an employee, have responsibility to comply with IP laws and to protect trade secrets.
Intellectual Property (IP)

• All modern tech businesses rely on a comprehensive set of approaches to protect their IP:
• Trademark: Distinguished feature that di■erentiates a product or organization in the marketplace.
• Copyright: Mechanism to protect creative work.
• Trade secrets: Protects information using con■dentiality agreements.
• Patents: State-conferred rights to an inventor for their invention.
• Licenses: Used with the other options to capitalize on IP.
• These are not exciting to think about but are fundamental to protect investments in research & development.

IP Protection

• Creating valuable products entails risk and investment.
• Ideas themselves are not inherently valuable.
• Protections exist to reward your risk by enabling you to pro■t from your product without having someone imitate it.

Individuals & Organizations Need Protection

• Intellectual Properties are creations of human intellect.
• Need to balance the rights of the creator with the public.
• Want to stimulate, not inhibit, the creation of new work.
• Creator:
• Time-bound exclusive rights.
• Public:
• Restricted rights for use of existing work in new creations.
IP: Balancing creative and public rights

• IP policies strive to not inhibit future innovation.
• This allows:
• Improvements to existing inventions.
• New works that include portions of existing work.
• Innovative interpretations of existing work.

Derivative works

…are

- What is the license of your software?
- Do you want to share your source?
- How do you want to distribute your system?
- Do you want to impose restrictions on others who extend your work?
- What is the license of any 3rd party software you had to change?
- How does this restrict your licensing options?
- What is the license of any 3rd party software you link to?
- Does their licenses allow for linking exceptions?

Licenses…
complicated!

• Some speci■c exemptions exist in copyright law for software:
• Backup copies are permissible.
• Reverse engineering for the purposes of detecting security vulnerabilities, conducting encryption research, or improving interoperability between programs.
• Standard conventions, restricted implementations, and obvious implementations cannot be protected with copyright.
Copyright exceptions for software

…are widely applied in the software space and clarify how code can be used and whether it can be modi■ed and distributed by others. Examples:
Licenses…
Restrictive/Copyleft: Generally require the same rights for derivative work.
E.g., GPL
Weak Copyleft: Enable portions of a system to be released under non-copyleft licenses.
Speci■c exceptions for linking to libraries.
E.g., LGPL
Permissive: Require only attribution, allowing non-derivative portions to remain proprietary.
E.g., MPL/MIT/BSD.

…are widely applied in the software space and clarify how code can be used and whether it can be modi■ed and distributed by others. Examples:

Licenses…

GPLv3 (General Public License) has been designed to enable software to be used, shared, and modi■ed. This is a copyleft license, which means that the license must be maintained when the code is modi■ed and/or distributed. Additionally, projects that use GPL code cannot use a less-restrictive license on their own product.

LGPLv3 (Lesser General Public License) is still a copyleft license, but more ■exible than GPL. While it requires that changes to code licensed under LGPL to be shared when used as a library by a product, the product itself need not be shared. LGPLv3 allows that software using LGPL libraries can be shared under a more restrictive license than the libraries it uses.

MPLv2 (Mozilla Public License) is another copyleft license that aims for increased ■exibility over LGPL in that it only mandates changes to ■les licensed MPL to be released (in contrast to the GPL and LGPL which require changes to the library to be released).

19 JANUARY 2021 NEWS

Amazon: NOT OK - why we had to change Elastic licensing

By Shay Banon

Share

On February 16, 2022, Elastic announced it had reached an agreement with Amazon on the trademark infringement lawsuit. Read about the agreement here.

Note: Since we initially published this blog, we completed our review and announced updates to the Elastic License.

We recently announced a license change: Blog, FAQ. We posted some additional guidance on the license change this morning. I wanted to share why we had to make this change.

This was an incredibly hard decision, especially with my background and history around Open Source. I take our responsibility very seriously. And to be clear, this change most likely has zero effect on you, our users. It has no effect on our customers that engage with us either in cloud or on premises. Its goal, hopefully, is pretty clear.

So why the change? AWS and Amazon Elasticsearch Service. They have been doing things that we think are just NOT OK since 2015 and it has only gotten worse. If we don't stand up to them now, as a successful company and leader in the market, who will?

Our license change is aimed at preventing companies from taking our Elasticsearch and Kibana products and providing them directly as a service without collaborating with us.

19 JANUARY 2021 NEWS

Amazon: NOT OK - why we had to change Elastic licensing

By Shay Banon

Share

On February 16, 2022, Elastic announced it had reached an agreement with Amazon on the trademark infringement lawsuit. Read about the agreement here.

Note: Since we initially published this blog, we completed our review and announced updates to the Elastic License.

We recently announced a license change: Blog, FAQ. We posted some additional guidance on the license change this morning. I wanted to share why we had to make this change.

This was an incredibly hard decision, especially with my background and history around Open Source. I take our responsibility very seriously. And to be clear, this change most likely has zero effect on you, our users. It has no effect on our customers that engage with us either in cloud or on premises. Its goal, hopefully, is pretty clear.

So why the change? AWS and Amazon Elasticsearch Service. They have been doing things that we think are just NOT OK since 2015 and it has only gotten worse. If we don't stand up to them now, as a successful company and leader in the market, who will?

Our license change is aimed at preventing companies from taking our Elasticsearch and Kibana products and providing them directly as a service without collaborating with us.

So why the change? AWS and Amazon Elasticsearch Service. They have been doing things that we think are just NOT OK since 2015 and it has only gotten worse. If we don't stand up to them now, as a successful company and leader in the market, who will?

Our license change is aimed at preventing companies from taking our Elasticsearch and Kibana products and providing them directly as a service without collaborating with us.

IP in Practice

- If you would like to hear about how IP is used in practice at a large organization, this interview from 2020 is interesting!
- Brian Dartnell is the VP Legal for Electronic Arts.
- Discusses the importance of IP in their organization.
- EA is fundamentally driven by IP.

[https://www.students.cs.ubc.ca/~cs-310/2020W1/Interview_BrianDartnell_PaulMain.mp4]

Ethics & IP

- We've talked about Ethics during testing and process, but it will also resurface during design and implementation.
- Ethics and IP are cross-cutting concerns that must be considered throughout the lifecycle of a software system.
- These important concerns require careful thought, but individuals on technical teams have great power to highlight these issues.