Process

Martin Kellogg

Process

Today's agenda:

- **Finish up VCS slides**
- Reading Quiz
- Development methodologies
- Planning, estimation, and risk
- Measuring progress

Version Control

Today’s agenda:

- Reading Quiz
- How does a version control system work?
- How to use your VCS
- **GitHub workflows**

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main
- in your fork, create a **feature branch**

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main
- in your fork, create a **feature branch**
- write code + tests

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main
- in your fork, create a **feature branch**
- write code + tests
- commit **early and often**, push to your fork

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main
- in your fork, create a **feature branch**
- write code + tests
- commit **early and often**, push to your fork
- **prepare** for code review: follow code review author's best practices

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main
- in your fork, create a **feature branch**
- write code + tests
- commit **early and often**, push to your fork
- **prepare** for code review: follow code review author's best practices
  - we'll discuss how to do a code review in a few weeks

How to make a PR on GitHub

- start by creating a ***fork*** of the project
  - a new repository controlled by you, connected to the main
- in your fork, create a **feature branch**
- write code + tests
- commit **early and often**, push to your fork
- **prepare** for code review: follow code review author's best practices
  - we'll discuss how to do a code review in a few weeks
- open PR against "**main**" repository from your fork's feature branch

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch
- write code (if there are already tests, don't bother to run them)

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch
- write code (if there are already tests, don't bother to run them)
- commit **all** of your code at once, when you're done

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch
- write code (if there are already tests, don't bother to run them)
- commit **all** of your code at once, when you're done
- **don't bother** to check if you've followed best practices

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch
- write code (if there are already tests, don't bother to run them)
- commit **all** of your code at once, when you're done
- **don't bother** to check if you've followed best practices
- **email** your changes to the maintainer of the original project

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch
- write code (if there are already tests, don't bother to run them)
- commit **all** of your code at once, when you're done
- **don't bother** to check if you've followed best practices
- **email** your changes to the maintainer of the original project
  - bonus points: email the full working copy, not just the diffs

How NOT to make a PR on GitHub

- start by creating a ***hard fork*** of the project
  - a new repository controlled by you, unconnected to the main
- do all of your work on the repository's **main** branch
- write code (if there are already tests, don't bother to run them)
- commit **all** of your code at once, whe
- **don't bother** to check if you've follo
- **email** your changes to the maintaine
  - bonus points: email the full work

I've seen people make all of these mistakes (and more)!

Process

Today's agenda:

- Finish up VCS slides
- **Reading Quiz**
- Development methodologies
- Planning, estimation, and risk
- Measuring progress

Reading Quiz: process

Q1: **TRUE** or **FALSE**: the author of “Waterfall doesn't mean what you think it means” argues that documentation is the key to a successful software project

Q2: Which of the following would the Agile Manifesto’s authors argue is the best measure of progress in a software project?

A. how much of the software is working
B. the engineers’ estimate of how much work is left
C. the percentage of statements that are covered by a test

Reading Quiz: process

Q1: **TRUE** or **FALSE**: the author of “Waterfall doesn't mean what you think it means” argues that documentation is the key to a successful software project

Q2: Which of the following would the Agile Manifesto’s authors argue is the best measure of progress in a software project?

A. how much of the software is working
B. the engineers’ estimate of how much work is left
C. the percentage of statements that are covered by a test

Reading Quiz: process

Q1: **TRUE** or **FALSE**: the author of “Waterfall doesn't mean what you think it means” argues that documentation is the key to a successful software project

Q2: Which of the following would the Agile Manifesto’s authors argue is the best measure of progress in a software project?

A. how much of the software is working
B. the engineers’ estimate of how much work is left
C. the percentage of statements that are covered by a test

Process

Today's agenda:

- Finish up VCS slides
- Reading Quiz
- **Development methodologies**
- Planning, estimation, and risk
- Measuring progress

Development methodologies

- Traditionally, a large component of undergrad Software Engineering classes

Development methodologies

- Traditionally, a large component of undergrad Software Engineering classes
- I’m not going to make you memorize the stages of the Waterfall method, or the tenets of Agile, or the like

Development methodologies

- Traditionally, a large component of undergrad Software Engineering classes
- I’m not going to make you memorize the stages of the Waterfall method, or the tenets of Agile, or the like
  - Why? No one actually follows these procedures to the letter

Development methodologies

- Traditionally, a large component of undergrad Software Engineering classes
- I'm not going to make you memorize the stages of the Waterfall method, or the tenets of Agile, or the like
  - Why? No one actually follows these procedures to the letter
- Instead, my goal in this lecture is to give you an overview of the traditional ways of organizing a software development effort and give you the vocabulary to talk about it

What is a process? A methodology?

Definition: a ***software process*** is the set of activities and associated results that produce a software product

What is a process? A methodology?

Definition: a ***software process*** is the set of activities and associated results that produce a software product

Definition: a ***software development methodology*** is a template for a process: a specific set of activities (usually accompanied by an animating philosophy) for a team to follow to achieve a desired result

What is a process? A methodology?

Definition: a ***software process*** is the set of activities and associated results that produce a software product

Definition: a ***software development methodology*** is a template for a process: a specific set of activities (usually accompanied by an animating philosophy) for a team to follow to achieve a desired result

e.g., the Agile manifesto

What is a process? A methodology?

Definition: a ***software process*** is the set of activities and associated results that produce a software product

Definition: a ***software development methodology*** is a template for a process: a specific set of activities (usually accompanied by an animating philosophy) for a team to follow to achieve a desired result

not a guarantee - just a goal

A list of methodologies

- Waterfall
- Spiral
- Agile
- Scrum
- Extreme Programming (XP)
- “wagile”

A list of methodologies

- **Waterfall**
- **Spiral**
- **Agile**
- **Scrum**
- Extreme Programming (XP)
- “wagile”

We’ll discuss these four - you can look up the others on your own if you’re curious

Why have a methodology at all?

Why have a methodology at all?

- Standardization among developers
- Shared language
- Estimation: your boss probably wants to know when you'll be able to ship!
- You implicitly have a process, whether you know it or not (and it might not be very good if you're not paying attention)

Why have a methodology at all?

- Standardization among developers
- Shared language
- Estimation: your boss probably wants to know when you'll be able to ship!
- You implicitly have a process, whether you know it or not (and it might not be very good if you're not paying attention)

▲

sometimes this is called an ***ad hoc*** methodology

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.

Examples of issues with ad hoc process

- **Requirements**: Mid-project inf
  suggested by customer or man

Project scope expands (a lot!)

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirem
  Test-debug-reimplement cycle

Release with known defects

Defect cost vs. detection time

- An IBM report gives an average defect repair cost of (2008$):
  - $25 during coding
  - $100 at build time
  - $450 during testing/QA
  - $16,000 post-release

[L. Williamson. IBM Rational software analyzer: Beyond source code. 2008.]

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.
- **Defect Tracking**: Bug reports collected informally.

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits d[illegible]s.
- **Defect Tracking**: Bug reports collecte[illegible]

Bugs forgotten

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.
- **Defect Tracking**: Bug reports collected informally.
- **System Integration**: Integration of independently developed components at the very end of the project.

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.
- **Defect Tracking**: Bug reports collected informally.
- **System Integration**: Integration [illegible] components at the very end of t[illegible]

Interfaces out of sync

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.
- **Defect Tracking**: Bug reports collected informally.
- **System Integration**: Integration of independently developed components at the very end of the project.
- **Scheduling**: When project is behind, developers are asked weekly for new estimates.

Examples of issues with ad hoc process

- **Requirements**: Mid-project informal agreement to changes suggested by customer or manager.
- **QA**: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.
- **Defect Tracking**: Bug reports collected informally.
- **System Integration**: Integration of independently developed components at the very end of the project.
- **Scheduling**: When project is behind, developers are asked weekly for new estimates.

Project falls further behind

A process hypothesis

- A process can **increase flexibility and efficiency** for software development.
  - If this is true, an up-front investment (of resources, e.g., “time”) in process can yield greater returns later on - by avoiding the problems on the previous slide!

A list of methodologies

- **Waterfall**
- **Spiral**
- Agile
- Scrum
- Extreme Programming (XP)
- “wagile”

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out “**in order**”:

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a document

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a document
- Analysis: create models, schema, and business rules

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a document
- Analysis: create models, schema, and business rules
- Design: create the software architecture

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a document
- Analysis: create models, schema, and business rules
- Design: create the software architecture
- Coding: the development, proving, and integration of software

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a document
- Analysis: create models, schema, and business rules
- Design: create the software architecture
- Coding: the development, proving, and integration of software
- Testing: the systematic discovery and debugging of defects

The Waterfall methodology

In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a document
- Analysis: create models, schema, and business rules
- Design: create the software architecture
- Coding: the development, proving, and integration of software
- Testing: the systematic discovery and debugging of defects
- Operations: the installation, migration, support, and maintenance of complete systems

The Waterfall methodology

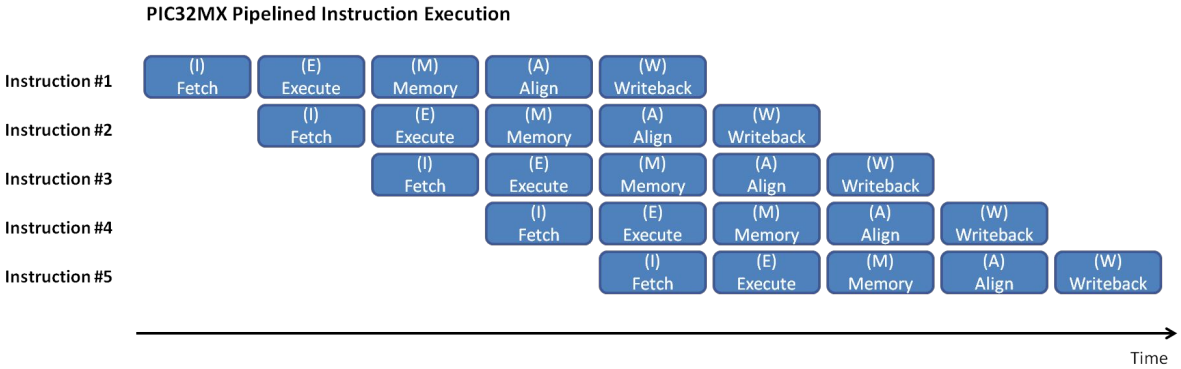
In the ***waterfall*** software development model, the following phases are carried out "**in order**":

- Requirements gathering: produce a docu
- Analysis: create models, schema, and bus
- Design: create the software architecture
- Coding: the development, proving, and integration of software
- Testing: the systematic discovery and debugging of defects
- Operations: the installation, migration, support, and maintenance of complete systems

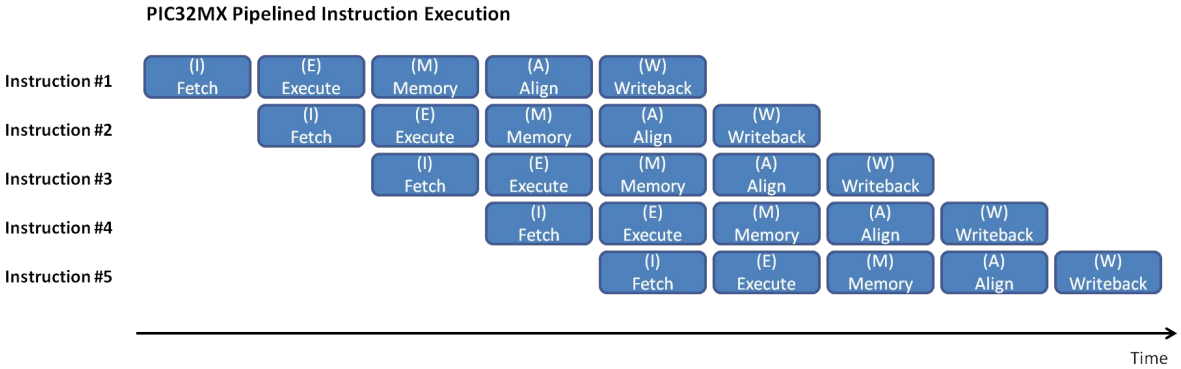
Is this realistic?
Why or why not?

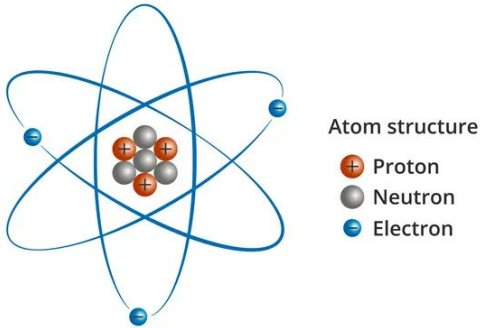
Other lies you’ve probably been told

Other lies you've probably been told

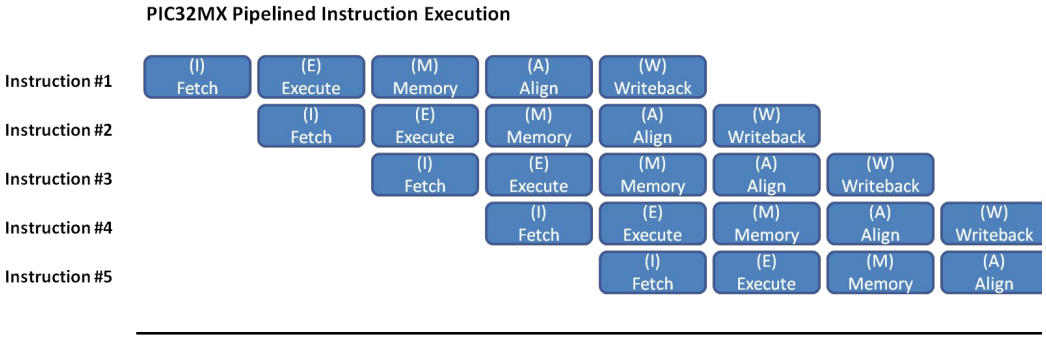


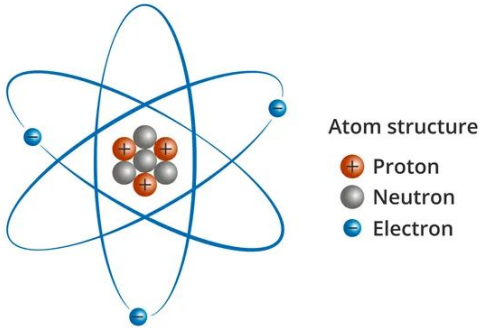
Other lies you've probably been told

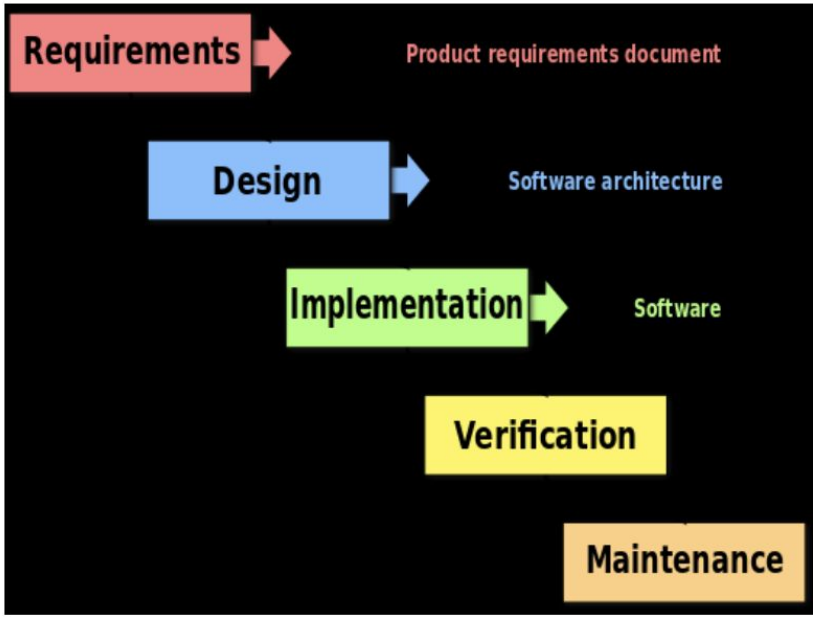




Other lies you've probably been told







The Waterfall methodology: an idealized model

- Do **NOT** attempt to actually follow a strict Waterfall methodology in real life
  - you **will** have a bad time

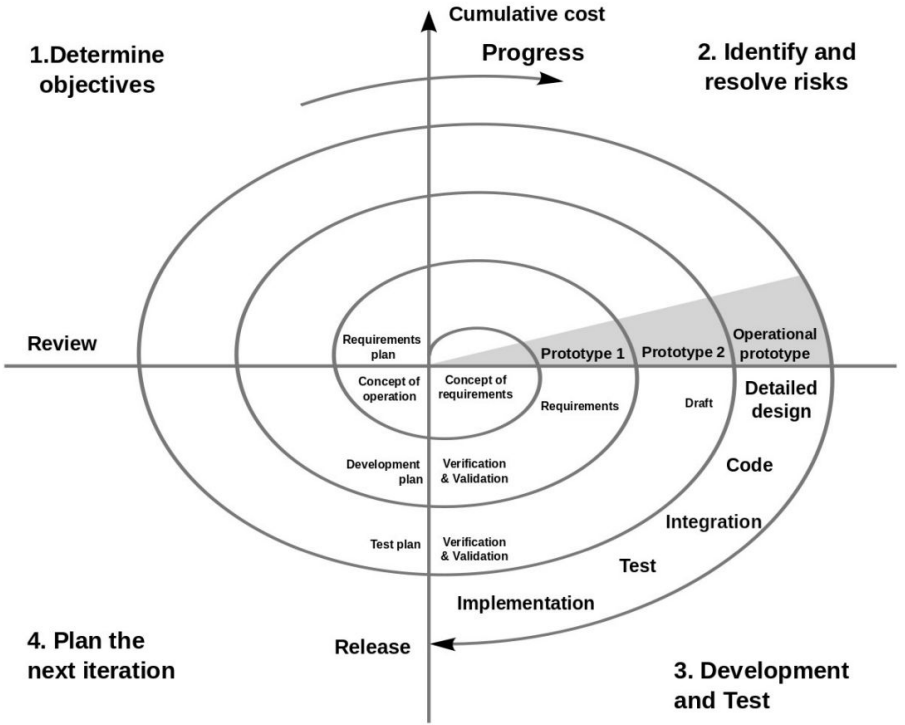
The Waterfall methodology: an idealized model

- Do **NOT** attempt to actually follow a strict Waterfall methodology in real life
  - you **will** have a bad time
- But, it provides a **useful foundation** for thinking about methodologies:
  - the Waterfall stages do represent real activities you'll do during the development lifecycle
  - you probably won't do them all in the proscribed order

The Waterfall methodology: an idealized model

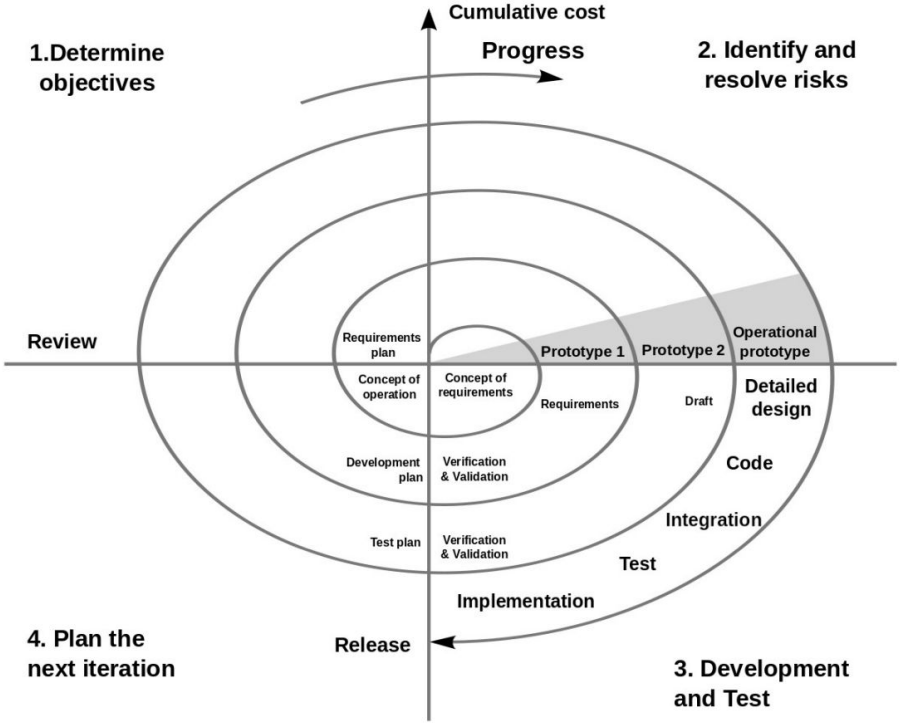
- Do **NOT** attempt to actually follow a strict Waterfall methodology in real life
  - you **will** have a bad time
- But, it provides a **useful foundation** for thinking about methodologies:
  - the Waterfall stages do represent real activities you'll do during the development lifecycle
  - you probably won't do them all in the proscribed order
- And, as we saw in today's reading, this "model" isn't even what the papers "introducing it" were advocating for!

A slightly more realistic model: spiral



A slightly more realistic model: spiral

- Key idea: construct a series of increasingly-complete **prototypes**
- Effectively **iterated waterfall**



A list of methodologies

- Waterfall
- Spiral
- **Agile**
- **Scrum**
- Extreme Programming (XP)
- “wagile”

Agile & Scrum

- Agile is more a **philosophy** than a methodology in the traditional sense

Agile & Scrum

- Agile is more a **philosophy** than a methodology in the traditional sense
- Scrum is an **instantiation** of that philosophy as a methodology

Agile Principles

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile Principles

Focus on people

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile Principles

- Individuals and interactions over processes and tools
- Working software over comprehensive do
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Always have a prototype

Agile Principles

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract n
- Responding to change over following a plan

Keep the client involved

Agile Principles

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Change requirements as you learn about the problem

The Scrum methodology

- Scrum is one common Agile methodology
- Focused around a “scrum master” who is responsible for process
- Work is divided into ***sprints*** where each team member is responsible for dealing with certain tasks
  - starts with a “sprint planning meeting”: tasks are assigned
  - each day includes a “standup” ceremony
  - at the end of the sprint, a “sprint retrospective meeting” looks back on how the sprint went
  - typically sprints are 1-2 weeks

Common features of Agile methodologies

- Sprint terminology is common, even when not directly using scrum
- “Daily standups” are a routine for many engineers
- Planning often happens in the form of ***user stories***

Common features of Agile methodologies

- Sprint terminology is common, even when not directly using scrum
- “Daily standups” are a routine for many engineers
- Planning often happens in the form of ***user stories***
  - As a ___, I want to ____

Common features of Agile methodologies

- Sprint terminology is common, even when not directly using scrum
- “Daily standups” are a routine for many engineers
- Planning often happens in the form of ***user stories***
  - As a ___, I want to ____
  - E.g., “as a new Covey.Town user, I want to create an account”

Common features of Agile methodologies

- Sprint terminology is common, even when not directly using scrum
- “Daily standups” are a routine for many engineers
- Planning often happens in the form of ***user stories***
  - As a ___, I want to ____
  - E.g., “as a new Covey.Town user, I want to create an account”

You’ll need to describe your group project using user stories in your project proposal

Process

Today's agenda:

- Finish up VCS slides
- Reading Quiz
- Development methodologies
- **Planning, estimation, and risk**
- Measuring progress

Planning

- A project should **plan** time, cost and resources adequately to **estimate** the work needed and to effectively **manage risk** during project execution.

Planning

- A project should **plan** time, cost and resources adequately to **estimate** the work needed and to effectively **manage risk** during project execution.

Planning = estimate +/- risk

Why is planning a software project difficult?

- Software tends to be **innovative**

Why is planning a software project difficult?

- Software tends to be **innovative**
  - Cost of copying existing code ≈ 0, so any project you're actually working on probably is different than what came before

Why is planning a software project difficult?

- Software tends to be **innovative**
  - Cost of copying existing code ≈ 0, so any project you're actually working on probably is different than what came before
  - "It's not research if you know it's going to work"

Why is planning a software project difficult?

- Software tends to be **innovative**
  - Cost of copying existing code ≈ 0, so any project you're actually working on probably is different than what came before
  - "It's not research if you know it's going to work"
  - Compare to other kinds of engineering: one highway/bridge/skyscraper/etc isn't that different than the next

Planning: milestones and deliverables

Definition: A ***milestone*** is a clean end point of a (sub)task

Planning: milestones and deliverables

Definition: A ***milestone*** is a clean end point of a (sub)task

- Reports, prototypes, completed subprojects, etc.
- “80% done” is **NOT** a suitable milestone (too vague)

Planning: milestones and deliverables

Definition: A ***milestone*** is a clean end point of a (sub)task

- Reports, prototypes, completed subprojects, etc.
- “80% done” is **NOT** a suitable milestone (too vague)

Definition: A ***deliverable*** is a milestone that’s customer-facing

- sometimes used interchangeably with milestone

Why milestones and deliverables?

- It’s easy to tell when a milestone or deliverable is done
- **Progress** towards milestones and deliverables is hard to measure

Why milestones and deliverables?

- It’s easy to tell when a milestone or deliverable is done
- **Progress** towards milestones and deliverables is hard to measure

“All I need to do is fix this one bug and then it’ll work, promise.”

Estimation

Two parts:

- How long do you think it will take to reach the next milestone?
- Splitting larger tasks into smaller ones

Estimation

Two parts:

- How long do you think it will take to reach the next milestone?
- Splitting larger tasks into smaller ones

Naturally **very fuzzy**: we can’t see the future

Estimation techniques: t-shirt sizing

small = I can do this task in an afternoon

Estimation techniques: t-shirt sizing

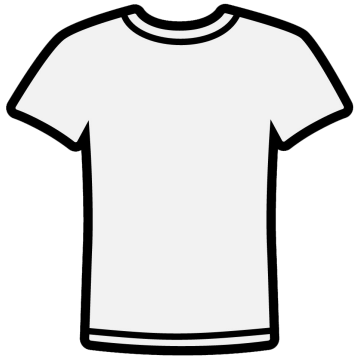
small = I can do this task in an afternoon

medium = I can do this task in a day or two

Estimation techniques: t-shirt sizing

small = I can do this task in an afternoon

medium = I can do this task in a day or two



large = too big to estimate how long it will take

- large tasks should usually come with a small task that is breaking the large task up into medium and small tasks

Estimation techniques: story points

- Assign stories 1, 2, 4, or 8 points (these numbers can vary, but the relationship should be **exponential**)
- Like large t-shirt estimates, high-point-value stories should usually have a smaller task to break them apart

Estimation techniques: story points

- Assign stories 1, 2, 4, or 8 points (these numbers can vary, but the relationship should be **exponential**)
- Like large t-shirt estimates, high-point-value stories should usually have a smaller task to break them apart
- T-shirt estimates and story points are two different ways to quantify the **relative** size of tasks
    - Also lots of other ways!

Estimation techniques: cocomo

Definition: a ***constructive cost model*** (***cocomo***) is a predictive model of time costs based on project history

Estimation techniques: cocomo

Definition: a ***constructive cost model*** (***cocomo***) is a predictive model of time costs based on project history

- requires experience with similar projects
- rewards documentation of experience
- basically, it's an empirically-derived set of “effort multipliers”. You multiply the time cost by some numbers from a chart:

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Risk and uncertainty

- **Risk management** is the identification, assessment, and prioritization of risks, followed by efforts to minimize, monitor and control unfortunate event outcomes and probabilities.

Risk and uncertainty

- **Risk management** is the identification, assessment, and prioritization of risks, followed by efforts to minimize, monitor and control unfortunate event outcomes and probabilities.
- Any effective plan for software development must take into account common risks, e.g.,:

Risk and uncertainty

- **Risk management** is the identification, assessment, and prioritization of risks, followed by efforts to minimize, monitor and control unfortunate event outcomes and probabilities.
- Any effective plan for software development must take into account common risks, e.g.,:
  - Staff illness or turnover, product is too slow, competitor introduces a similar product, etc.

Risk and uncertainty

- **Risk management** is the identification, assessment, and prioritization of risks, followed by efforts to minimize, monitor and control unfortunate event outcomes and probabilities.
- Any effective plan for software development must take into account common risks, e.g.,:
  - Staff illness or turnover, product is too slow, competitor introduces a similar product, etc.

bus factor: number of people that need to be simultaneously unavailable until there is a part of the system that no one understands

Strategies for risk management

Strategies for risk management

- Address risk early
- Selectively innovate to increase value while minimizing risk (i.e., focus risk where needed)
- Use iteration and feedback (e.g., prototypes)
- Estimate likelihood and consequences
  - Requires experienced project leads
  - Rough estimates (e.g., <10%, <25%) are OK
- Have contingency plans

Strategies for risk management

- Address risk early
- Selectively innovate to increase value while minimizing risk (i.e., focus risk where needed)
- Use iteration and feedback (e.g., prototypes)
- Estimate likelihood and consequences
  - Requires experienced project lea
  - Rough estimates (e.g., <10%, <25
- Have contingency plans

Your ability to do this will come from **practice** (in your first job, senior engineers + tech leads + PMs should do these things)

Process

Today's agenda:

- Finish up VCS slides
- Reading Quiz
- Development methodologies
- Planning, estimation, and risk
- **Measuring progress**

Measuring progress

Easy strategy: only track milestones and deliverables

Measuring progress

Easy strategy: only track milestones and deliverables

- Downside: no way to know how close you are to the next one

Measuring progress

Easy strategy: only track milestones and deliverables

- Downside: no way to know how close you are to the next one

Can we do better? Unfortunately, not really.

Measuring progress: best practices

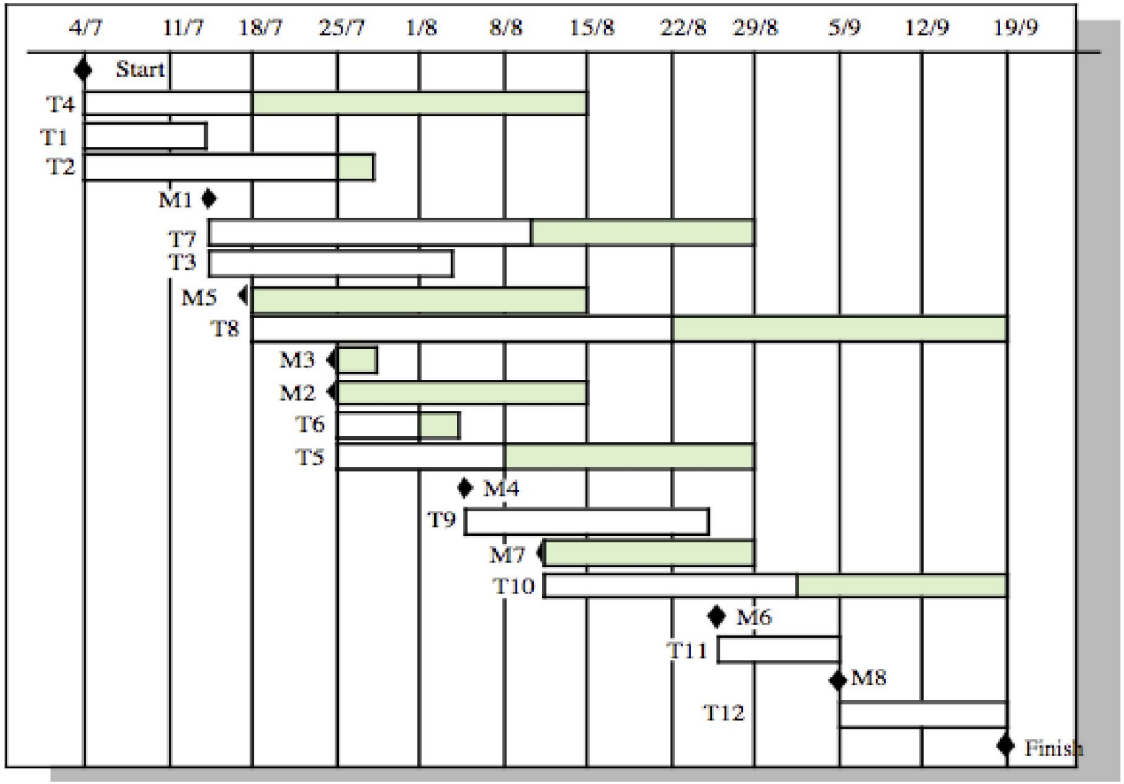
- have many milestones/deliverables
  - think back to Agile: this is a reason to always have a prototype

Measuring progress: best practices

- have many milestones/deliverables
  - think back to Agile: this is a reason to always have a prototype
- avoid relying too heavily on developers' estimates
  - we are bad at estimating
  - "last mile" problem: what seems to be last 10% of the work often takes 40% or more of the development time

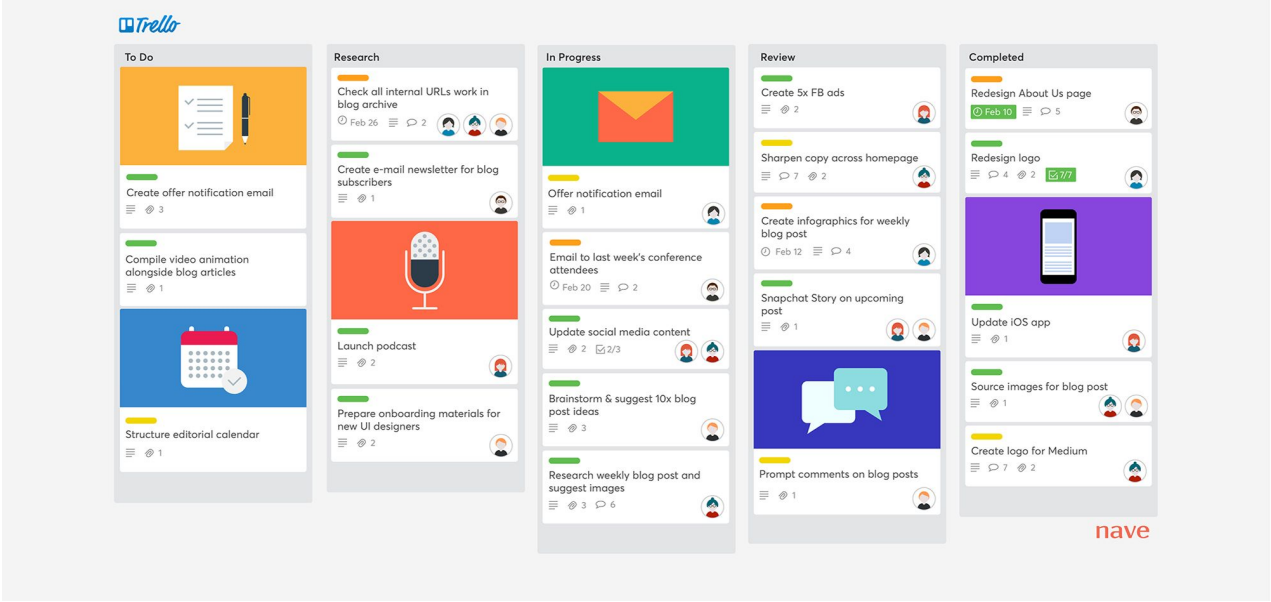
Measuring progress: tools

- Gantt chart



Measuring progress: tools

- Gantt chart
- KanBan board



Measuring progress: tools

- Gantt chart
- KanBan board
- Many others: use what works for you

Takeaways

- Process can save time, but don’t overdo it
- Lots of methodologies: choose what makes sense for you
- Agile philosophy is generally a good one to follow
  - But don’t focus on it at the expense of actually doing your job
- Estimation is hard and you will get it wrong
  - Use rough estimation strategies to avoid over-promising
- Include lots of buffer + risk in your estimates
- Don’t trust developer estimates in general