Software Requirements

CSE 403 Software Engineering

Winter 2025

Today’s Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. What are techniques used to specify them?

Today's Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. ~~What are techniques used to specify them?~~ Part-2 on Friday
4. Version control and git (Part-1)

(Optional) Pitch signup sheet available – see Ed for pointer

Recapping where requirements fit in

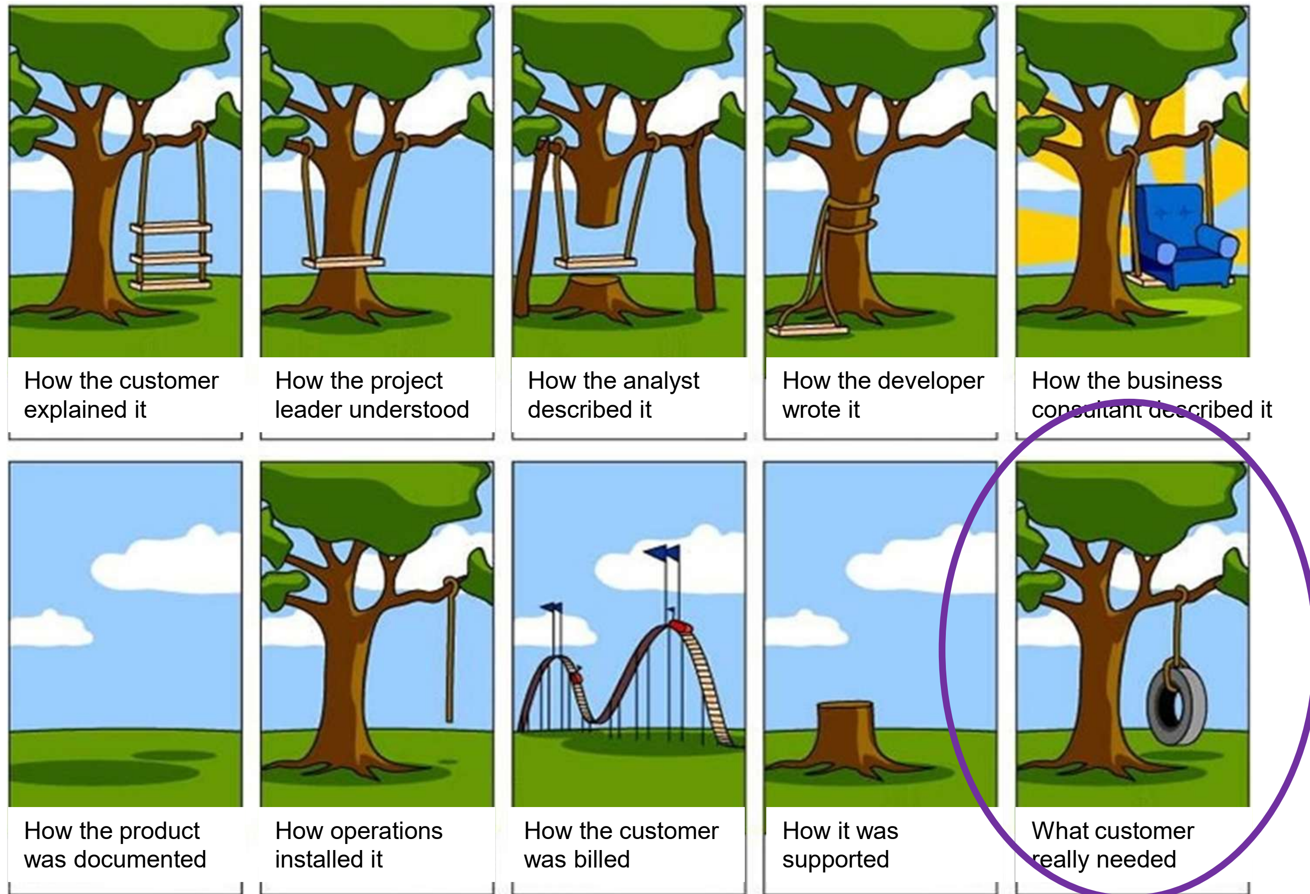
Virtually all SDLC models have the following stages →

Requirements are at the top of the list as we start the journey of product development

<u>Common stages</u>

Requirements
Design
Implementation
Testing
Release
Maintenance

Sharing a visual of their importance



What exactly are software requirements?

Requirements specify what to build

- describe **what, not how**
- describe customer needs, not how they'll be implemented
- reflect product design (product goals), not software design

> Product requirements describe the product's functionality in terms understandable by devs and customers, with as close to zero ambiguity as possible
>
> -Isaac Reynolds, Google GPM

Let’s work through an example

Are these good requirements for an audio player?

- Available on web and mobile
- Provide volume control
- Provide ability to flag favorites using a pulldown menu
- Enable variable playback speed
- Propose songs using ChatGPT recommendations
- Propose songs based on customer selected genres
- Written in javascript for extensibility and reliability

How about our swing example

What are good **and sufficient** requirements for the swing?

- Attaches to a single branch of a tree
- Seats one person 3-5ft tall
- Swings when pushed
- Appeals to environmental advocates
-
-

Requirements are hard but important

They help us:

- **Understand** precisely what is required of the software
- **Communicate** this understanding precisely to all involved parties
- **Monitor and control** production to ensure that system meets specification

In practice, they're used by many during SDLC

- **Customers**: what should be delivered (contractual base)
- **Project managers**: scheduling and monitoring (progress indicator)
- **Designers**: basis for a spec to design the system
- **Developers**: a range of acceptable implementations
- **QA / Testers (DevTest)**: a basis for testing, verification, and validation

Today's Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. ~~What are techniques used to specify them?~~ Part-2 on Friday
4. Version control and git (Part-1)

Let's start with some data

From a Standish report on software project success

Customer involvement is 3rd highest factor of project success!

https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

CHAOS FACTORS OF SUCCESS

FACTORS OF SUCCESS	POINTS	INVESTMENT
Executive Sponsorship	15	15%
Emotional Maturity	15	15%
User Involvement	15	15%
Optimization	15	15%
Skilled Resources	10	10%
Standard Architecture	8	8%
Agile Process	7	7%
Modest Execution	6	6%
Project Management Expertise	5	5%
Clear Business Objectives	4	4%

The 2015 Factors of Success. This chart reflects our opinion of the importance of each attribute and our recommendation of the amount of effort and investment that should be considered to improve project success.

Successful businesses always start with the user's goal

Marshall Field's

The customer is always right

Google

Focus on the user and all else will follow

amazon

Customer obsession rather than competitor focus

Understand and serve the customer better than anyone else

So, how do we engage with customers

Ideas?

- Be a user yourself (but be careful not to bias)
- Talk with users informally (hallway chats, mixers)
- Talk with users formally (interviews, surveys, diary studies, field studies)
- Build low-fidelity prototypes (mocks, UX prototypes, eng prototypes)
- Launch and get feedback early (“launch and iterate”)

Keep your customer (user) at the center of the discussion
Listen, observe and **ask clarifying questions**

Do’s and don’ts in requirements gathering

Do:

- Talk to the **customers** -- to learn how they work
- Ask questions throughout the process -- "**dig**" for requirements
- Think about **why** users do something in your service, not just what
- Allow (and expect) requirements to change later

Do’s and don’ts in requirements gathering

Do:

- Talk to the **customers** -- to learn how they work
- Ask questions throughout the process -- "**dig**" for requirements
- Think about **why** users do something in your service, not just what
- Allow (and expect) requirements to change later

Don't:

- Be too specific or detailed
- Describe complex business logic or rules of the system
- Describe the exact user interface used to implement a feature
- Try to think of everything ahead of time* (caveats apply)
- Add unnecessary features not wanted by the customers

The whole process is more formally known as <u>requirements engineering</u>

The science of eliciting, analyzing, documenting, and maintaining requirements

As you collect your class project requirements (02 Requirements), consider three categories:

- Functional requirements
  - e.g., input-output behavior
- Non-functional requirements
  - e.g., security, privacy, scalability
- Additional constraints
  - e.g., programming language, frameworks, testing infrastructure

Requirements can be extensive – leveraging existing frameworks (categories, templates) can help

- User features
- Performance and System Health
- Reliability
- Scalability
- Warranties or maintenance goals
- Possible or likely future goals
- Target platforms or environments
- Regulatory and legal
- External documentation, user "help"
- Marketing claims
- Logging and success metrics
- Manual testing guides
- Accessibility
- Internationalization, localization, language support
- Troubleshooting guides
- Leak prevention
- Threat models and security guarantees
- User privacy
- Simplicity and usability

It’s essential to prioritize

If everything is a “Priority 0” (P0), then nothing is!

- P0 means we’d be embarrassed not to have this
- P1 is what makes the feature better than the competition
- P2 is nice to have

It’s essential to prioritize

If everything is a “Priority 0” (P0), then nothing is!

- P0 means we’d be embarrassed not to have this
- P1 is what makes the feature better than the competition
- P2 is nice to have

Consider the example of a simple “camera” app.

- Takes photos.
- Takes videos.
- Crashes <0.01% of sessions.
- Opens in <1000ms 90% of the time.
- Takes slow motion videos.
- Takes time lapse videos.
- Does 4K30 resolution.
- Supports manual photography controls.
- Supports RAW capture mode.

It’s essential to prioritize

If everything is a “Priority 0” (P0), then nothing is!

- P0 means we’d be embarrassed not to have this
- P1 is what makes the feature better than the competition
- P2 is nice to have

Consider the example of a simple “camera” app.

P0	*Takes photos.*
P0	*Takes videos.*
P0	*Crashes <0.01% of sessions.*
P0	*Opens in <1000ms at the P90.*
P1	*Takes slow motion videos.*
P1	*Takes time lapse videos.*
P1	*Does 4K30.*
P2	*Supports manual photography controls.*
P2	*Supports RAW capture mode.*

Meet Alistair Cockburn – Requirements SME

Alistair Cockburn (/ˈælɪstər ˈkoʊbərn/ *AL-ist-ər KOH-bərn*) is an American computer scientist, known as one of the initiators of the agile movement in software development. He cosigned (with 17 others)[1] the Manifesto for Agile Software Development.[2]

Alistair Cockburn

Alistair Cockburn in 2007

Nationality	American
Occupation	Computer programmer

Life and career [edit]

Cockburn started studying the methods of object oriented (OO) software development for IBM. From 1994, he formed "Humans and Technology" in Salt Lake City. He obtained his degree in computer science at the Case Western Reserve University. In 2003, he received his PhD degree from the University of Oslo.

Cockburn helped write the Manifesto for Agile Software Development in 2001, the agile PM Declaration of Interdependence in 2005, and co-founded the International Consortium for Agile in 2009 (with Ahmed Sidky and Ash Rofail). He is a principal expositor of the use case for documenting business processes and behavioral requirements for software, and inventor of the Cockburn Scale for categorizing software projects.

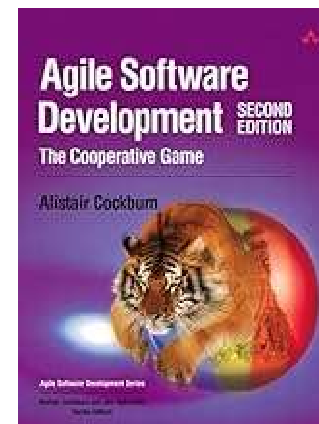
The methodologies in the Crystal family (e.g., Crystal Clear), described by Alistair Cockburn, are considered examples of lightweight methodology. The Crystal family is colour-coded to signify the "weight" of methodology needed. Thus, a large project which has consequences that involve risk to human life would use the Crystal Sapphire or Crystal Diamond methods. A small project might use Crystal Clear, Crystal Yellow or Crystal Orange.

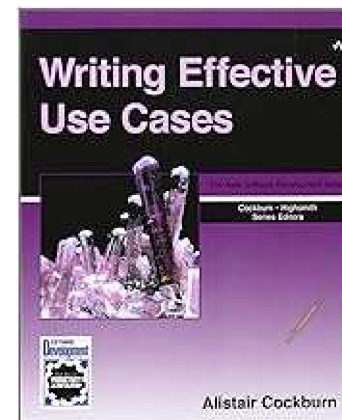
Cockburn presented his Hexagonal Architecture (2005) as a solution to problems with traditional layering, coupling and entanglement.

In 2015, Alistair launched the Heart of Agile movement which is presented as a response to the overly complex state of the Agile industry.

Selected publications [edit]

- *Surviving Object-Oriented Projects*, Alistair Cockburn, 1st edition, December, 1997, Addison-Wesley Professional, ISBN 0-201-49834-0.
- *Writing Effective Use Cases*, Alistair Cockburn, 1st edition, January, 2000, Addison-Wesley Professional, ISBN 0-201-70225-8.
- *Agile Software Development*, Alistair Cockburn, 1st edition, December 2001, Addison-Wesley Professional, ISBN 0-201-69969-9.





https://en.wikipedia.org/wiki/Alistair_Cockburn

Cockburn requirements template

1. Purpose and scope
2. Terms (glossary)
3. **Use cases (the central artifact of requirements)**
4. Technology used
5. Other
   - Development process: participants, values (fast-good-cheap), visibility, competition, dependencies
   - Business rules (constraints)
   - Performance demands
   - Security, documentation
   - Usability
   - Portability
   - Unresolved (deferred)
6. Human factors (legal, political, organizational, training)

Many companies will have a template for you to use

Uniformity is good for you and the customer

See what we're asking of you in the 02 Requirements assignment

Use a **use case** to capture a requirement

As a [user], I want to [action] so that [result]

Camera app

1. As a parent, I want to take sharp photos of my kids in medium-low light so I can have memories of early holiday mornings.
2. As a creative, I want to adjust the look-and-feel of my photos so that they match how I remember the moment.
3. As a YouTube Shorts creator, I want to caption my videos so that people without sound don't skip my videos.
4. As a restaurateur, I want to take fresh, juicy-looking photos of food so that customers want to eat at my restaurant.

Use a more formal **use case** to describe the requirement (the goal) as a journey

A sequence of actions taken by the "system" and the "actor"

Actor: As a parent ("actor"),

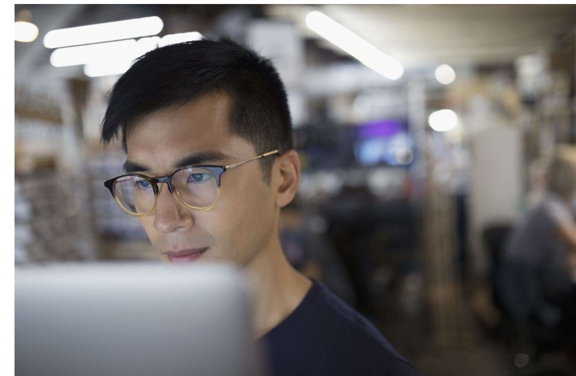
Goal: I want to take pictures of my young kids where they're all smiling.

Steps:

1. User arranges the family for a photo.
2. User presses the shutter 3 times in 5 seconds.
3. System saves 3 full-quality images.
4. User taps "gallery".
5. System opens the gallery and shows a button to "select a better moment".
6. User taps the button.
7. System creates and shows the "Best Take".
8. User taps "Save as copy".*
9. System shows user the saved copy in the gallery.

Closing thoughts: watch out for these as you engineer

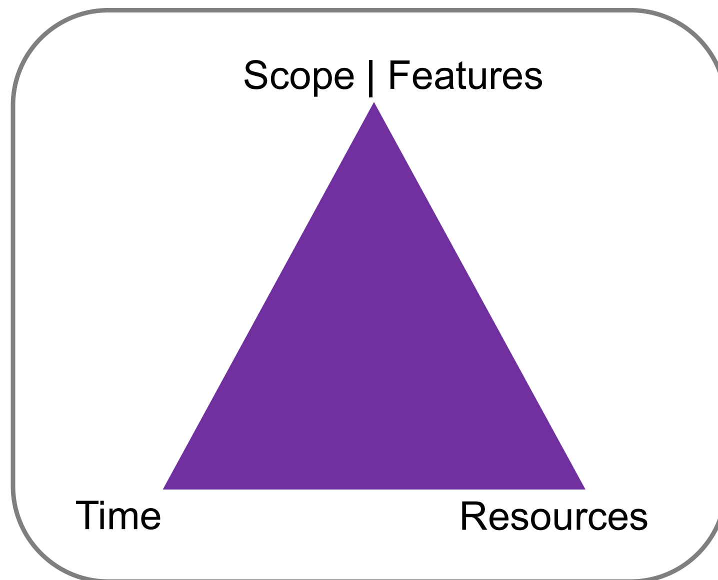
- Unclear scope leading to unclear requirements
- Finding the right balance (depends on customer, and the team):
  - Comprehensible vs. detailed
  - Graphics vs. tables and explicit and precise wording
  - Short and timely vs. complete and late
- Capturing implementation details instead of requirements
- Projecting your own models/ideas
- Feature creep



Feature creep?

For your project, consider **major** features (P0/P1s) and **stretch** features (P2s)

Feature creep is the gradual accumulation of features over time, beyond what was originally committed and/or actually needed



Why does it happen? Because features are fun!

- Developers like to code them
- Sales teams like to pitch them
- Users (think they) want them

Why can it be bad?

- Can put your project delivery at risk
- Too many options, more bugs, more delays, less testing, …

Today's Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. ~~What are techniques used to specify them?~~ Part-2 on Friday
4. Version control and git (Part-1)

Why use version control

11:51pm

Why use version control

Common App
Essay

11:51pm

Common App
Essay FINAL

11:57pm

Why use version control – **backup/restore**

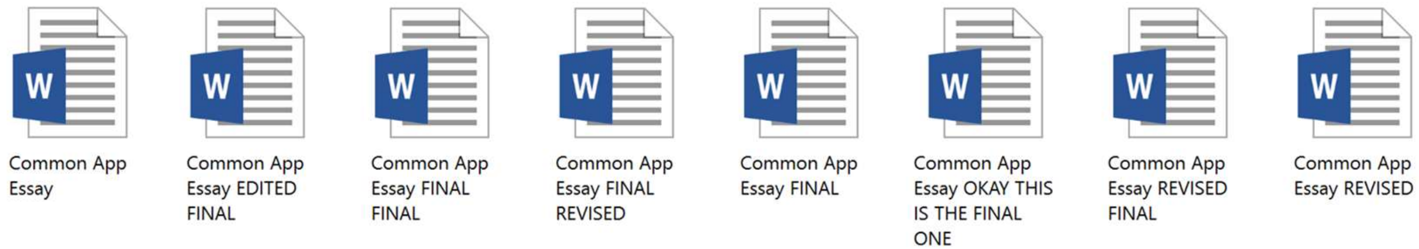
11:51pm

11:57pm

11:58pm

11:59pm

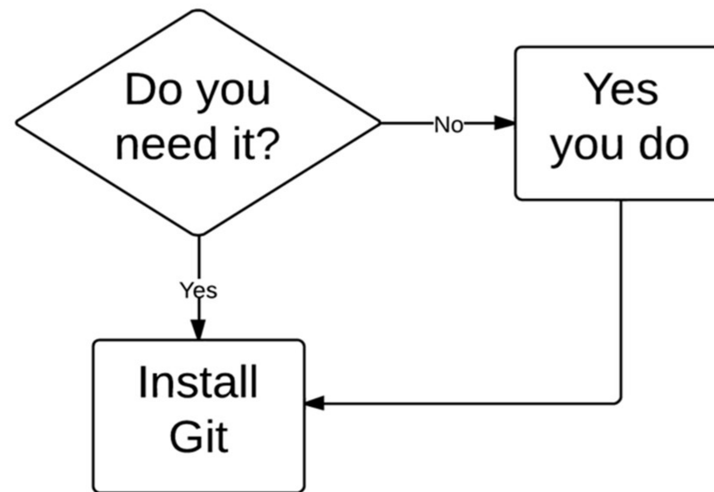
Why use version control – **teamwork**



Who is going to make sense of this mess?

Version control

Version control records changes to a set of files over time
This makes it easy to review or obtain a specific version (later)



Goals of a version control system

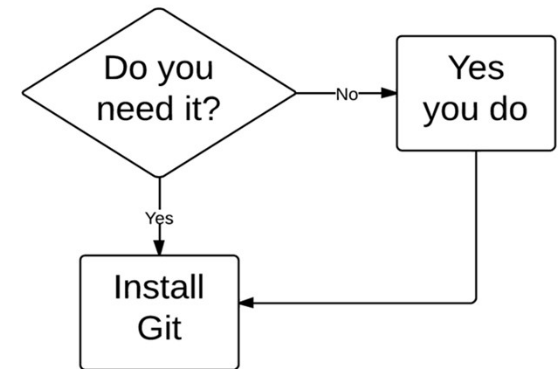
Version control records changes to a set of files over time

This enables you to:

- Keep a history of your work
  - Summary commit title
  - See which lines were co-changed
- Checkpoint specific versions (known good state)
  - Recover specific state
- Binary search over revisions
  - Find the one that introduced a defect
- Undo arbitrary changes
  - Without affecting prior or subsequent changes
- Maintain multiple releases of your product

AND it enables you to effectively coordinate with others working on the same work product

Who uses version control?



Everyone should use version control

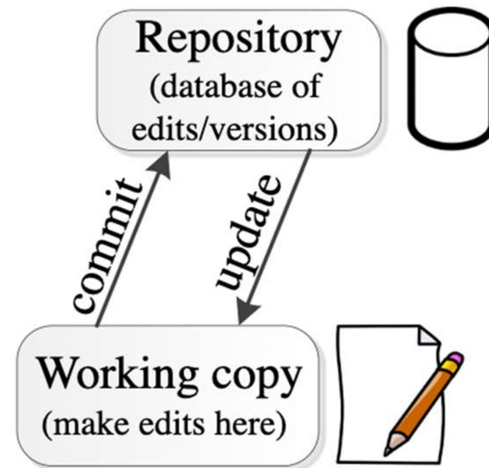
- Large teams (100+ developers)
- Small teams (2-10+ developers)
- Yourself (and your future self)

Example application domains

- Software development
- Hardware development
- Research & experiments (infrastructure and data)
- Applications (e.g., (cloud-based) services)
- Services that manage artifacts (e.g., legal, accounting, business, …)

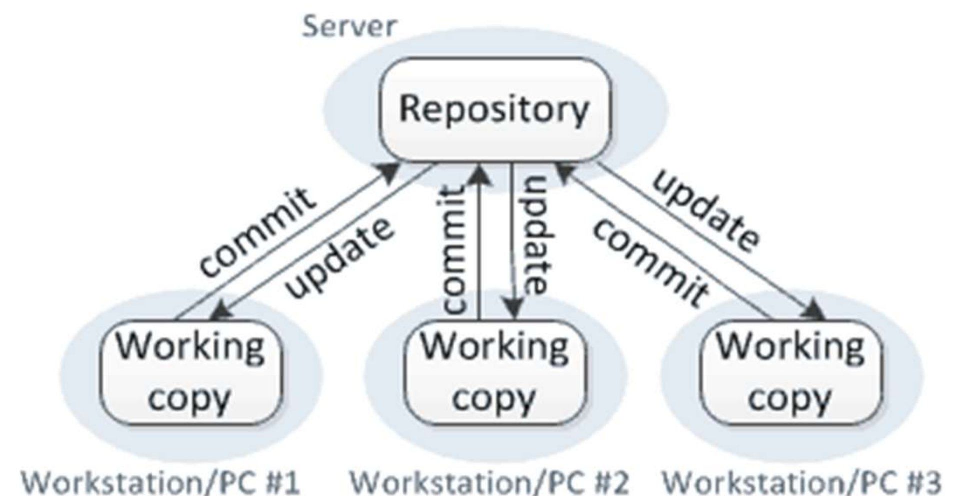
Version control repositories

Working by yourself



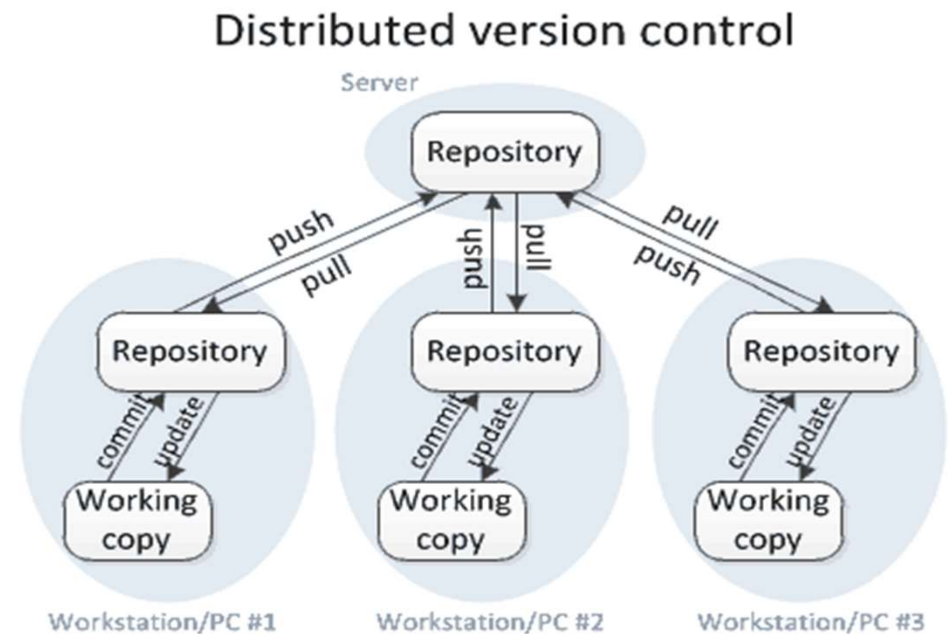
Centralized version control (older method)

- **One central repository**
  It stores a history of project versions
- Each user has a **working copy**
- A user **commits** file changes to the repository
- Committed changes are immediately visible to teammates who **update**
- Examples: SVN (Subversion), CVS

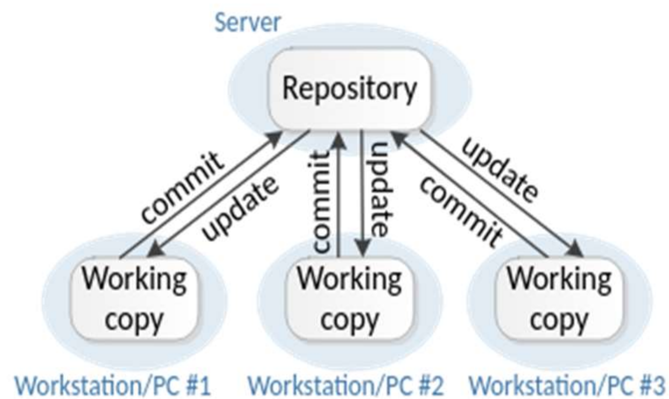


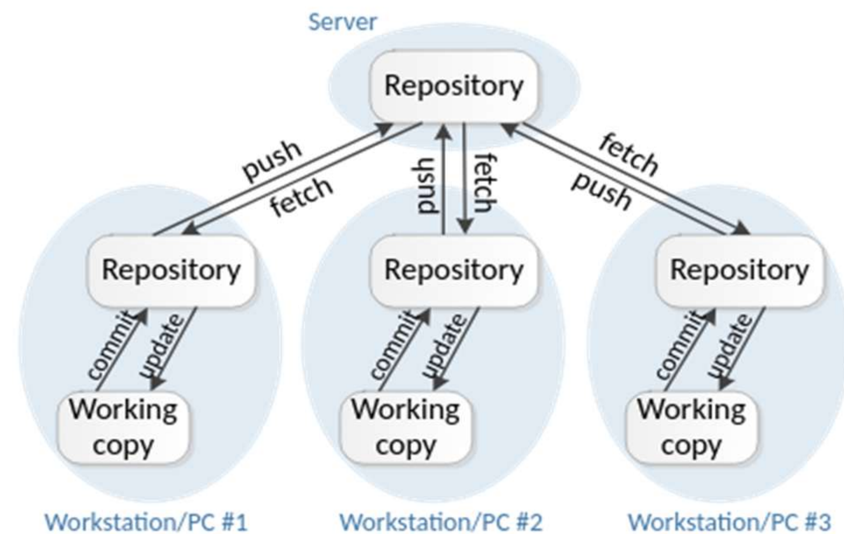
Distributed version control (newer method)

- **Multiple copies of a repository**
  Each stores its own history of project versions
- Each user **commits** to a **local** (private) repository
- All committed changes remain local unless **pushed** to another repository
- No external changes are visible unless **pulled** from another repository
- Examples: Git, Hg (Mercurial)



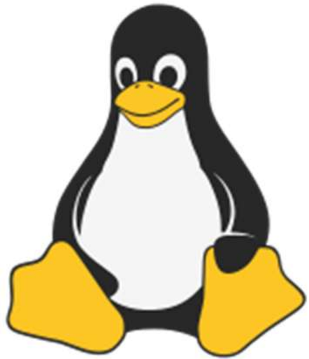
Two different version control modes





Version control with Git

git



Linux

Linus Torvalds - Wikipedia

Let’s do a little true/false quiz to see what you know already about git

PollEv.com/cse403wi

git: true or false

0 surveys underway

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

"git clone" is a git command

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

"git cherry-pick" is a git command

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

"git fetch" is a git command

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

"git fork" is a git command

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

"git branch" is a git command

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

"git pull" is a git command

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

A merge conflict in git arises as soon as two users change the same file

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

After editing a file, only some of the edits may end up in a "git commit"

True

False

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Coming up next

1. What are requirements and what is their value?
2. How can we gather requirements?

3. What are techniques used to specify requirements?
   - Use cases
   - Personas, user scenarios

4. More on version control with git and github