Compiler-level Optimisation Branch Prediction

A mini-lecture series

CSE498 Collaborative Design - Secure and Efficient C++ Software Development

01/15/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Mini lecture series

- Over the course of the semester
- One mini piece of information that I think is very useful
- Range from application design theory, programming, security, other programming concepts, etc.
- Also include a famous computer scientist and their contributions

Most upvoted stackoverflow question

- Take a long array of random items (say 100,000 items)
  - int random_array[] = {7, 116, 2, 236, 68, 70, 227, 75, 170, 119}
- random_array.sort()
- int sum
- for(auto& item: random_array) {
  - if(item > 128): sum += item;
- }

- https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array

Some things to consider

- Sorting is lower bounded by nlogn
- You must loop through the entire array

Why is processing a sorted array faster than processing an unsorted array?

Ask Question

Asked 12 years, 6 months ago Modified 29 days ago Viewed 1.9m times

27399

In this C++ code, sorting the data (*before* the timed region) makes the primary loop ~6x faster:

```
#include <algorithm>
#include <ctime>
#include <iostream>

int main()
{
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;

    // !!! With this, the next loop runs faster.
    std::sort(data, data + arraySize);

    // Test
    clock_t start = clock();
    long long sum = 0;
    for (unsigned i = 0; i < 100000; ++i)
    {
        for (unsigned c = 0; c < arraySize; ++c)
        {   // Primary loop.
            if (data[c] >= 128)
                sum += data[c];
        }
    }

    double elapsedTime = static_cast<double>(clock()-start) / CLOCKS_PER_SEC;

    std::cout << elapsedTime << '\n';
    std::cout << "sum = " << sum << '\n';
}
```

- Without `std::sort(data, data + arraySize);`, the code runs in 11.54 seconds.
- With the sorted data, the code runs in 1.93 seconds.

Branch prediction

35130

+2200

You are a victim of branch prediction fail.

What is Branch Prediction?

Consider a railroad junction:

Image by Mecanismo, via Wikimedia Commons. Used under the CC-By-SA 3.0 license.

Guessing

- If you guess right, then the train keeps going
- If you guess wrong, then the train must stop, backup, and restart

- Modern processors are slow and have long pipelines. This means they take forever to “warm up” and “slow down”

- Compiler is trying to identify a pattern and follow it

Results

- If the array is sorted, then the compiler essentially will be correctly guessing most of the time if it used the previous data
- If the array is not sorted, then it is random guessing

- Sorted

```
T = branch taken
N = branch not taken

data[] = 0, 1, 2, 3, 4, ... 126, 127, 128, 129, 130, ... 250, 251, 252, ...
branch = N  N  N  N  N  ...   N    N    T    T    T  ...   T    T    T  ...

       = NNNNNNNNNNNN ... NNNNNNNTTTTTTTTT ... TTTTTTTTT  (easy to predict)
```

- Not sorted

```
data[] = 226, 185, 125, 158, 198, 144, 217, 79, 202, 118,  14, 150, 177, 182, ...
branch =   T,   T,   N,   T,   T,   T,   T,  N,   T,   N,   N,   T,   T,   T  ...

       = TTNTTTTNTNNTTT ...   (completely random - impossible to predict)
```

Person of the day
Bjarne Stroustrup

- Developed the C++ language
- “C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.”

Black-box

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

01/27/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Black-box vs White-box

- Usually used in a testing context, but also system design
- Sometimes, AI or machine learning models are called black-box

- White-box means you have full access to the code
- Black-box means you do not have any information on the code
  - Focuses on the system's *external observable behaviour*

- In other words: view the system in terms of the inputs and outputs
- Input Transformation

Examples

- Connect to a web
  - Order ramen at a restaurant, do you care how they make it?
  - Type in google.com into firefox, do you care what happens under the hood?
  - Get on a train from Lansing to Chicago?

Black-box model of a function

Some magic occurs here

Black-box model of a function

- When you develop a function, you can see its whole content
- But other people sees that function as a black-box entity
- They may include the .h file associated
  - They won’t see the code itself

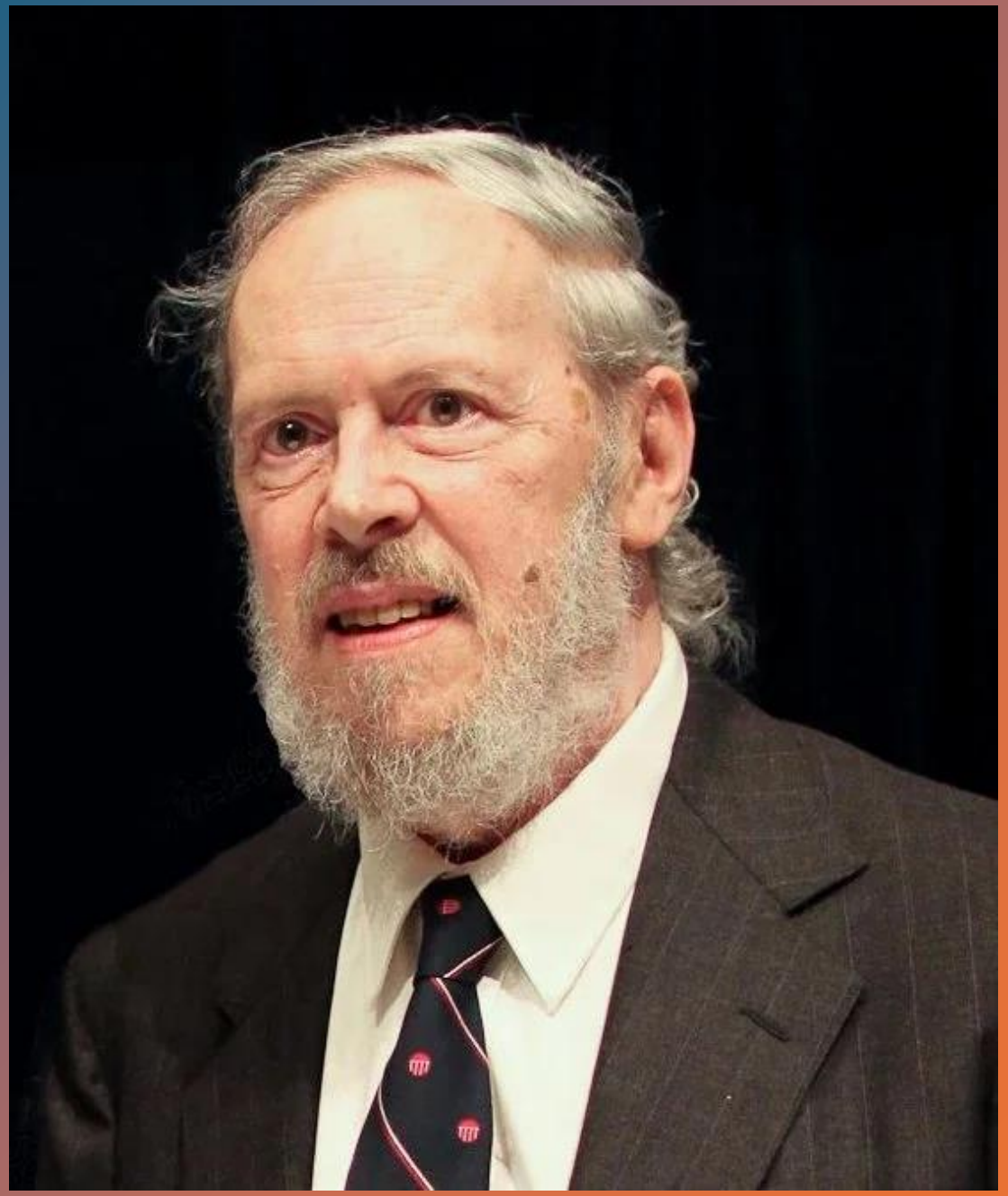
- What do you care about when you import a function?
  - Functionality?
  - Efficiency?
  - How the code was written?

Why is it important to view it as black-box?

- A person who uses the program or function should not need to know how it is coded
- **Procedural Abstraction**
  - A developer using your function should not access to code body to see how it works
- This is also known as information hiding

Advantages

- Allows you to use a test-driven development approach effectively
  - Testers do not need technical knowledge, focus on what types of output should the system return given certain inputs
- Modularity
  - Forces you to break apart your code into reasonable pieces
  - Each piece should serve a single objective



Persons of the day

Dennis Ritchie,
Ken Thompson,
and Brian Kernighan
(and the rest of bell labs)

- Creator of the C language and the UNIX operating system
- Development started in 1970s in Bell labs
- How did C++ get its name?
- 1967 - Basic Combined

Programming Language (BCPL)

➔1969 - B [Ritchie and Thompson]

➔1972 - C [Ritchie and Kernighan]

➔1985 - C++ [Stroustrup]

Git

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

01/29/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Git, the version manager

- Keeps track of code (file in general) changes
- Collaborate with other when working on the same files

- Allows you to branch off and work on a different part of the project, remerge with main once you are ready.
- Committing is essentially “saving” a state of your repository, allowing you to revert back to this point

Cloning a repo

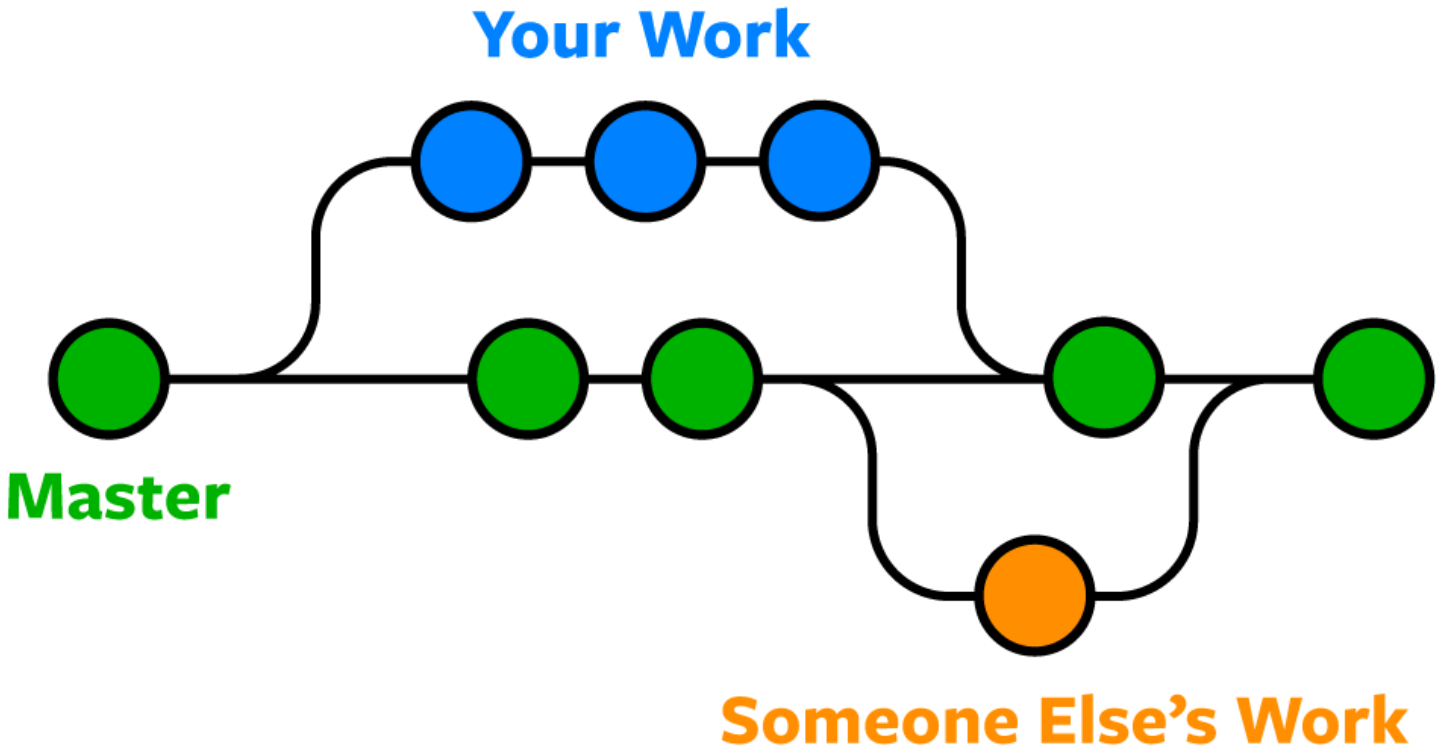
- Copies the file from the repository
- Create an empty directory, and use the following command:
  - Git clone https://github.com/CSE498/CSE498-Spring2025.git
- Should download our current empty folders into your local folder

Branching

- Usually, the **Main** repo only contains fully functional code
- **You should never push to main, ever**
  - Seriously, you will get fired
- When you work on a feature, branch off (so that you are not affecting other teams when working)
- Every company might have a different setup, good to check out the branching to see

- Use gitlog to check current branching history

Branching

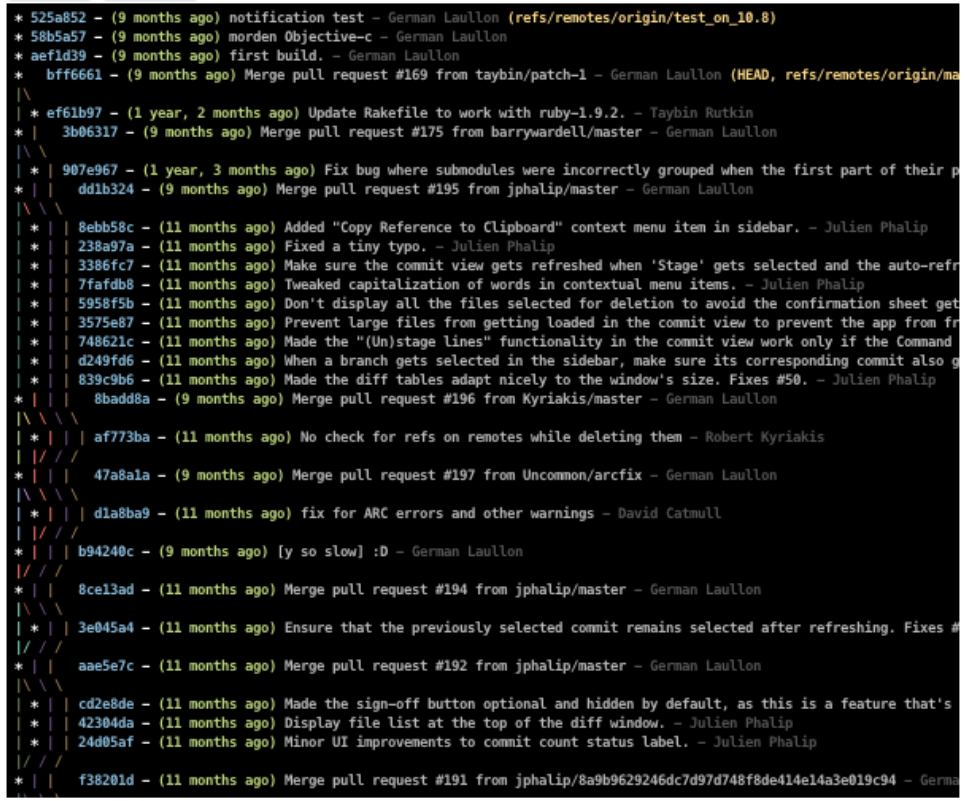


https://www.nobledesktop.com/learn/git/git-branches

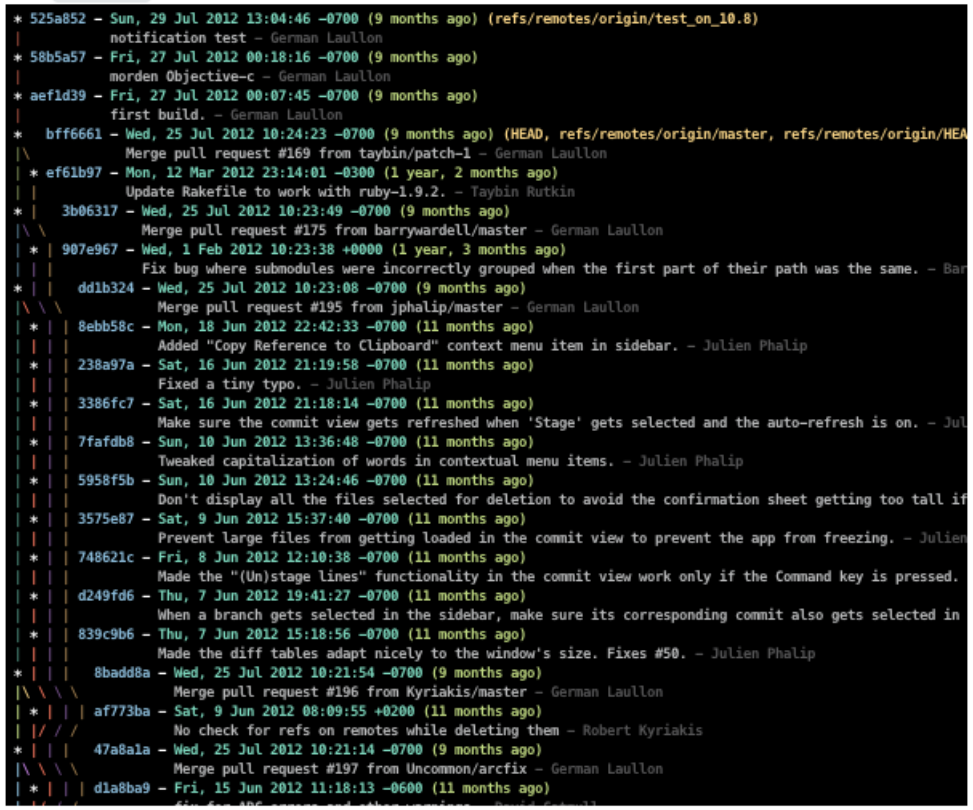
Visualisation

- https://stackoverflow.com/questions/1057564/pretty-git-branch-graphs

`git lg` / `git lg1` looks like this:



and `git lg2` looks like this:



Creating a new branch

- git checkout –b NEW_BRANCH_NAME
- This is equivalent to doing
  - Git branch NEW_BRANCH_NAME // creates new branch
  - Git checkout NEW_BRANCH_NAME // switch to the branch

```
C:\Users\Veda\Documents\cse498\gittest>git checkout -b dev
Switched to a new branch 'dev'
```

- If you do *git status* now, then you should see

```
C:\Users\Veda\Documents\cse498\gittest>git status
On branch dev
nothing to commit, working tree clean
```

Working on local branch

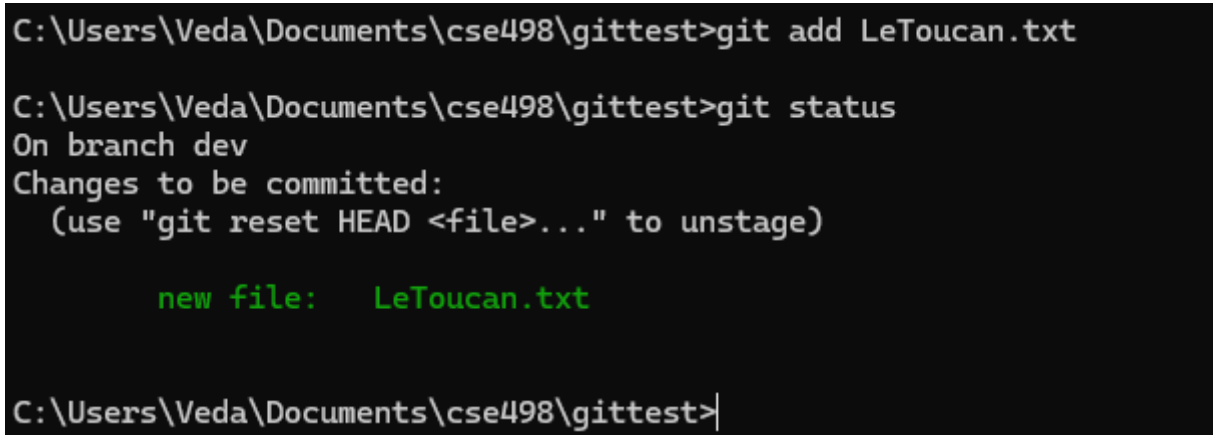
- Let's supposed I have created some content or changed file:

```
C:\Users\Veda\Documents\cse498\gittest>echo "praise" >> LeToucan.txt

C:\Users\Veda\Documents\cse498\gittest>git status
On branch dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        LeToucan.txt

nothing added to commit but untracked files present (use "git add" to track)
```

- I would want to *stage* the change
- **Staged** files are those that are ready to be **committed** to the repo

Staging

- Git add . (all local file in the current directory)
- Git add LeToucan.txt
  - Now lit up green
- To unstage, use:
  - Git rm --cached …



Committing

- Once you are ready to commit (finished a function or hit a good save point), you should move from **stage** to **commit**.
- Please always include a descriptive change
  - NOT: “fixed some bugs”
  - NOT: “added some logic”
  - NOT: “changes”
  - GOOD: “implemented logic for calculating Euclidean Distance”
- Git commit –m “initialised the letoucan text file”

Committing

```
C:\Users\Veda\Documents\cse498\gittest>git add .

C:\Users\Veda\Documents\cse498\gittest>git commit -m "initalised the letoucan text file"
[dev 4671d14] initalised the letoucan text file
 1 file changed, 1 insertion(+)
 create mode 100644 LeToucan.txt

C:\Users\Veda\Documents\cse498\gittest>git status
On branch dev
nothing to commit, working tree clean

C:\Users\Veda\Documents\cse498\gittest>git log
commit 4671d14842fcae4eeb55a1c206015195f455eafc (HEAD -> dev)
Author: Kira <[redacted]com>
Date:   Tue Jan 28 17:41:15 2025 -0500

    initalised the letoucan text file
```

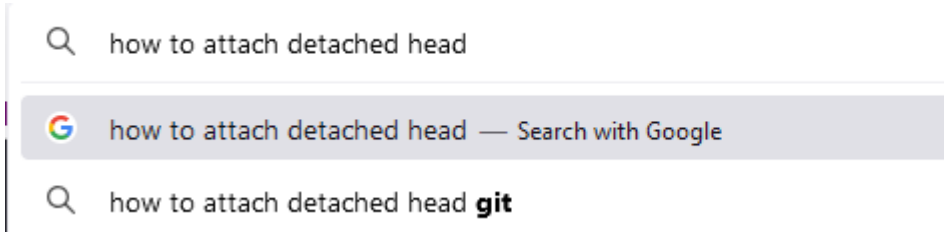
Reverting to a previous state

- I made a bad commit here, can i go back?
- Yes, git checkout 4671d1 (the hash of the commit you want to go back to)

```
C:\Users\Veda\Documents\cse498\gittest>git commit -m "oh no this commit have bugged out the file"
[dev 6131a0b] oh no this commit have bugged out the file
 1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\Veda\Documents\cse498\gittest>git status
On branch dev
nothing to commit, working tree clean

C:\Users\Veda\Documents\cse498\gittest>git log
commit 6131a0b1f0f060683a3c8f18408489ebe4ab131d (HEAD -> dev)
Author: Kira <[redacted]com>
Date:   Tue Jan 28 17:46:22 2025 -0500

    oh no this commit have bugged out the file

commit 4671d14842fcae4eeb55a1c206015195f455eafc
Author: Kira <[redacted].com>
Date:   Tue Jan 28 17:41:15 2025 -0500

    initalised the letoucan text file
```

Reattaching a detached head

- I am now in a detached state, and I can reattach it through merge
- Git branch tmp (creates a new branch with this commit)
- Git checkout dev
- Git merge tmp




```
C:\Users\Veda\Documents\cse498\gittest>git checkout 4671d1
Note: checking out '4671d1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 4671d14 initalised the letoucan text file
```

Conflicts

- If you try to merge two branches, you may encounter conflicts
- You must resolve these before merge can continue
  - Select which code segment you want to keep

```
C:\Users\Veda\Documents\cse498\gittest>git merge tmp
Auto-merging LeToucan.txt
CONFLICT (content): Merge conflict in LeToucan.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
<<<<<<< HEAD
"praise"
OH NO I HAVE CORRUPTED OR SCREWED UP THIS FILE
=======
"praise"
yay we fixed the file!
>>>>>>> tmp
```

What does my branch now look like?

```
C:\Users\Veda\Documents\cse498\gittest>git log --oneline --graph --decorate
*   22e9092 (HEAD -> dev, origin/dev) fixed conflicts in toucan file from merging
|\
| * afc58be (tmp) fixed letoucan file
* | 6131a0b oh no this commit have bugged out the file
|/
* 4671d14 initalised the letoucan text file
* 8805aa6 (origin/main, main) first commit
```

Ready to actually upload to server (git)?

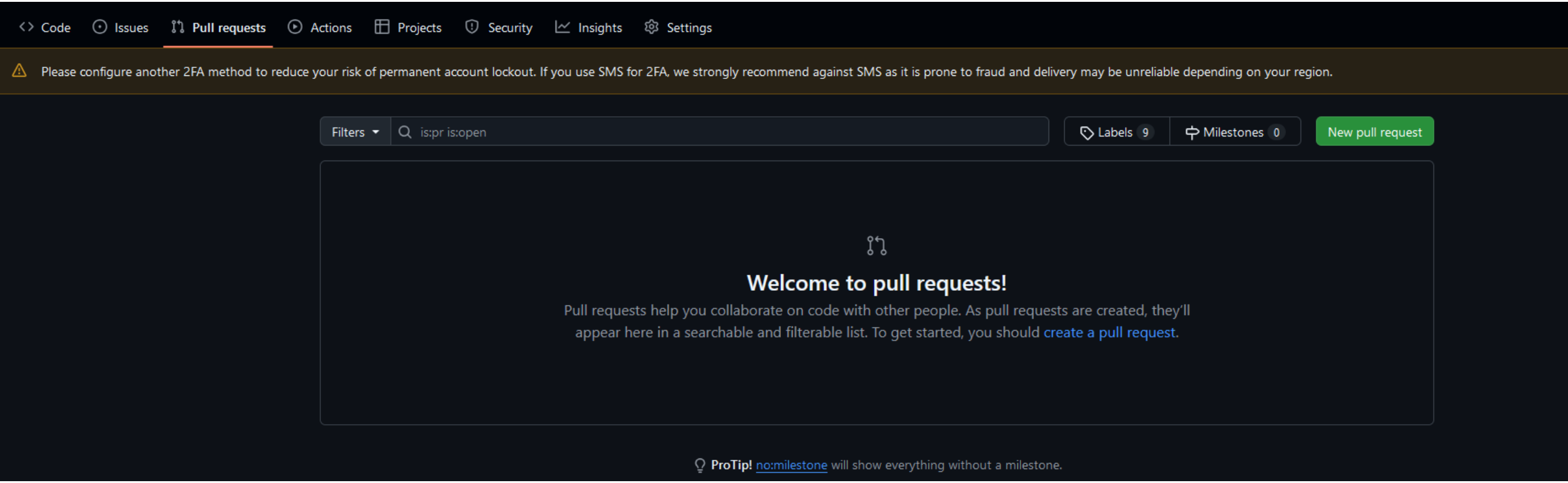
- Once you are ready to share your code
- Git push
- Now other people on your branch (or really anyone on the repo) can use git fetch or git pull to get your pushed code.

```
C:\Users\Veda\Documents\cse498\gittest>git push
Counting objects: 6, done.
Delta compression using up to 32 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 636 bytes | 636.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/EvilJuiceBox/testrepo.git
   6131a0b..22e9092  dev -> dev
```

Get updates from main

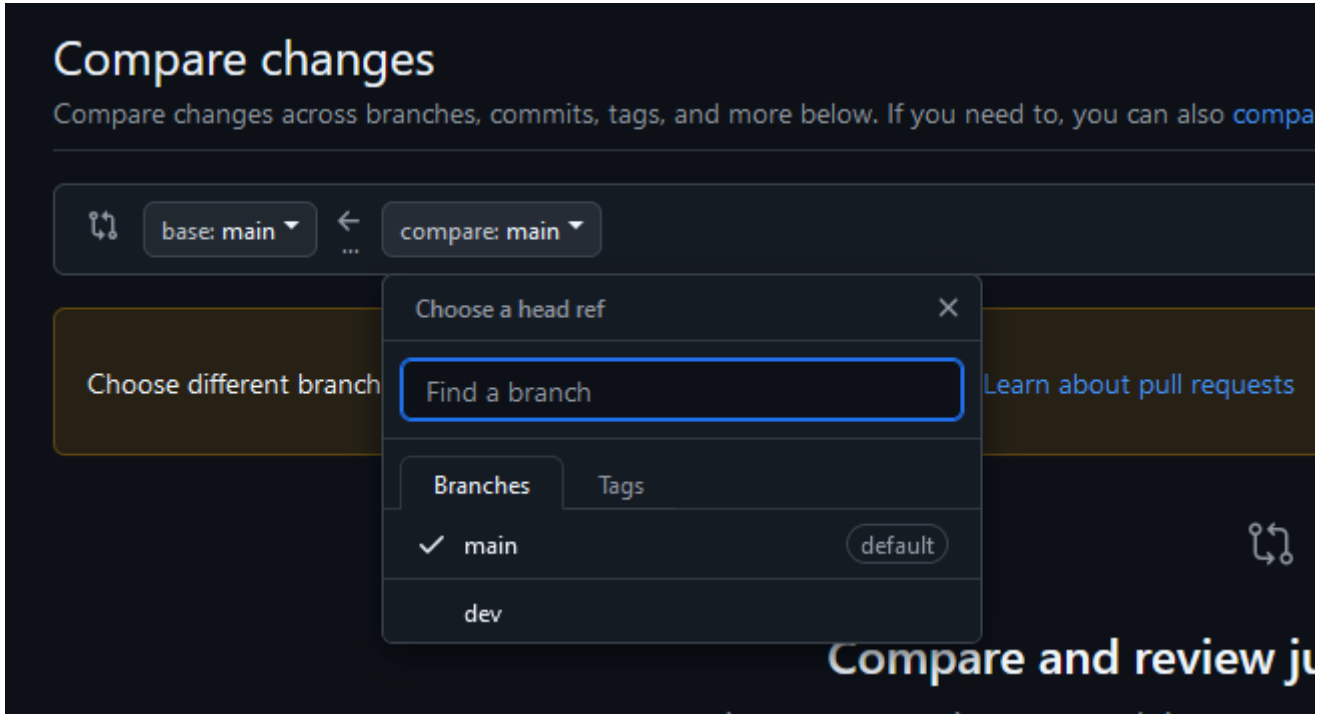
- Git fetch
- Git pull
- Gets upstream and updates your repo with the new code from your parent branch

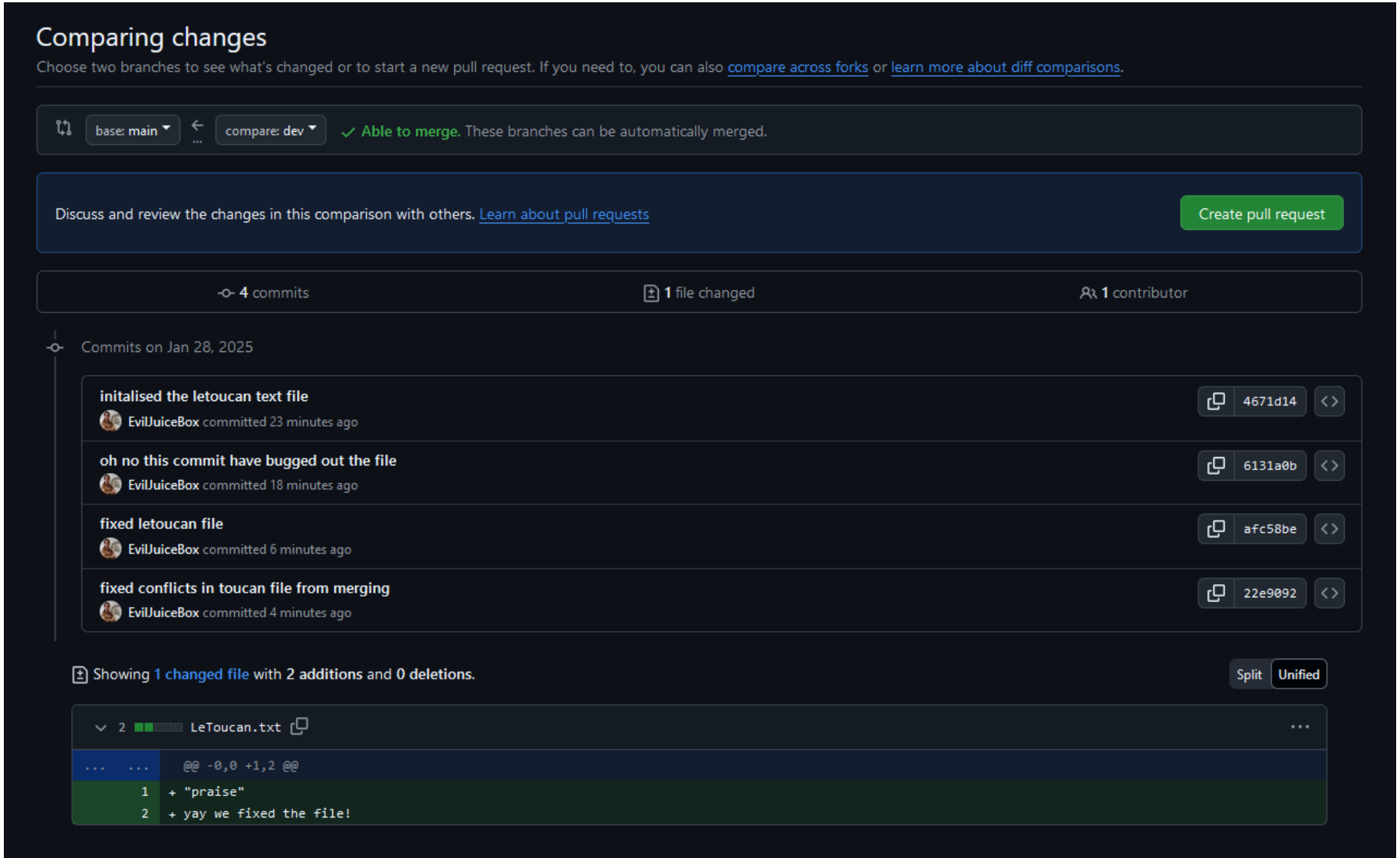
Pull request

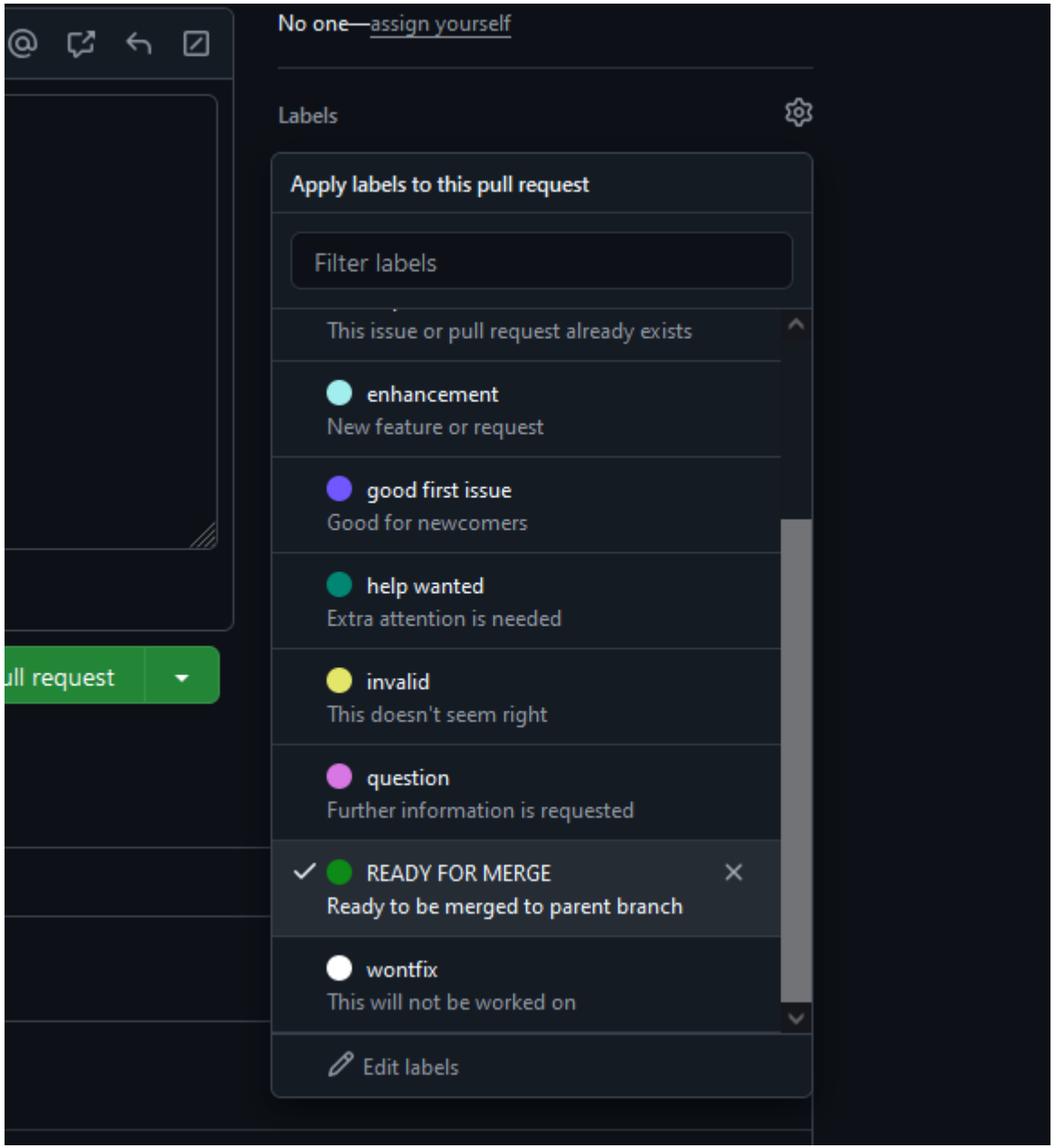


Pull request

- Set your target branch and destination branch




Comparing changes
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.
base: main
compare: dev
Able to merge. These branches can be automatically merged.
Discuss and review the changes in this comparison with others. Learn about pull requests
Create pull request
4 commits
1 file changed
1 contributor
Commits on Jan 28, 2025
initalised the letoucan text file
EvilJuiceBox committed 23 minutes ago
4671d14
oh no this commit have bugged out the file
EvilJuiceBox committed 18 minutes ago
6131a0b
fixed letoucan file
EvilJuiceBox committed 6 minutes ago
afc58be
fixed conflicts in toucan file from merging
EvilJuiceBox committed 4 minutes ago
22e9092
Showing 1 changed file with 2 additions and 0 deletions.
Split
Unified
2
LeToucan.txt
@@ -0,0 +1,2 @@
1 + "praise"
2 + yay we fixed the file!


No one—assign yourself
Labels
Apply labels to this pull request
Filter labels
This issue or pull request already exists
enhancement
New feature or request
good first issue
Good for newcomers
help wanted
Extra attention is needed
invalid
This doesn't seem right
question
Further information is requested
READY FOR MERGE
Ready to be merged to parent branch
wontfix
This will not be worked on
Edit labels

Open EvilJuiceBox wants to merge 4 commits into `main` from `dev`

Conversation 0 | Commits 4 | Checks 0 | Files changed 1 | +2 −0

EvilJuiceBox commented now

Ready for merge to main.

Code reviewers @EvilJuiceBox

EvilJuiceBox added 4 commits 27 minutes ago

initalised the letoucan text file	4671d14
oh no this commit have bugged out the file	6131a0b
fixed letoucan file	afc58be
fixed conflicts in toucan file from merging	22e9092

EvilJuiceBox added the READY FOR MERGE label now

Require approval from specific reviewers before merging

Rulesets ensure specific people approve pull requests before they're merged.

Add rule

Continuous integration has not been set up

GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request | You can also open this in GitHub Desktop or view command line instructions.

Try the new merge experience

Reviewers
No reviews
Still in progress? Learn about draft PRs

Assignees
No one—assign yourself

Labels
READY FOR MERGE

Projects
None yet

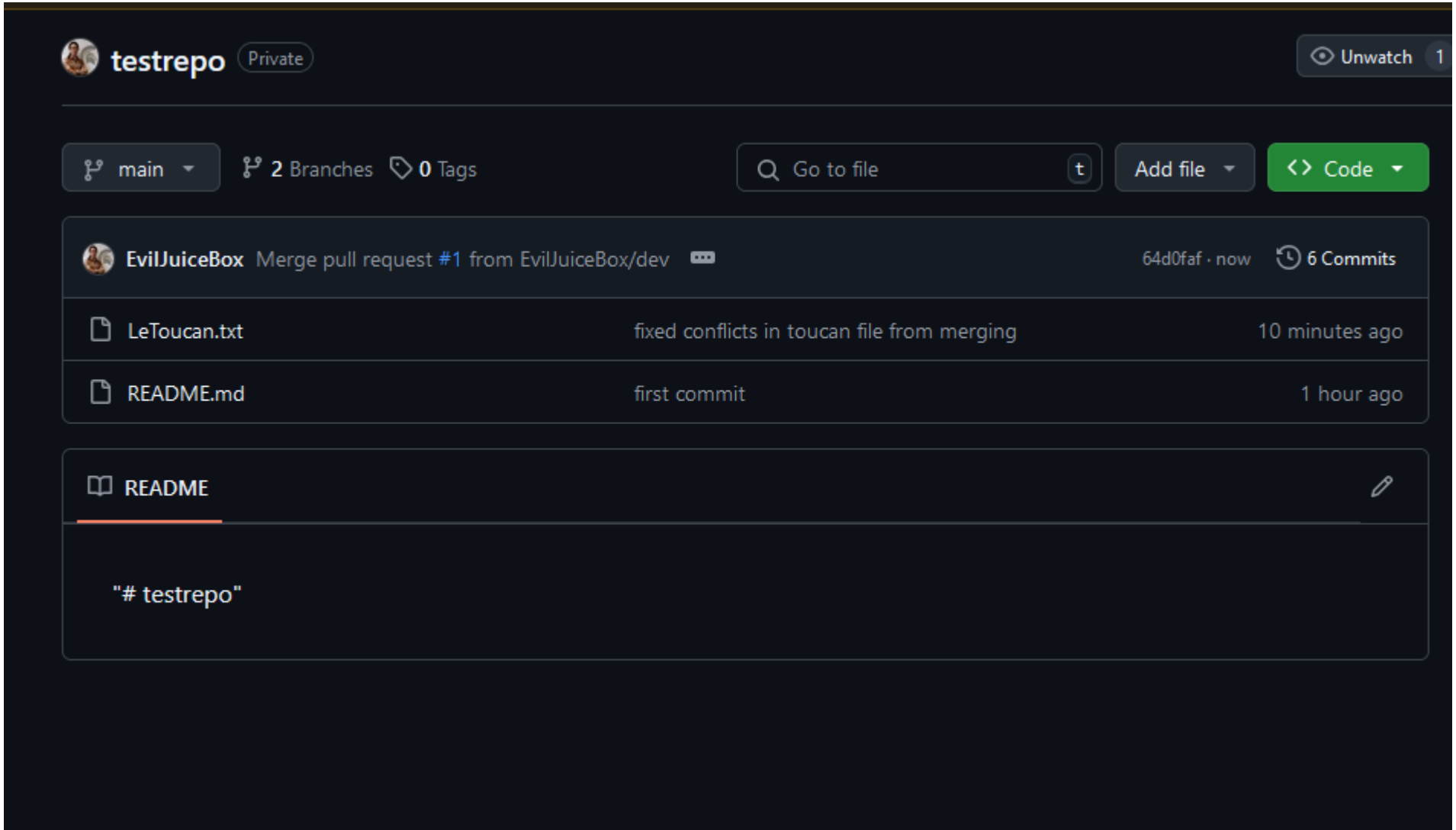
Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

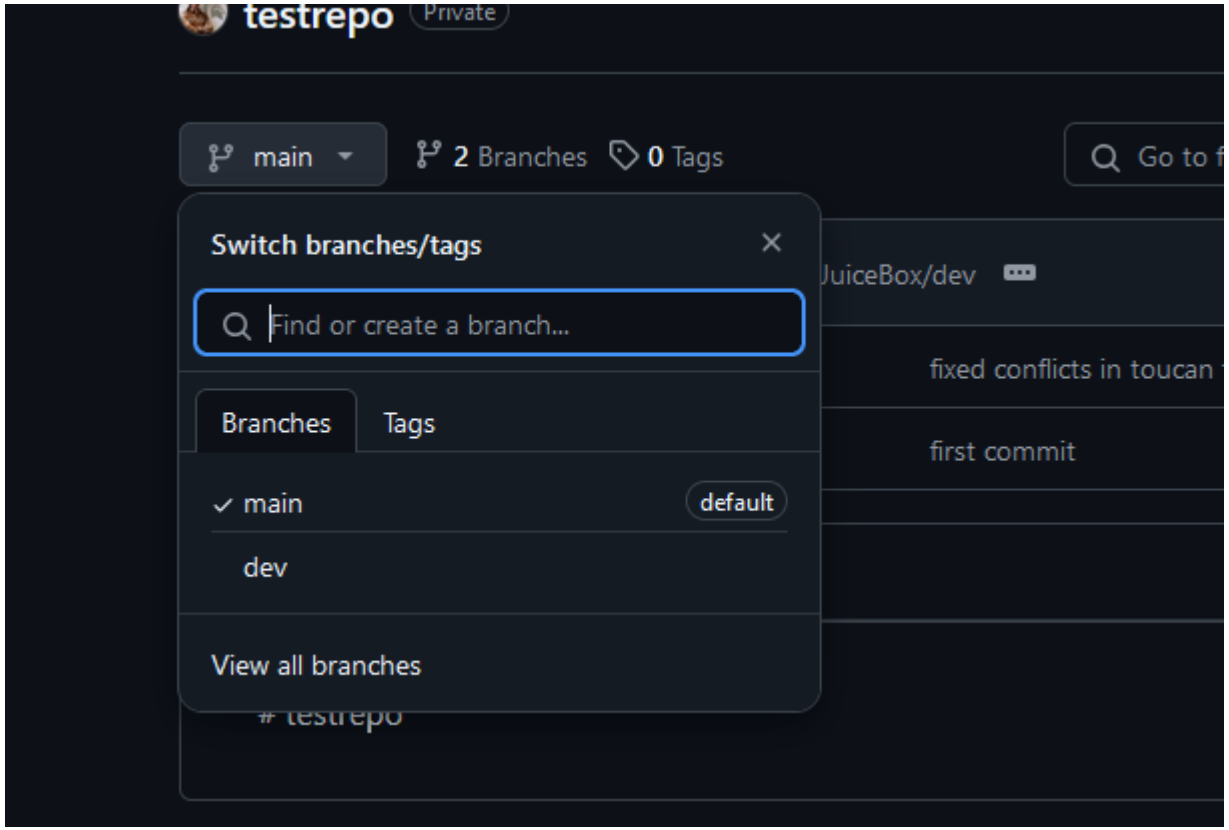
Notifications Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

1 participant

Shows up on main, yay!



By the way, you can see all the current branches here as well



Comments

- 1) Create your team branch, create a pull request
  - Please name it [TEAMNAME] … (e.g., [Stroustrup] Random Class)
- 2) Immediately tag it as NOT READY TO MERGE
- 3) Make sure everyone switches to this branch before you work on it (so you do not work on another team's branch or main)
- 4) Mark it as READY TO MERGE and @eviljuicebox or Dr. Ofria when you are ready. Feel free to DM us on discord if we do not reply

Resources if you are not comfortable

- https://git-scm.com/docs/gittutorial
- https://www.w3schools.com/git/
- https://learngitbranching.js.org/ << Very good
- https://gitimmersion.com/

Person of the day
Linus Torvalds

- Creator of Linux operating system
- GNU
- Also created Git
- “I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.”

Yes, this is the first version of git's readme

- I choose to believe its so people can type "git gud"

```
        GIT - the stupid content tracker

"git" can mean anything, depending on your mood.

 - random three-letter combination that is pronounceable, and not
   actually used by any common UNIX command.  The fact that it is a
   mispronounciation of "get" may or may not be relevant.
 - stupid. contemptible and despicable. simple. Take your pick from the
   dictionary of slang.
 - "global information tracker": you're in a good mood, and it actually
   works for you. Angels sing, and a light suddenly fills the room.
 - "goddamn idiotic truckload of sh*t": when it breaks

This is a stupid (but extremely fast) directory content manager.  It
doesn't do a whole lot, but what it _does_ do is track directory
contents efficiently.
```

Resources link

- https://cse.msu.edu/~chanken1/resources.html#git

Non-conventional Testing Techniques

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/03/2025

Kira Chan

https://cse.msu.edu/~chanken1/

How do we test our code?

- Written test?
  - You have some domain knowledge, and you are using them to test your program
  - You can independent verification
  - Random people on the street?
- Relatively easy to test if you have 1 input parameter (is it?)
- 2 parameters? R^2
- 3 parameters? R^3
- 4 parameters? R^4
- …

Test cases

- A successful test case is one that fails (catches an error)
  - Why?
- If you do not know that if a successful test case means
  - 1) Your test case is bad
  - 2) Your code is bad

A software tester walks into a bar

- Runs into a bar.
- Crawls into a bar.
- Dances into a bar.
- Flies into a bar.
- Jumps into a bar.
- And orders:
- a beer.
- 2 beers.
- 0 beers.
- 99999999 beers.
- a lizard in a beer glass.
- -1 beer.
- "qwertyuiop" beers.
- Testing complete.

- A real customer walks into the bar and asks where the bathroom is.
- The bar goes up in flames.

Testing is hard

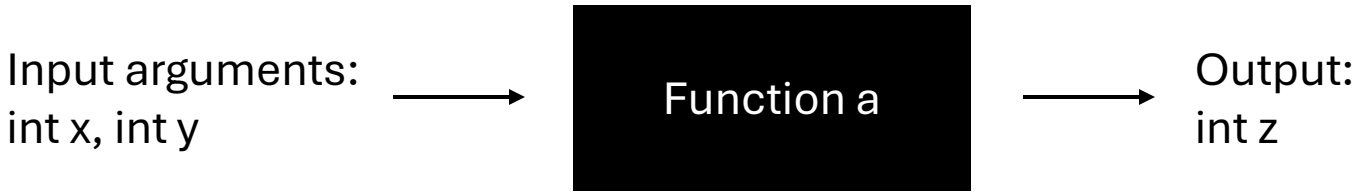
- Humans test with bias
- When you test your code, you test it *gently*

Towards Automated Testing

- Can we automate some testing?
- Some things we will cover today
- Fuzz testing (fuzzing)
- Evolutionary Search-based testing

Black-box model of a function

- Remember the concept of black-box testing?
- Given some input, transform it into outputs.



Some magic occurs here

Fuzzing

- Remember the concept of black-box testing?
- Given some input, transform it into outputs.
- What if we provide invalid, unexpected, or random data as input to monitor for crashed
- Developed by University of Wisconsin – Madison Professor Barton Miller
  - Uses external “noise” to test if a system is tolerant
- Found that traditional UNIX, mac, and Windows programs would routinely crash with unexpected inputs

Types of fuzzing

- Coverage-guided fuzzing
  - Tracks “code coverage”
  - Trigger all the code logic to make sure output is right
- Mutation-based fuzzing
  - Modifies existing data and input to find crashes
  - Advantages: you know what input crashed it, you can trace it

Use cases

- Incorrect behaviours
- Input errors and crashes
- Can catch memory issues
  - Memory leaks
  - Buffer overflows
  - Use after frees
  - Stack overflows
  - Crashes, etc.
- Security issues
- Very good at finding odd programming errors

Evolutionary Inspired Search Testing

- Leverage things we see in nature
- Particle Swarm Optimisation
- Ant Colony Optimisation

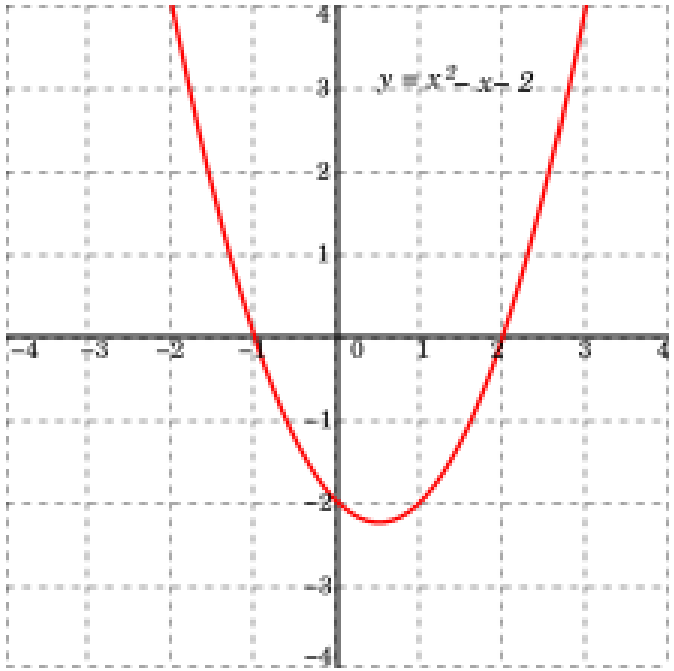
Skipped a lot of details

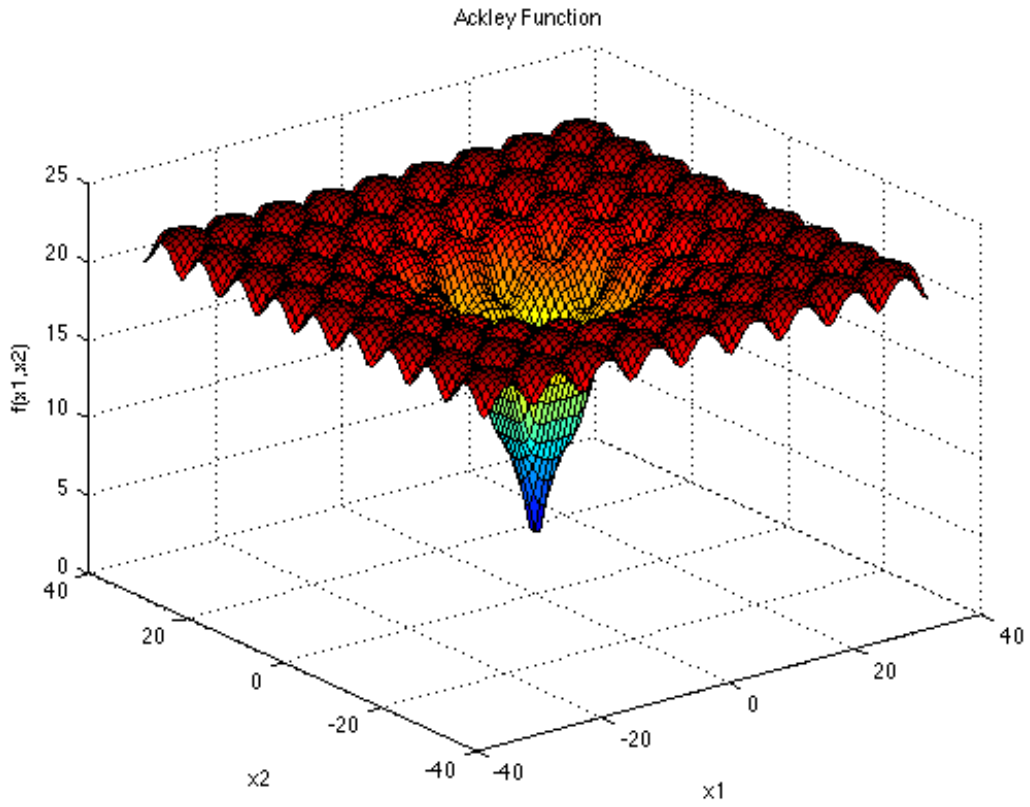
- I am leaving out a lot of details here, otherwise we will spend a semester on this
- If you are interested, then this is definitely grad school material
- We have a course **CSE848 Evolutionary Computation** taught by Dr. Wolfgang Banzhaf
  - Very big figure in the evolutionary computing theory
- MSU actually as a lot of famous people in this field

Evolutionary Search-based Testing

- Inspired by Darwinian Evolution
- Map solutions to a ``genome''
- Use nature inspired techniques to evolve solutions (a population)
- Mutation, crossover, and selection mechanism
- Evaluate an individual based on how well they perform (i.e., how close to error)
- Individuals who have high scores get to reproduce
- Slowly get to the solution

Solving for optimum





On searching

- I am going to ``punt'' the search stuff later
- I will cover how most problem are really just a search / optimisation problem in principle

Completeness

- When do you stop testing?

- These techniques are not meant to replace traditional testing
- They are complementary, often meant to help find bugs you otherwise would not due to developer bias

- You can make the code during the day, and fuzz test it overnight

Person of the Day
John Henry Holland

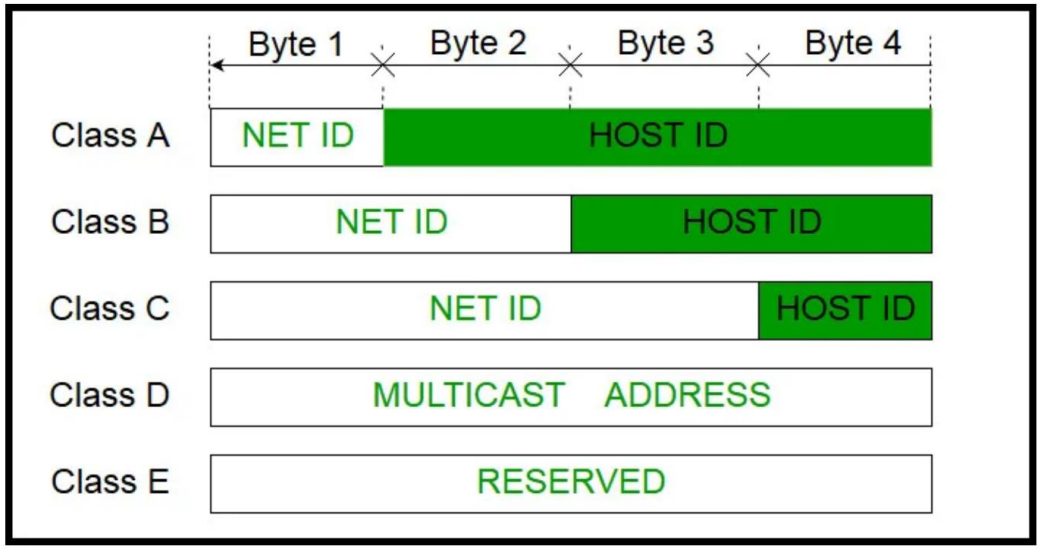
- Pioneer of Evolutionary Computing field
- Introduced genetic algorithms in 1960s
- Received his M.S. and Ph.D. at University of Michigan and have been a professor there

Something I forgot to mention last lecture

- Full exhaustive testing (i.e., testing every input) is actually just as good as random testing given enough input.

Fun fact of the day

- Do you know why msu has a 35.xx.xx.xx IP address?
- We fall under the class A network, what has much more IP addresses
- We are part of Merit Network, a non-profit group
  - MSU
  - University of Michigan
  - Wayne State University
- Originally for shared computing
- Backbone for the NSFNET



Testing Driven Development

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/05/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Testing (again)

- A software development process that promotes the idea of writing automated test **before** writing the code
- Very tied with the *Agile* development process
- Unit-test
  - Describes a small piece of test case that test a single (isolated) component

Pipeline (actually works well with leetcode)

1. Understand the problem space and inputs (behaviour of the sys)
2. Develop the test cases
    a) Based on what you expect to be easy correct inputs
    b) Correct harder inputs
    c) Edge cases (divide by zeros, negatives, etc.)
3. Run the tests, make sure they should fail
4. Write simplest code that passes the tests (building functionality)
5. Be sure all test cases pass
6. Refractor as needed, while ensuring the test still pass
    a) Move code to where it belongs
    b) Remove duplicated code
    c) Split into multiple methods, etc.

Advantages

- TDD ensures that all code have been tested (since you write the test first)
- Provides confidence for developers in their code (double-edged)
- Well-documented code
- **Facilitates Continuous Integration –** helps CI/CD process
  - **Integration testing:** Future code pushes will not break existing code

Disadvantages

- Increased code written
- False security
- Writing comprehensive test cases is time consuming (maybe not a bad thing)

Verification and Validation (V&V)

- Verification refers to: Does the product / software work correctly (according to requirements)
  - Verify that your product works as designed
- Validation refers to: Does the product / software fit the customer's needs
  - Validate that you build the right product

- Independent Verification and Validation (IV&V): an independent third party tests your software

Broad types of testing

- Unit Testing
- Integration Testing
- Acceptance Testing
- Regression Testing
- Compatibility Testing
- Performance Testing

Unit Testing

- Tests a single function / unit of your code

- Verification or Validation?
  - Verification

Integration Testing

- Does your component work with the system as a whole?
- Think:
  - Will your code break another part of the system
  - Group 1’s code when pushed, should not lead to a failure in Group 3’s test cases

- Verification or Validation?
  - Verification

Regression Testing

- Checks if the application behaves the way it used to (before your new changes / patch)
- Verification or Validation?
  - Validation (?)

Acceptance Testing

- Evaluate the compliance of the system built with the business or customer's needs
- Assess whether it is acceptable for delivery or not

- Definition from ISTQB:
  - Formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria[3] and to enable the user, customers or other authorized entity to determine whether to accept the system
- Verification or Validation?
  - Validation

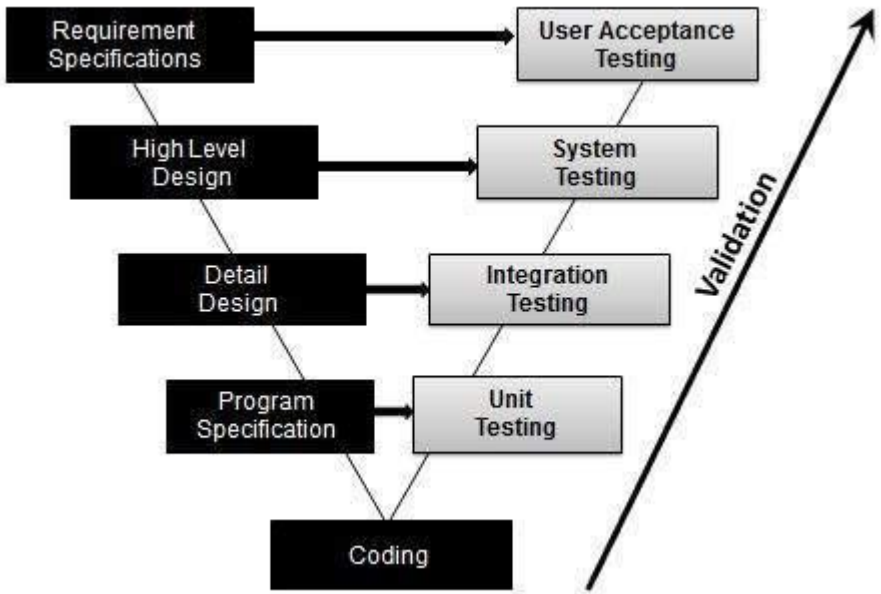
Performance Testing

- Test how well your software performance under certain workload, conditions, on various machines, etc.
- Verification or Validation?
  - ?

Compatibility Testing

- Makes sure that it works on all supported (or extended) platforms
- Verification or Validation?
  - ?

V model summary



Broad types of testing

- Unit Testing [verification]
- Integration Testing [verification]
- Acceptance Testing [validation]
- Regression Testing [verification/validation]
- Compatibility Testing [other]
- Performance Testing [other]

CI/CD

- Continuous Integration and Continuous Delivery
- When you push a change, it will automatically deploy test to make sure it doesn’t break in existence

Person of the Day
Al Gore

- Inventor of the world wide web, HTML markup language, URL, HTTP

- Just kidding, he promoted legislation that funded and expanded the APPANET

Person of the Day
Tim Berners-Lee

- Inventor of the world wide web, HTML markup language, URL, HTTP
- In an interview, he admitted that the initial pair of slashes “//” in a web address are unnecessary
  - “seemed like a good idea at the time”

Terminologies

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/10/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Stakeholders

- Parties involved in the development of the software

Stakeholders (Examples)

- Lawyers
- Developers (you)
- Testers (IVV, performance team, security)
- Managers
- Government Agencies

- Customers
- Users
- Distinction: **Customers pay for the software. Users use the software**

Requirements

- Defines the behavior of a software
- Enumerated list of “The system shall do x”
- Often, you have to gather the requirements from your customers
- This process is called *Requirements Engineering*
  - Understand the needs of the customer to build the software that suits their needs

Functional vs NonFunctional Objectives

- Functional objectives describe what the system should do
  - The system must verify the user before showing their payroll information
  - The autonomous vehicle must reach its destination
- Nonfunctional objectives describe the general property of the system
  - The payroll information should load within 0.5 seconds
  - The autonomous vehicle should exhibit safe driving styles

“System Behavior”

- You can observe the behavior of the software developed as a whole
- When you observe the software, what type of *behavior* does it exhibit?

- We want to ensure *correctness* and *safety*

Operating Context

- The environment context in which your software is deployed in
- Could be just an app (not interacting with environments)
- Could be a cyber physical system deployed (think autonomous vehicle)

Uncertainty

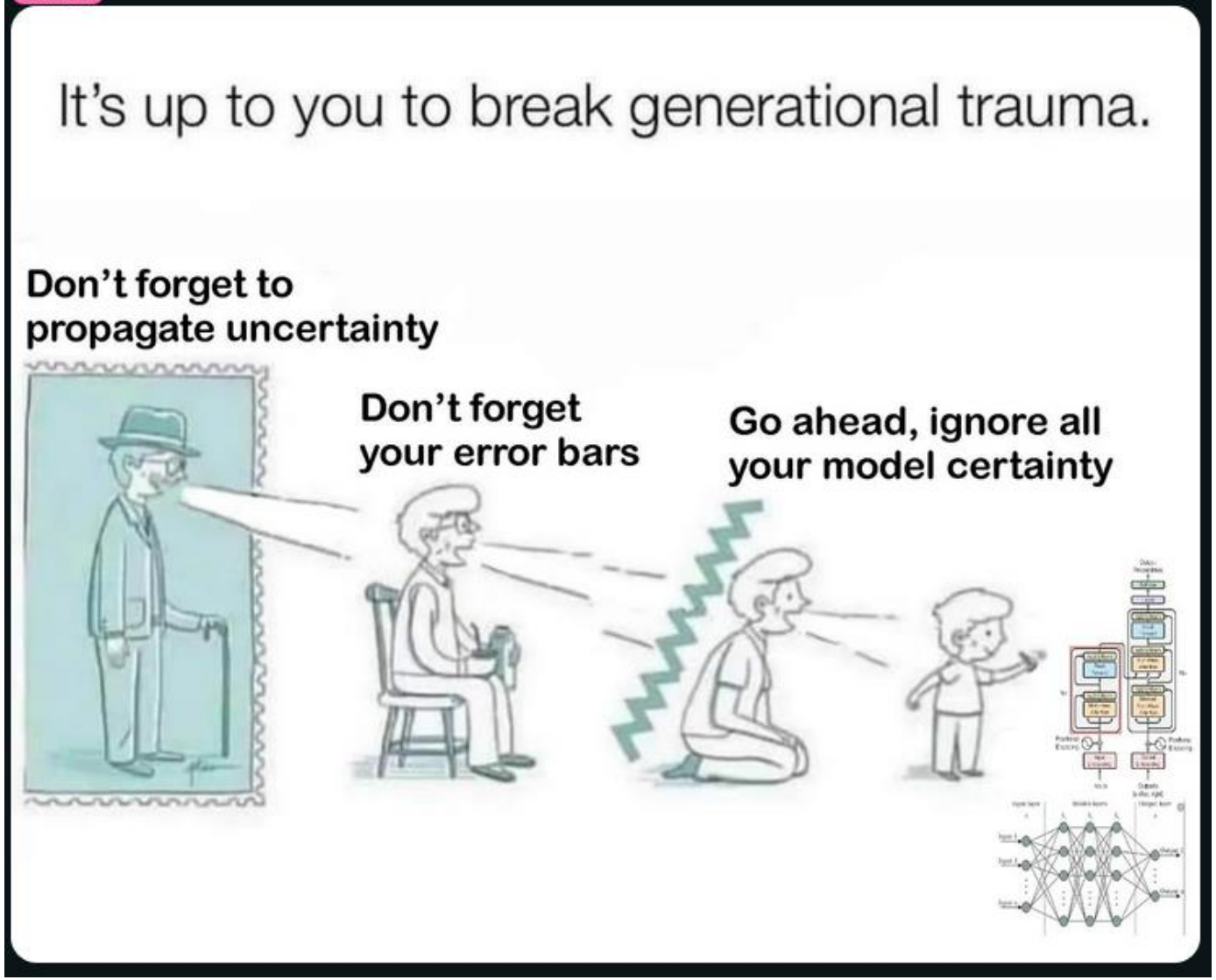
- Once you have developed the requirements of the software, operating context which you have not seen before form **uncertainty**

- Aleatoric uncertainty: Also known as stochastic uncertainty. They are unknowns that change each time we run the same experiment
  - Uncertainty in data, noise, and intrinsic randomness
- Epistemic uncertainty: Also known as systematic uncertainty. Caused by things one could known in principle but does not in practice
  - Lack of knowledge of information
  - Thus, can be fixed by gaining more information

r/mathmemes • 1 yr. ago
TobyWasBestSpiderMan

These kids probably don't even know the difference between Aleatoric and Epistemic certainty!

Statistics



Design time vs Run time

- Design time denotes the development process
  - When the software is being designed, built, and tested
- Run time denotes after the product is deployed

- You might have heard of the term run-time error

Design time vs Run time

- You would like to be able to address as much “hiccups” as you can during design time
- Uncertainty that can be addressed should be explicitly addressed

- If you can enumerate the uncertainties during design time, then you can better take steps to address it (avoid, mitigate, etc.)

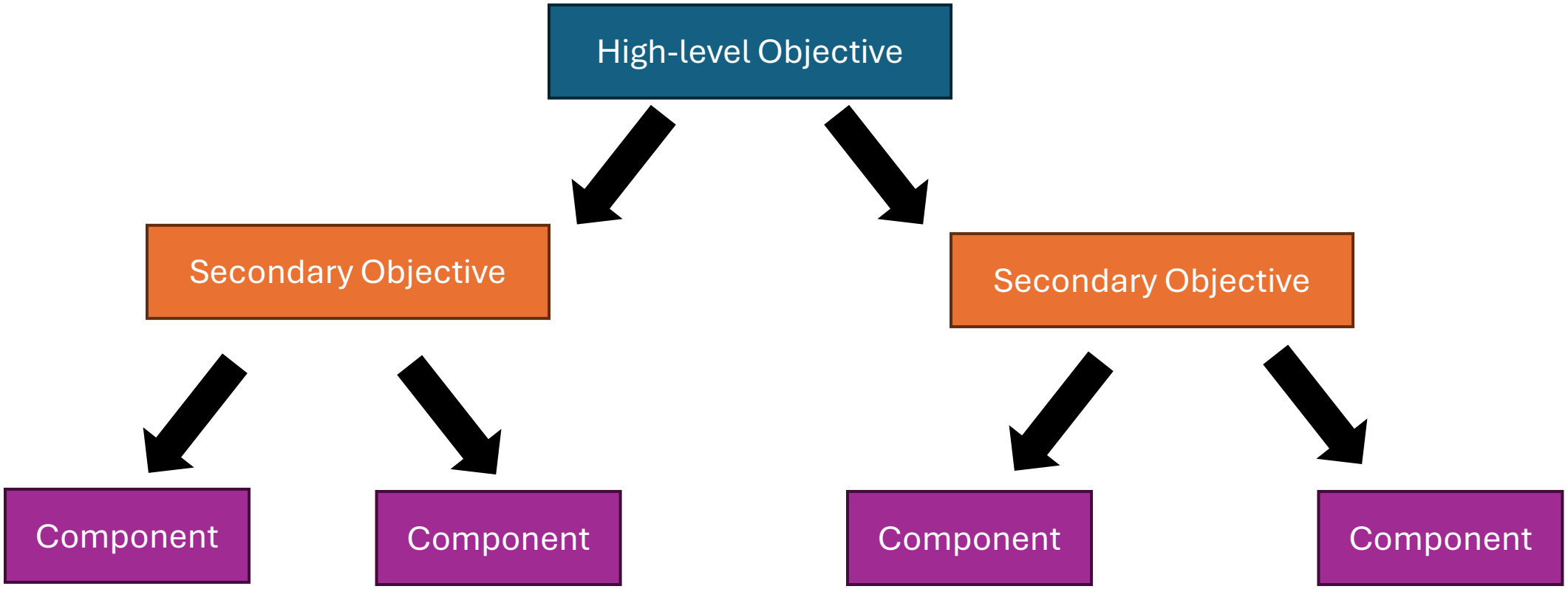
Determinism and Non-Determinism

- If I run a program 10 times, if the outcome is exactly the same all 10 times, it is *deterministic*
- If I run it 10 times, if any of the outcome is different, it is non-deterministic
  - (Undefined behaviour)

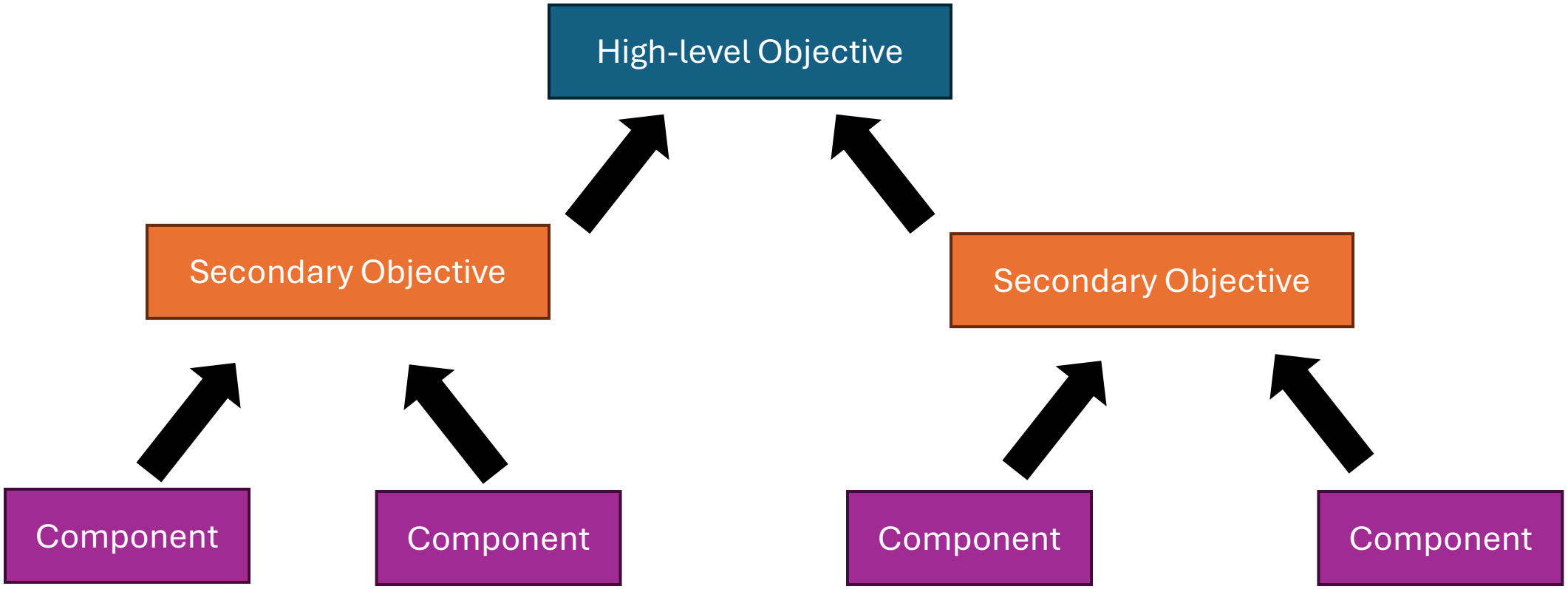
On building a system

- Top-down approach
- Bottom-up approach

Top-down


High-level Objective
Secondary Objective
Secondary Objective
Component
Component
Component
Component

Bottom-up


High-level Objective
Secondary Objective
Secondary Objective
Component
Component
Component
Component

Website example

- Top-down approaches are really good when the problem is well defined or well know
  - If the customer knows exactly what type of website they want and how it should look, function, etc.
  - Can be challenge if the top-level objective is not really known
- Bottom-up approaches are good when the problem is more nebulous
  - If the customer doesn’t really know what they want, and its your job to find out
  - Build basic structures, containers, parts of the website. Show them and change from there

Person of the day
Grace Hopper

- First to devise machine independent programming languages
  - COBOL
- Developed the first compiler
- Found the first ``bug'', and thus popularised the term

Software Development Life Cycle (SDLC)

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/12/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Introductions

- Software Development Life Cycle (SDLC)
  - Sometimes called **Software Development Process**
- Describes the **process** in which a software is developed
- In small teams, you can kind of just agree on how to code up the software
- In bigger teams (1000+), there are a lot more challenges to nonstructured teams
- Different time zones, different languages, different culturesi

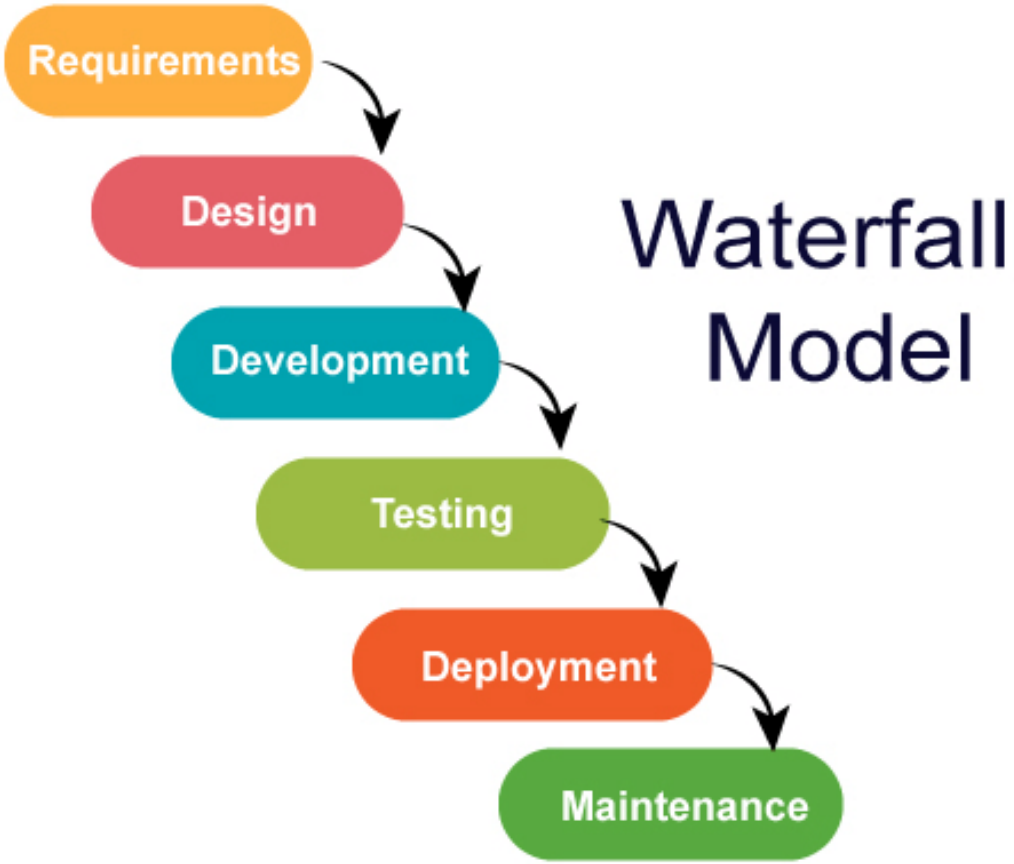
Motivation

- There is a need for structure in how teams develop the software together
- Let’s walk through the steps of creating a software system.

Life cycle

1. Requirements gathering (figure out what needs to be built)
2. Design (how will you build it?)
3. Coding (build the product!)
4. Testing (test the product!)
5. Deploying (ship the product.)
6. Maintenance (fix? Update the product if needed)

Oh, look we have a process!



https://medium.com/@chathmini96/waterfall-vs-agile-methodology-28001a9ca487

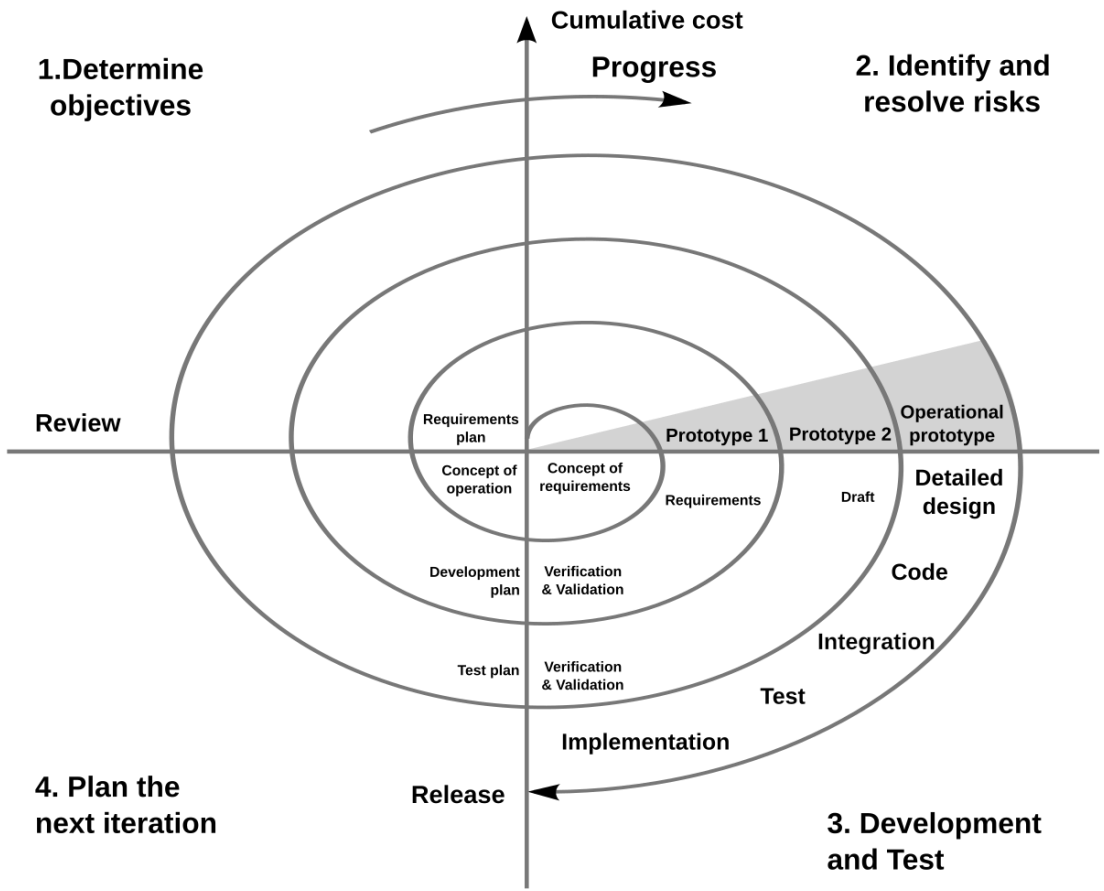
Process model

- Describes the way we can structurally develop a piece of software

Waterfall Model

- Pros:
  - Simple, concise, and details are clearly outlined
- Cons:
  - Rigid
  - Testing only comes towards the end

Spiral



Boehm, B

Agile

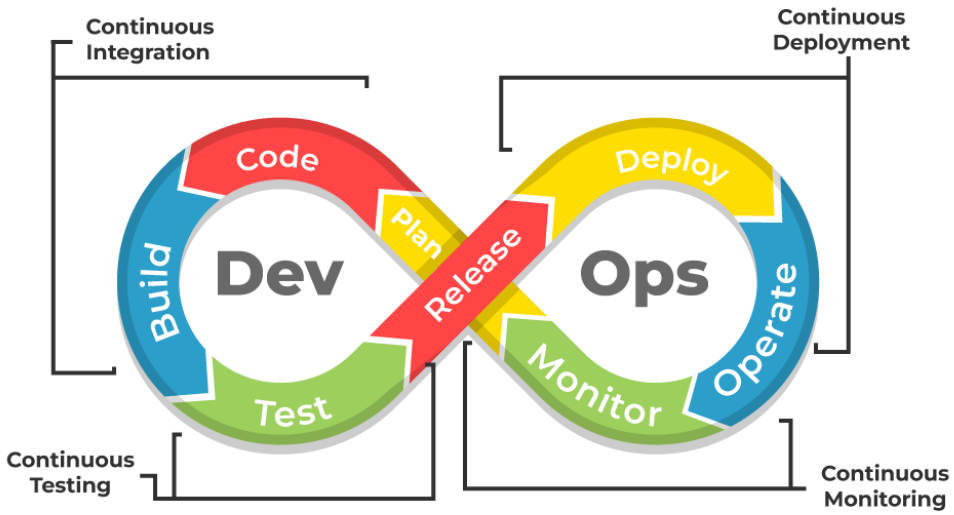
- Focuses on **flexibility, collaboration, and customer satisfaction**
- Quick iterable changes
- Focuses on developer creativity
- Often, the customer will literally have an office in the room
  - Pair Programming
  - Test Driven Development
  - Cross functional team
  - Daily Standup

Agile

- Pros
  - Fast delivery
  - Customer satisfaction
  - Changes in requirement is not that bad
- Cons
  - **Requires really good programmers and a high level of expertise**
  - Not really suitable for large projects that require planning and meticulous design
  - Difficult to estimate effort or time resources needed
  - Uncertainty and stress on developer
  - Developer burnout

DevOps

- Developments and operations
- Promotes the collaboration between development team and operations team
- Intention: increase speed to delivery



geeksforgeeks

Kanban

- Card Issue-based approach
- Outline the stages of your steps (features or tasks)
- Tackle each step one piece at a time
  - Think JIRA board or Trello board

Current model

- Mix-n-match of agile, devops, kanban, etc.
- Jira style
- “Agile but nobody really sticks to it, just get your tickets and do them” – friend from startup

Summary

- Process models intend to provide a structured approach to developing software
- You can mix and match processes
- Should figure out what works best for you and your team. There is no one size fit all answer
- Each process has pros and cons

Person of the Day
Ada Lovelace

- First ever computer programmer
- On her notes translating Charles Babbage's Analytical Engine, she added an algorithm that can compute Bernoulli numbers.
- She was the first to recognise that the machine had applications beyond pure calculation.

Technical debt and other challenges

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/17/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Technical Debt

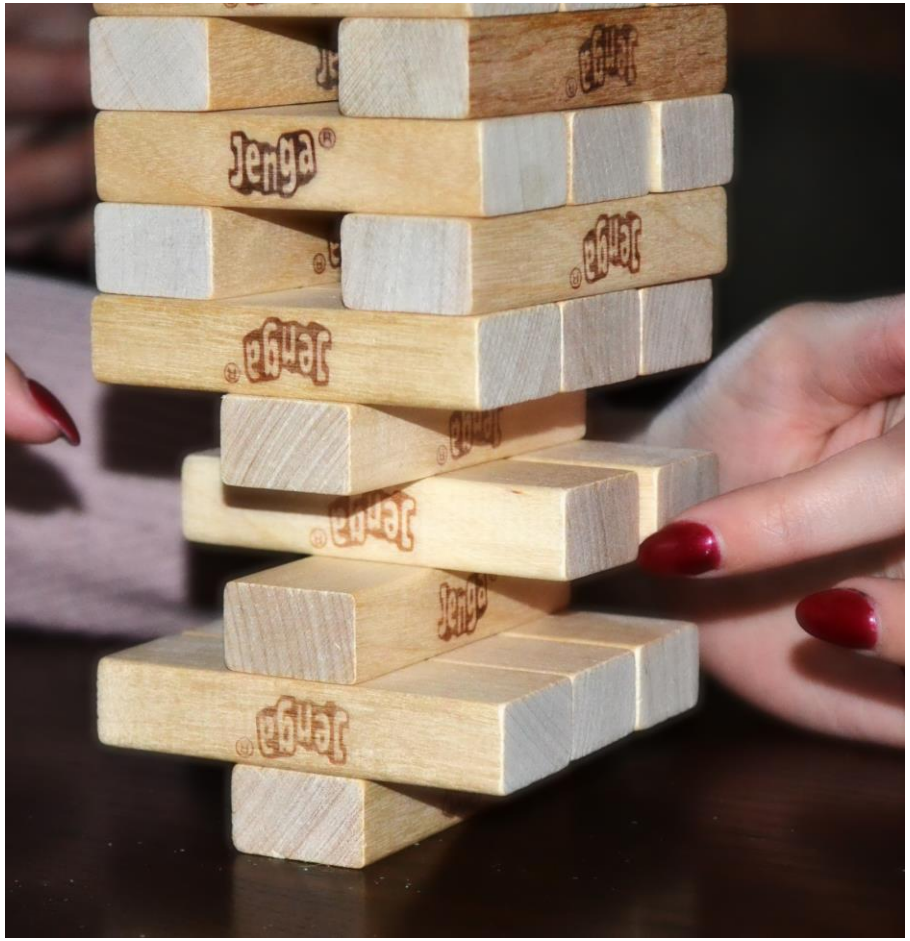
- Technical debt refers to the implied future cost resulting from immediate solutions
- “Punting the problem for future me”
- Example:
  - Hard coding
  - Outdated libraries or deprecated dependencies
  - Lack of documentation
  - Lack of testing
  - Deferred updates
  - TODOs, FIXMEs

Causes

- Business and management pressure
  - Prototyping
- Knowledge gap
- Development process challenges
  - Lack of requirements (incomplete)
  - Conflicting requirements
- Changing requirements
- Security****

Problems that arises

- Maintenance after deployment
- Adding new features (think of a Jenga tower)
- New developers or switching teams
- User experiences
- Speed



Legacy code

- How many of you have worked directly with an “ancient” programming language
  - Fortran
  - Lisp
  - COBOL
  - BASIC
  - Pascal
- Easier to think of them as input output black-box transformation and write a wrapper for them

Complexity and difficulties

- Essential complexity
  - Inherent to the problem itself, and thus cannot be truly eliminated
- Accidental complexity
  - Introduced accidentally as we are trying to solve the underlying problem
- Fred Brooks article: No Silver Bullet - Essence and Accident in Software Engineering
  - One of the most well known and cited articles

Person of the Day
Fred Brooks

- Landmark contributions to computer architecture, operating systems, and software engineering
- Invented the interrupt signal
- Started the Computer Science department at University of North Carolina at Chapel Hill

Testing Completeness

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

03/07/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Testing is exhaustive

- It is mathematically impossible to test a program completely
- Testing should focus on edge cases and reach across the search space as far and deep as possible
- But that also doesn’t mean that testing a function is necessary complete when you test the immediate functionality
  - Things in the class may interact with each other and affect other things

Testing Extremes

- On one end, you can test as many inputs as you can think of
- Testing the same ``type'' of input also does not help
  - Tests should test different things

- On the other hand, testing just the immediate functionality is also incomplete
- Good testing is about balancing the two ends, picking the most effective test cases.

TDD

- TDD involves building the test **cases** first, then write the code

- This means you should build test cases that test all the expected functionality of the functions, and in addition all the edge cases and other issues that can arise.

- Test case generation: leetcode?

Some common mistakes

- I will try to use a theme of testing a stand vector class (or your own implementation of it), but the general concepts will apply to a lot of classes

Testing if the code breaks only

- Suppose you are testing the function that pushes back an item

- Vector.pushback(42)
- check(Vector.size == 1)

- What else should we do?
  - Check if the item itself is actually 42
  - Type checking?
  - Add more than one item, add a lot of items, make sure they all match expectation

Testing only the intended functionality

- Suppose we want to test the vector.size() function

- Vector.init()
- Vector.pushback(10, 20, 42)
- Check(vector.size() == 3)

- What might be some issues with this testing?
- What else to test for?
  - Check for size correctness after add, removes, clears, appends, extends, etc.
  - Check for size of the entire vector memory allocation (sizeof)

Not testing the actual values

- Suppose we are testing a remove function
- Vector = {1,2,3,4,5}
- Vector.remove(3)
- Vector.remove(4)
- Check(vector.size == 3)?

- Issue?
- This can also apply to add, remove, etc. Any time you modify something, check if the modification is as expected

Checking related functions

- Suppose we are testing the clear() function of a vector

- Vector.clear()
- Check if size == 0?

- Issue: What else should you do?
  - Try to add again! What if your clear function breaks the fundamental container for the class?
  - Try to clear if there are no items in the container

Not testing incorrect parameters

- Lets consider pushing items into a vector<int>
- Vector.push_back(1)
- Vector.push_back(2)
- Vector.push_back(‘3’)

- Matrix shapes, push back an incorrectly shaped row?

Edge cases testing

- This one might be a little challenging
  - Requires a lot of expertise in the problem and some thoughts
  - Think of it as a puzzle, and your job is to break the function
- Check for common issues:
  - Out of bounds
  - Input with “0” or null or nullptr
- If your problem implements an existing problem, then you can use a lot of existing inputs to test your program
  - Example: knapsack problem

Testing for input const-ness

- If your function do not modify the input (and should not). It is good practice to makes sure that your input copy is the same after you call the function
- Make a deep copy of the original object at the start of the testing, and make sure they are identical after the test cases

Testing only one instance of an input

- Suppose we want to test a random generator
- Num = randint(0, 100)
- Check(0 <= num <= 100)

- Num2 = randint(0, 100)
- Check(num != num2)

- Set a different seed and test if the first number generated is the same?
- Issues?

Change the way you think about testing

- Computers are fast now, use their computation to your advantage.
- Use many inputs (random insertion and deletion)
- Randomness testing example
- Generate 1 million numbers
- Test the average
- Split the values into “buckets”, then test if the buckets fall within the expected variance.

Summary

- Don’t test only if the code throws errors
- Don’t test only if the intended functionality is working
- Make sure you check related functions
- Will your test case pass if you use the “right input” only
- Check if the function unexpectedly modifies input
- Use a longer number of input when testing

Person of the day
Margaret Hamilton

- Coined the term “software engineering”
- Developed the on-board flight software for NASA’s Apollo program
- Director of the Software Engineering Division of the MIT Instrumentation Laboratory

Turing Completeness

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

03/17/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Turing completeness

- For a programming language to be Turing complete:
  - It must be able to do everything that a Turing machine can do

Turing Machines

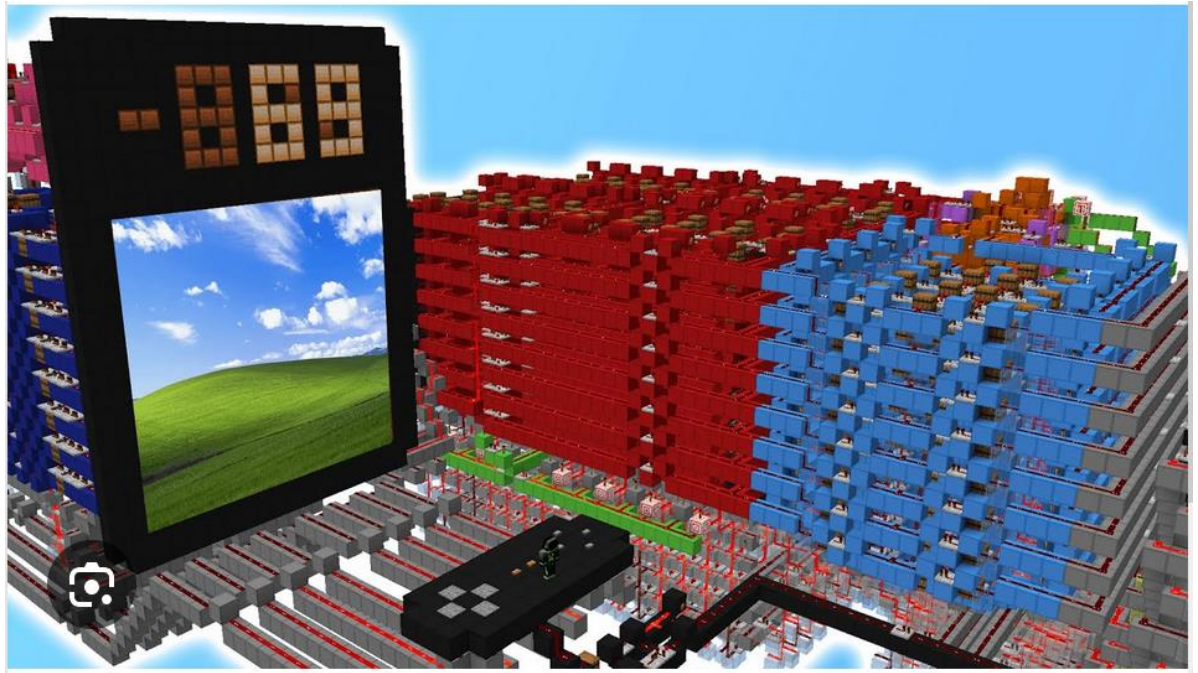
- An infinitely long tape
- A Read/Write Head that can store 0 or 1 on any part of that tape
- If your program can do that, it is a Turing Machine
- It is powerful enough to compute anything that can be computed

Turing complete conditions

- 1) Conditional Branching
  - If-then-else statements
- 2) Ability to go to a different part of the tape
- 3) Must be able to have arbitrary amount of memory

Turing Equivalence

- Turing equivalence is a system that is exactly as powerful as a Turing machine
- Given two computers P and Q
  - P can simulate Q
  - Q can simulate P



I Made a Working Computer with just Redstone!

Visit ›

Turing completeness

- Most programming languages are turing complete
- BASIC
- PASCAL
- C++
- Python
- Minecraft
- Magic the gathering

Magic: The Gathering Is Turing Complete

Alex Churchill
Independent Researcher, UK
http://www.myhomepage.edu
alex.churchill@cantab.net

Stella Biderman
LucyLabys, Georgia Institute of Technology, Atlanta, GA, USA
Booz Allen Hamilton, Atlanta, USA
http://www.stellabiderman.com
stellabiderman@gatech.edu

Austin Herrick
Penn Wharton Budget Model, University of Pennsylvania, Philadelphia, PA, USA
aherrick@wharton.upenn.edu

Implications

- For any type of program written in a given language, you can write an equivalent program in a different language.
- So why do we care what type of language we write a program in?

Person of the Day
Alan Turing

- Father of theoretical computer science
- Concepts of algorithms
- Help with War efforts on German Ciphers during WWII

References

- https://evinsellin.medium.com/what-exactly-is-turing-completeness-a08cc36b26e2
- https://www.youtube.com/watch?v=RPQD7-AOjMI
- https://cs.stackexchange.com/questions/134273/explain-the-difference-between-turing-complete-and-turing-equivalence

“Simplicity is the prerequisite for reliability”

- Edsger W. Dijkstra

Edsger W. Dijkstra and Structured Programming

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

03/19/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Edsger Dijkstra

EWD1036-0

On the cruelty of really teaching computing science

The second part of this talk pursues some of the scientific and educational consequences of the assumption that computers represent a radical novelty. In order to give this assumption clear contents, we have to be much more precise as to what we mean in this context by the adjective "radical". We shall do so in the first part of this talk, in which we shall furthermore supply evidence in support of our assumption.

The usual way in which we plan today for tomorrow is in yesterday's vocabulary. We do so, be-

Edsger Dijkstra

- Dijkstra’s shortest path algorithm
- Semaphore
- Reverse Polish Notation to reduce computer memory access when evaluating expressions
- Banker’s algorithm
- System self-stabilization
- Separations of concerns
- …

Biggest Contribution: Structured Programming

Spaghetti code

TI-84

- Graphing calculator
- Supports programming using a BASIC-like language
- Sequential programming instructions
- Rather than functions, you would jump to a different segment of code, using:
  - Goto statements
  - Labels

Quadratic equation program

- Input -> A, B, C
- Discriminant_sqrt = 0
- Goto **dis_label**
- Label **combine_label**
- X_pos = (-b + discriminant) / 2a
- X_neg = (-b – discriminant) / 2a
- Print(Solutions: X_pos, X_neg)

- Label **dis_label:**
- Discriminant_sqrt = sqrt(b^2 – 4ac)
- Goto **combine_label**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Scalability

- Now, imagine hundred thousands of lines in the same file, with many goto, labels, and jump commands.

Spaghetti code

Issues

A Case Study of Toyota Unintended Acceleration and Software Safety

Prof. Phil Koopman

Why does it matter?

- If all languages are turing complete, then why does it matter?

Structured Programming

- Promotes the use of structured control flows
  - As opposed to unstructured jumps to various sections of the program using Goto statements
- Elimination of global variables
- Loops and functions
- Hierarchical decomposing of the program

Structured vs Unstructured Programming

- Unstructured programming can also achieve the same result as structured programming
- But are much more prone to errors, development issues, lack of clarity, etc.

- It also does not mean you cannot write (basically) unstructured code in a structured programming language

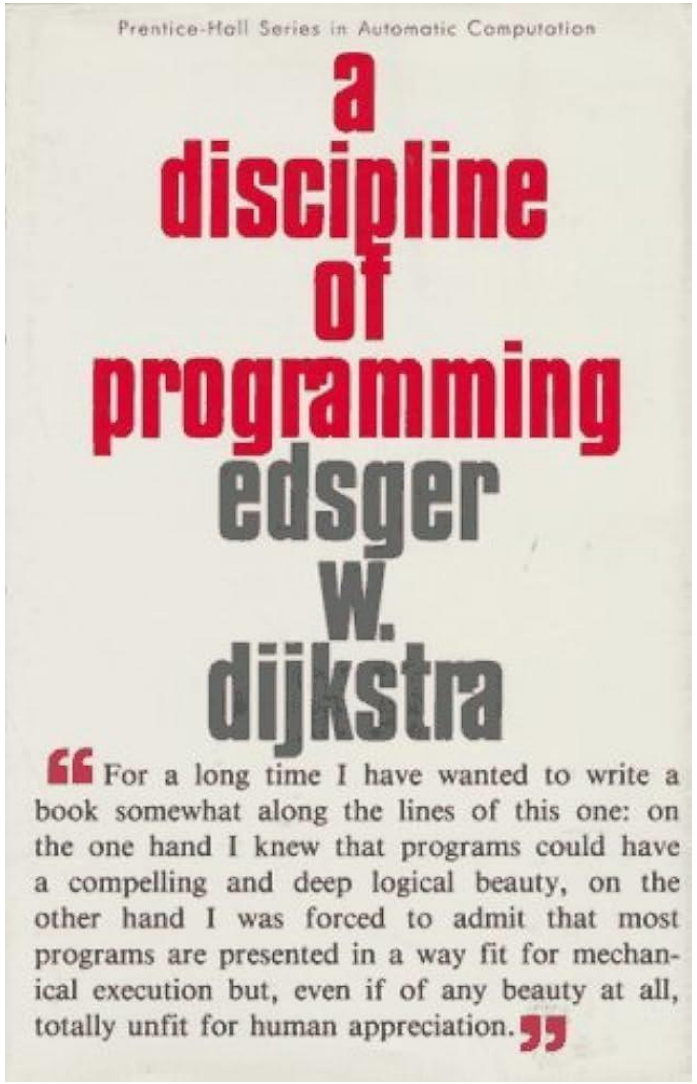
Examples

- Procedural Programming
  - E.g., functional programming
- Object-oriented Programming
- Model-based Programming
  - E.g., databases or query languages

Advantages

- Improved readability of code, and it forces the developer and reader to follow the logical structure of the code
- More efficient
- Easier to understand and modify
- Reusability

A discipline of programming



Person of the day
Edsger Dijkstra

- Famous for Dijkstra shortest path
- Structured Programming
  - No goto statements
  - Modern programming practices
- Semaphores
- Never used a computer (What??)
- Favourite composer is Mozart
- “Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.”

Hilbert's 10th Problem, Undecidability, & The Halting Problem

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

03/24/2025

Kira Chan

https://cse.msu.edu/~chanken1/

David Hilbert

- Brilliant mathematician in the early 1900s
- During his speech to the International Congress of Mathematicians in 1900, he presented 23 unsolved problems
- His 10^{th} problem:
  - For any given Diophantine equation, can we *decide* whether the equation has a solution?
- This problem remained unsolved for 70 years, until 1970

Decidability

- In computer theory, an undecidable problem is one which it is impossible to construct an algorithm that can always lead to a yes or no answer
- Hilbert called it ***Entscheidungsproblem*** (decision problem)

- Example: Halting problem

Halting Problem

- Can you write a computer program *P* that:
  - For a given input, a computer program *i,* determine whether *i* will:
    - Halt (exit execution)
    - Run indefinitely

- To prove this, you must show that
  - Decide on ALL existing programs, even ones that have not existed
  - OR use prove by contradiction (show an instance of the problem that cannot logically exist)

Alan Turing's Paper in 1936

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

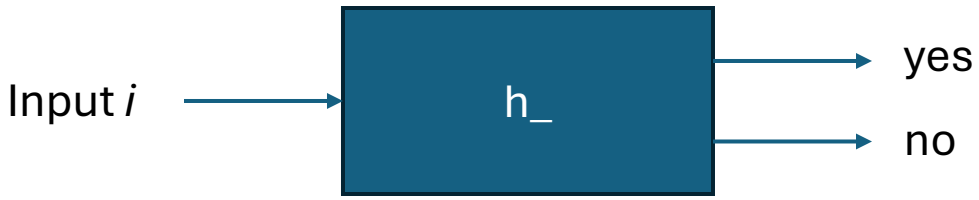
By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

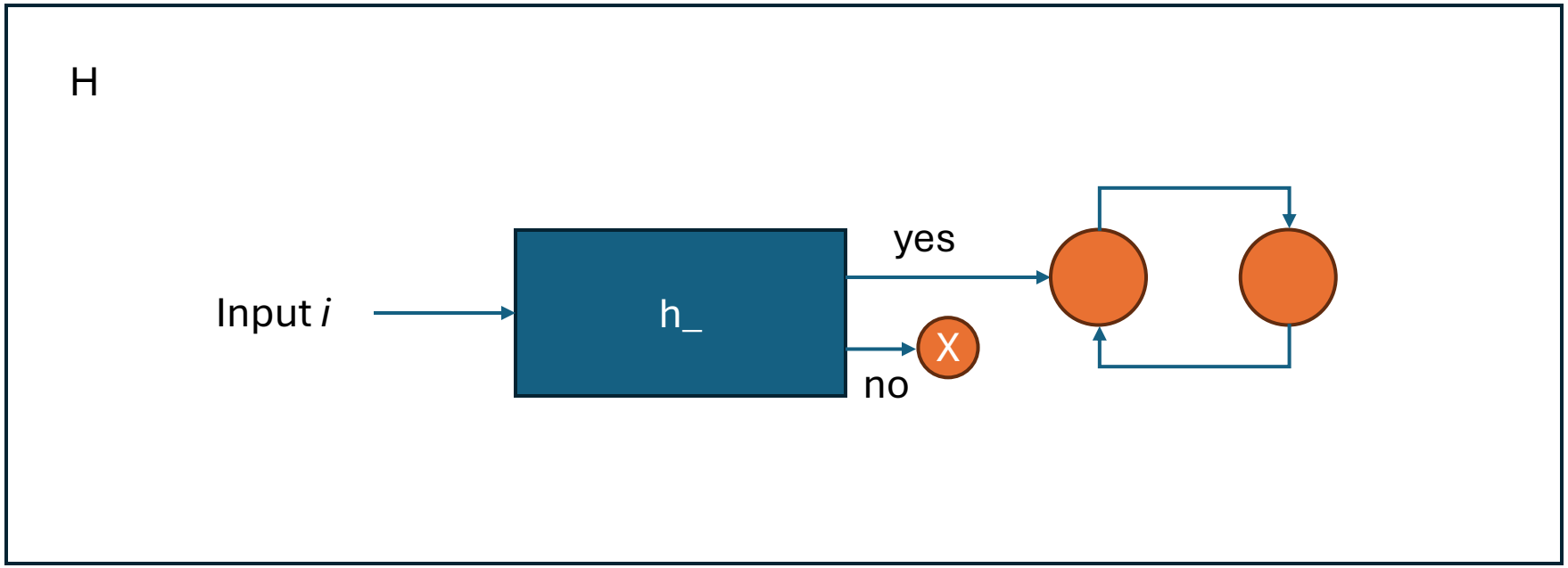
Halting Problem

- Suppose there exist a program *h_* that can magically decide whether a program halts or not



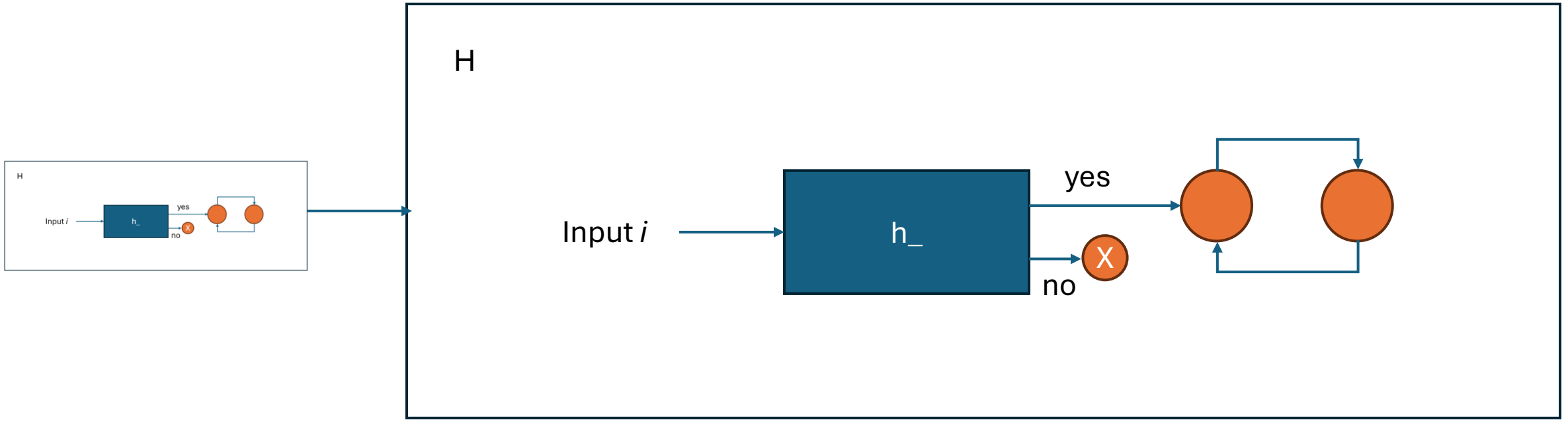
Halting Problem

- Let's construct a new program *H*, based on *h_* as follows
  - If *h_* says that the problem halts, then run forever
  - Else, halt



Contradiction

- What happens when we feed the program an instance of itself?
  - If *H* halts, then *h_* returns yes; but it runs forever
  - If *H* does not halt, then *h_* returns no; but it does halt



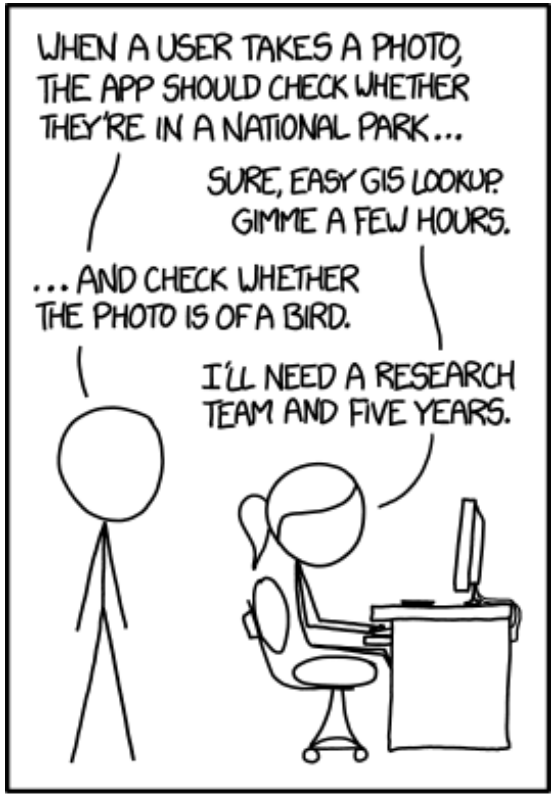
Halting Problem - Results

- The halting problem showed an instance of a problem that *cannot be solved by computers*
- This problem showed that some functions are mathematically definable, but not computable

Implications

- Actually equivalent (can be reduced) to Hilbert's 10^{th} problem
  - Proved by Matiyasevich
- Previously, we discussed that exhaustive testing is mathematically infeasible
- Microsoft rolls out a new update, and they want to make sure that it does not break backwards compatibility at all. This is actually just the halting problem
- Practically, it is to be able to tell your boss that "what you are asking for is mathematically impossible, and I can prove it"
  - Write a program that can automatically debug other programs

XKCD - 1425

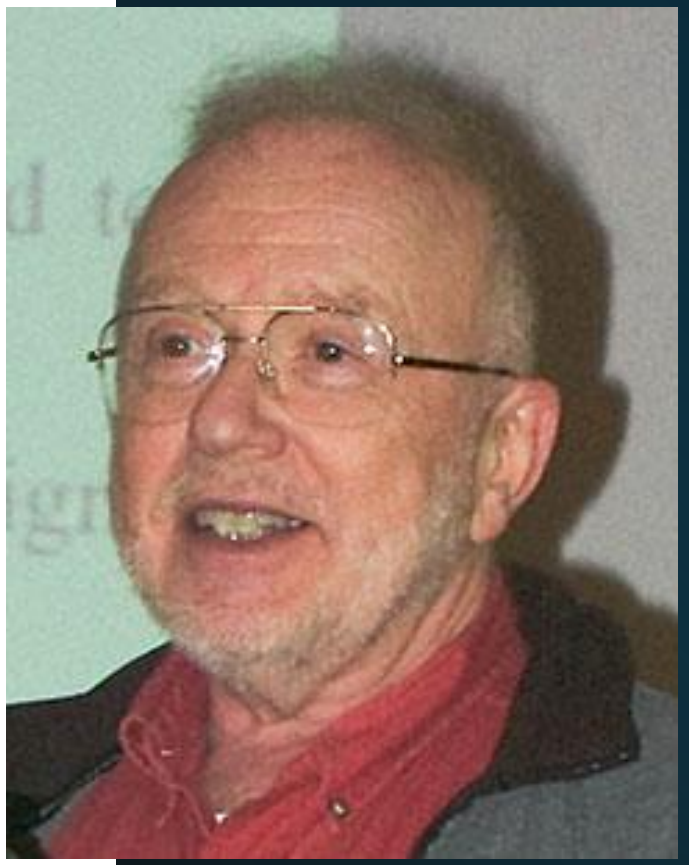

WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Summary

- One of the most important results in our field
- Halting problem showed that there exist problems in which computers cannot compute
- Some what defines the limits of what programming can do

Person of the day
David Parnas

- Another prominent figure in software engineering
- Information hiding
- Modularity, component-based reuse
- Separation of concerns
- Software of defence planes
- Abstraction
- Scalability
- Design patterns are really based on things that Parnas came up with



Design Patterns

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

03/25/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Definition

- General and reusable solution to a commonly occurring problem in software design
- It is a design pattern, not a snippet of code
- Pair<Problem, Solution>

- Introduced by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (a.k.a., the group of four) in 1994
- Introduced the original 22 class design patterns in their book
- The book is written by analysing code and extracting patterns from them, not designing the patterns to be implemented

Analogy

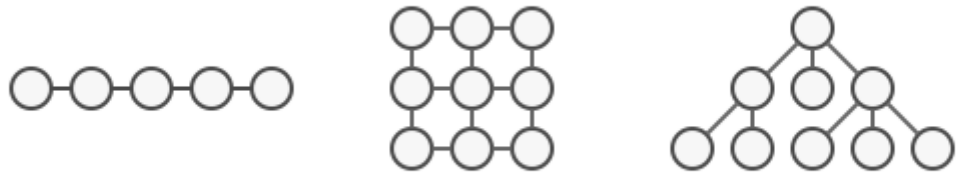
- They are similar to a blueprint that you can customize to solve a particular problem
- Can be categorized based on complexity, levels of abstraction, etc.

Three classification of patterns

- Creational patterns
  - Designed for class instantiation
  - Examples: Abstract factory, builder, singleton, etc.
- Structural patterns
  - Designed for class composition and package structure
  - Examples: Adapter, Bridge, Façade, Flyweight, etc.
- Behavioral patterns
  - Design for communications between classes
  - Examples: Interpreter, mediator, observer, etc.

Example: Iterator pattern

- Turns out traversing a data is a commonly reoccurring theme
- Data can be organised in many ways, and even traversed in many ways in the same container



Various types of collections.

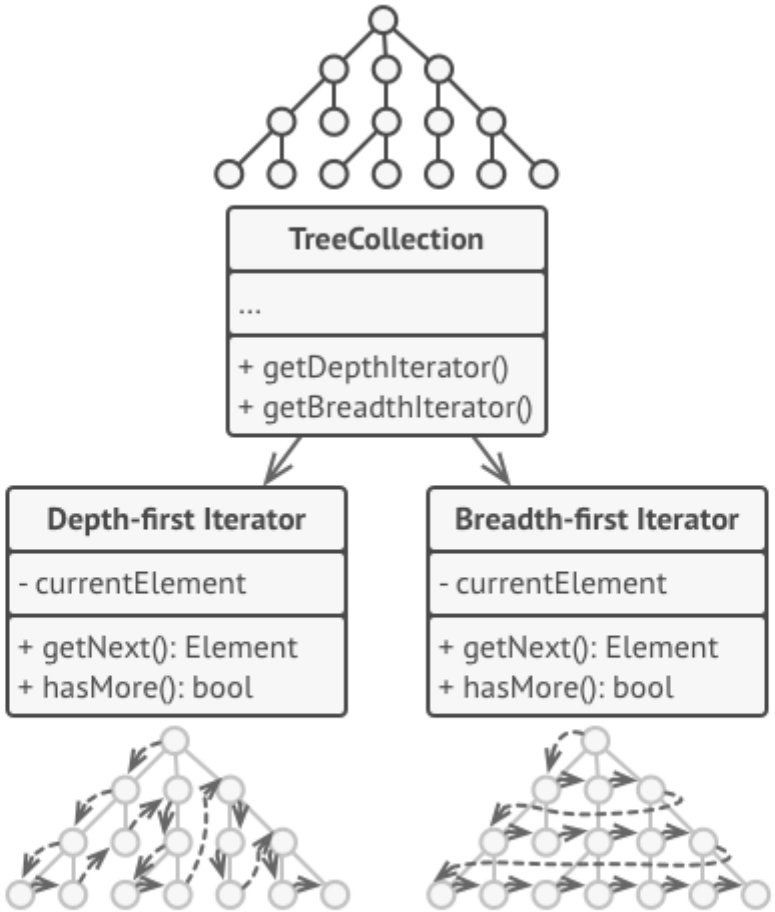
https://refactoring.guru/design-patterns/iterator

Iterator Pattern

- Generate an iterator that describes how an object shall be traversed

- Implement a getNext() function that describes how to step to the next item
- Implement a hasMore() function to check whether there are items still to be traversed

Visualisation



Iterators implement various traversal algorithms. Several iterator objects can traverse the same collection at the same time.

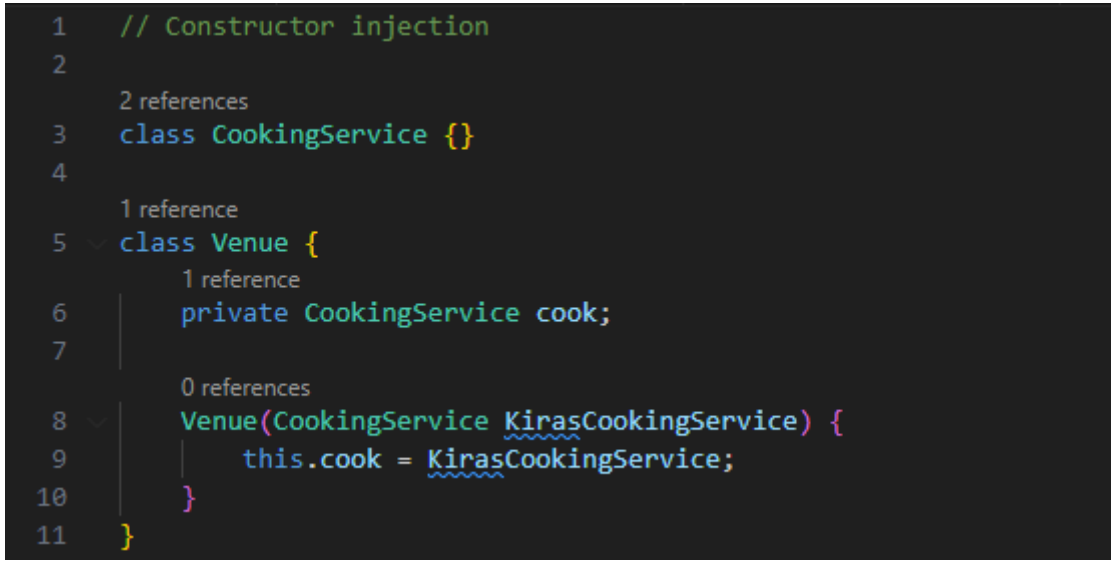
Discussions

- Design patterns are not intended to force a way to implement something
- They are merely commonly used patterns to address some problem
- It will start ``clicking'' when you look at a snippet of code and get that feeling of "oh, I have seen this before"

Example: Dependency Injection

- Dependency injection: pass the classes that your class depends on as interfaces rather than creating a separate instance of them
- Removes dependencies on other classes

- Example: Venue hosting for food
- Commonly used in application
  or web development



```
// Constructor injection

2 references
class CookingService {}

1 reference
class Venue {
    1 reference
    private CookingService cook;

    0 references
    Venue(CookingService KirasCookingService) {
        this.cook = KirasCookingService;
    }
}
```

Example: Curiously reoccurring template pattern (CRTP)

- Where a class has a base class which is a template specialization for the class itself
- More generally known as “F-bound polymorphism”
- Allows for static polymorphism (decides which method to execute during compile time)
- Also gives the template class the ability to be a base class for its specialisations

```
template <class T>
class X{...};
class A : public X<A> {...};
```

- https://stackoverflow.com/questions/4173254/what-is-the-curiously-recurring-template-pattern-crtp
- https://www.fluentcpp.com/2017/05/12/curiously-recurring-template-pattern/

CRTP

- From the perspective of the base object, the derived object is itself, but downcasted
- Therefore, the base class can access the derived class by static_casting itself into the derived class

```
template <typename T>
class Base
{
public:
    void doSomething()
    {
        T& derived = static_cast<T&>(*this);
        use derived...
    }
};
```

- Example on these slides are from: https://www.fluentcpp.com/2017/05/12/curiously-recurring-template-pattern/

Usefulness

- Here's a class that has an attribute *value* and 3 different functions
  - Scale
  - Square
  - SetToOpposite
- Supposed I have another class with a *value* that want the same functions
  - Should we just copy over the functions?

```
class Sensitivity
{
public:
    double getValue() const;
    void setValue(double value);

    void scale(double multiplicator)
    {
        setValue(getValue() * multiplicator);
    }
    void square()
    {
        setValue(getValue() * getValue());
    }
    void setToOpposite()
    {
        scale(-1);
    };

    // rest of the sensitivity's rich interface...
};
```

CRTP approach

- Pull out the functions into a separate Base class
- Have the Derived class inherent from it
- Now other classes can take the same approach!
- And we can add more functionality generically

```
template <typename T>
struct NumericalFunctions
{
    void scale(double multiplicator);
    void square();
    void setToOpposite();
};
```

```
class Sensitivity : public NumericalFunctions<Sensitivity>
{
public:
    double getValue() const;
    void setValue(double value);
    // rest of the sensitivity's rich interface...
};
```

Implementation of the Base class

```
template <typename T>
struct NumericalFunctions
{
    void scale(double multiplicator)
    {
        T& underlying = static_cast<T&>(*this);
        underlying.setValue(underlying.getValue() * multiplicator);
    }
    void square()
    {
        T& underlying = static_cast<T&>(*this);
        underlying.setValue(underlying.getValue() *
underlying.getValue());
    }
    void setToOpposite()
    {
        scale(-1);
    };
};
```

Example use case

```cpp
template <typename T>
struct Base {
  void foo() {
    (static_cast<T*>(this))->foo();
  }
};

struct Derived : public Base<Derived> {
  void foo() {
    cout << "derived foo" << endl;
  }
};

struct AnotherDerived : public Base<AnotherDerived> {
  void foo() {
    cout << "AnotherDerived foo" << endl;
  }
};

template<typename T>
void ProcessFoo(Base<T>* b) {
  b->foo();
}


int main()
{
    Derived d1;
    AnotherDerived d2;
    ProcessFoo(&d1);
    ProcessFoo(&d2);
    return 0;
}
```

Output:

```
derived foo
AnotherDerived foo
```

Patterns become standardised eventually if used enough

- C++23 introduces “deducing this” that allows you to access the derived class from the base class
- Iterators are basically the standard way to access items now

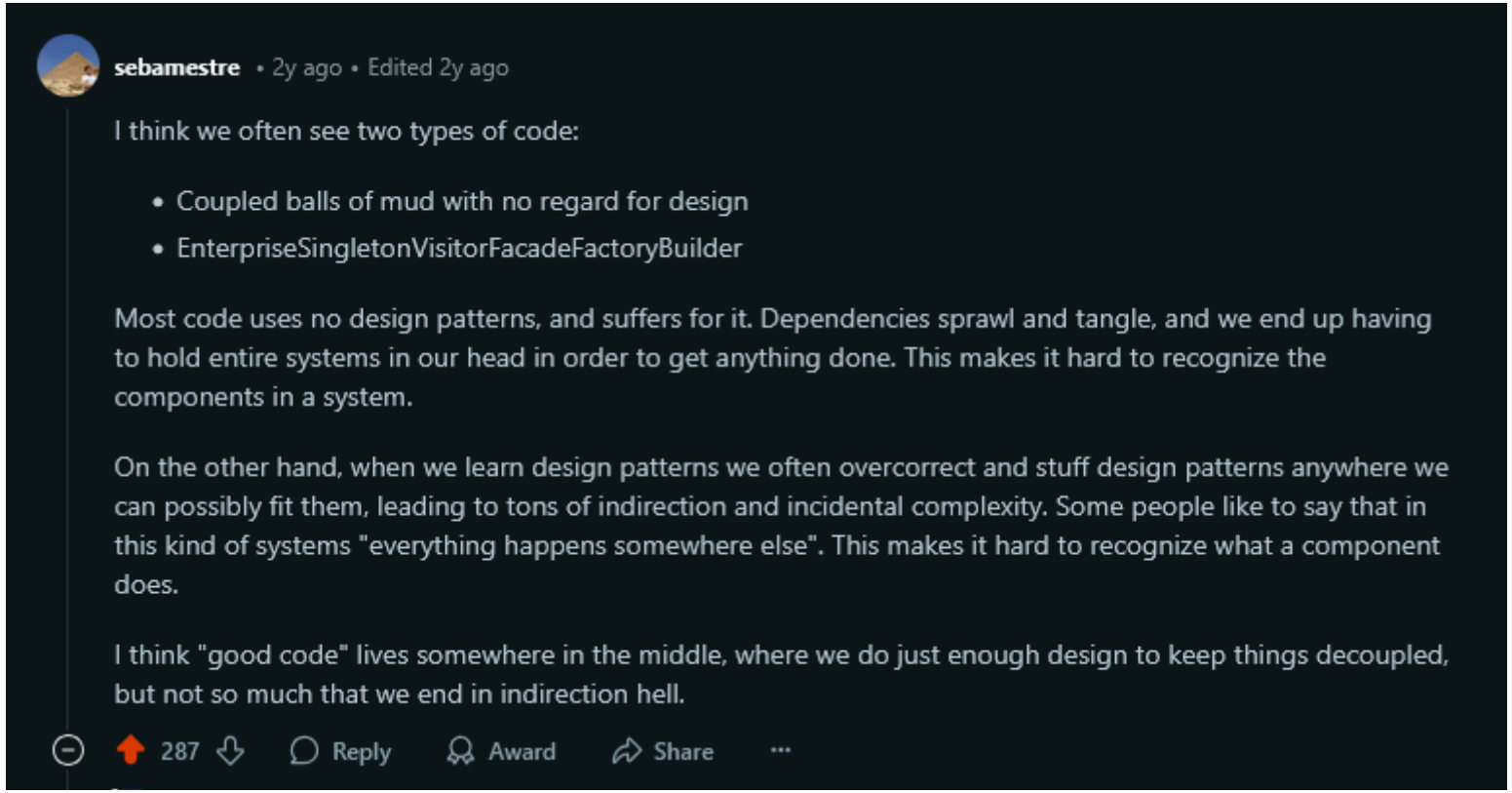
Criticism

- Can lead to inefficient solutions
- Introduces complexity
- Lead to anti-patterns
  - Commonly-used process or pattern that has more consequences than good effects

Summary

- They really allow for developers to talk about a problem and prevent re-inventing a (poorly designed) wheel
- Typically, design patterns can emerge from good coding without explicitly trying to incorporate them
- If you are forcing code to fit based on a design pattern, you are probably doing it wrong
  - They should occur naturally
- The C++ standard library actually uses and incorporates them profusely

Perfect discussion post found on reddit



sebamestre • 2y ago • Edited 2y ago

I think we often see two types of code:

- Coupled balls of mud with no regard for design
- EnterpriseSingletonVisitorFacadeFactoryBuilder

Most code uses no design patterns, and suffers for it. Dependencies sprawl and tangle, and we end up having to hold entire systems in our head in order to get anything done. This makes it hard to recognize the components in a system.

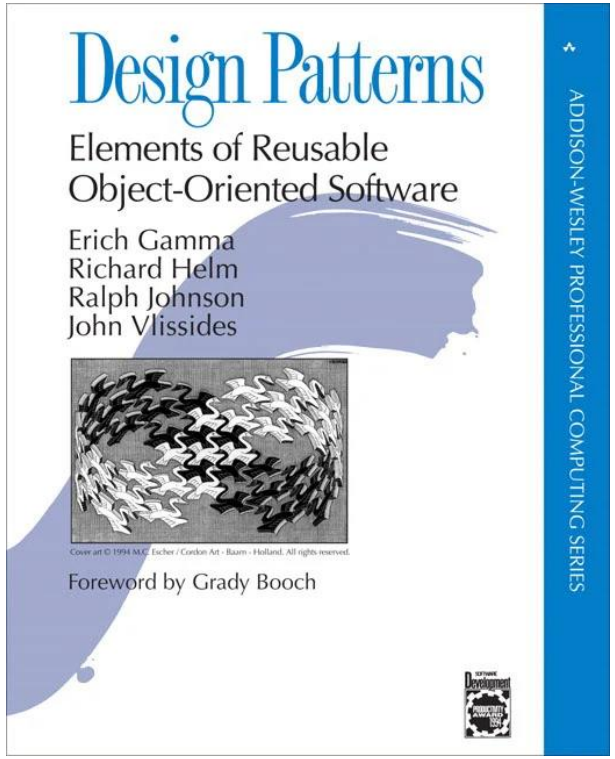
On the other hand, when we learn design patterns we often overcorrect and stuff design patterns anywhere we can possibly fit them, leading to tons of indirection and incidental complexity. Some people like to say that in this kind of systems "everything happens somewhere else". This makes it hard to recognize what a component does.

I think "good code" lives somewhere in the middle, where we do just enough design to keep things decoupled, but not so much that we end in indirection hell.

287 Reply Award Share ...

Persons of the day
Group of four

- Commonly referred to as the Gang of Four (GOF)
  - The authors do not like that name
- Gamma, Helm, Johnson, Vlissi
- Known for their software engineering book on
  - Includes examples in C++



std::move()

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

04/02/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Move semantics vs value semantics

- When you use an assignment operator, you *copy* over the values
- This is called **value semantics**
- int *b* = 20
- int *a* = b
- // both *a* and *b* are set to 20, each with their own copy

- What if you don't need *b* anymore?
  - If *b* is a really large object, then we just wasted time copying over the resource

Std::move()

- Std::move() allows you to take ownership of an object
- It effectively sets an L-value into an X-value
  - Recall L-value is one with a memory address or has a name
  - X-value is one that is “eXpiring”, and thus will disappear when out of scope
- Basically, same as:

  (Object&&) m_obj;
  - This is called an r-value reference

C: > msys64 > ucrt64 > include > c++ > 14.2.0 > bits > C move.h > {} std > move<_Tp>(_Tp &&)

```
117
118   /**
119    *  @brief  Convert a value to an rvalue.
120    *  @param  __t  A thing of arbitrary type.
121    *  @return The parameter cast to an rvalue-reference to allow moving it.
122    */
123   template<typename _Tp>
124     _GLIBCXX_NODISCARD
125     constexpr typename std::remove_reference<_Tp>::type&&
126     move(_Tp&& __t) noexcept
127     { return static_cast<typename std::remove_reference<_Tp>::type&&>(__t); }
128
```

Move constructor

```
Object(Object&& other) noexcept
{
    // take other.size if exist
    m_data = other.m_data; // yoink their data
    Other.m_data = nullptr; // set their data to be nullptr, otherwise?
}
```

Example use case

- “Swap value of var A and var B”

- Void swap(a, b)
  - temp = a
  - a = b
  - b = temp

- Void swap_move(a, b)
  - temp = std::move(a)
  - a = std::move(b)
  - b = std::move(temp)

Move constructor and assignment operator

- We learned how to do the constructor
  - This is all sunny for construction of an object
  - Object n_obj = std::move(o_obj)
  - Is the same as: Object n_obj(std::move(o_obj))
- But what happens when we want to just assign?
  - Only gets call if we assign a variable into an *existing* variable
    - Object n_obj = new Object()
    - n_obj = std::move(o_obj)

Need to define move assignment operator

```
//again, take in a temporary object
Object& operator=(Object&& other) noexcept {
    // Check if it is the same object
    if (this != &other) {
        //What happened to my old data? (Memory leak)
        // Need to make sure we delete the old data FIRST
        delete m_data;

        // same as move constructor
        m_data = other.m_data; // yoink their data
        Other.m_data = nullptr; // set their data to be nullptr
        }
    Return *this;
}
```

When not to use move

- Never return std::move()
  - Prevents compilers from doing Return Value Optimization (RVO), a form of **copy elision**
  - RVO is mandatory after C++17 even if you turn optimisation level to 0
  - Optimisation-level 0 doesn’t actually turn off all optimisations, just most
  - Pay a runtime overhead cost

Std::move() doesn’t actually move

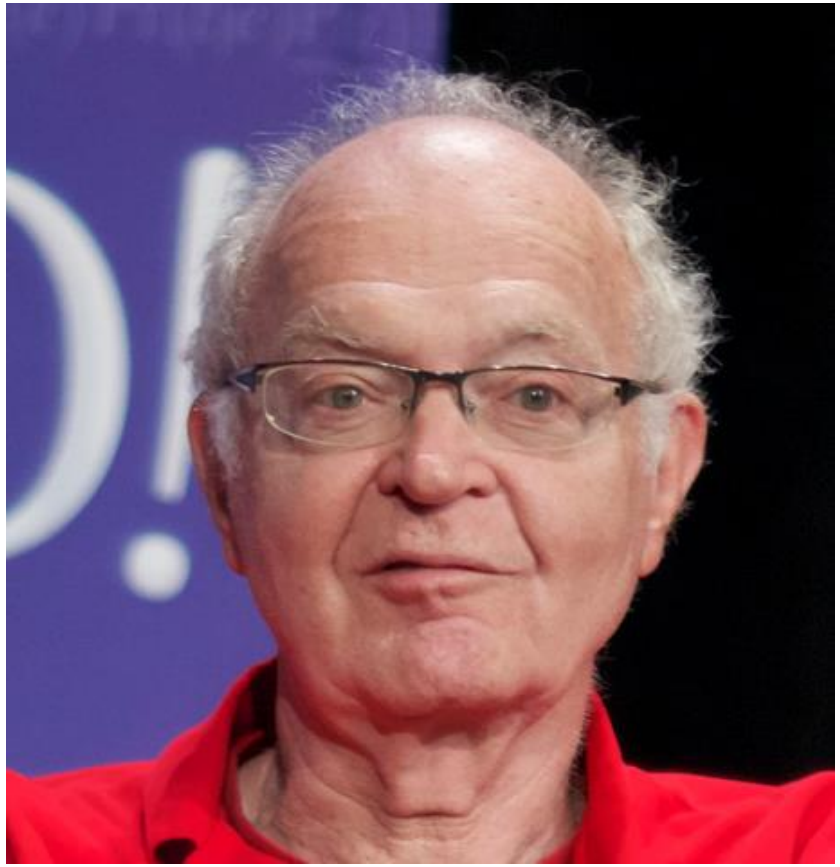
- There is no actual moving of data under the hood
- The ownership of the object just transfers

References

- https://en.cppreference.com/w/cpp/utility/move
- https://www.youtube.com/watch?v=ehMg6zvXuMY
- https://www.youtube.com/watch?v=OWNeCTd7yQE
- https://stackoverflow.com/questions/3413470/what-is-stdmove-and-when-should-it-be-used
- https://stackoverflow.com/questions/3106110/what-is-move-semantics
- https://en.cppreference.com/w/cpp/language/value_category

Person of the Day
Donald Knuth

- Recipient of the 1974 ACM Turing Award
- Known for his work with analysis of algorithms
- Author TeX, a typesetting language
- Author of *The Art of Computer Programming*
- Strongly dislikes patents to trivial solutions in software



Memory Safety

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

04/07/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Why is this an issue?

- C and C++ are low-level languages, which means they allow access and manipulation to memory directly as the developer
- This is a double-edged sword
  - Allows you to do some efficient and fast operations
  - But also requires you, as the developer, to be very cautious

Some stats

- Google reported that 70% of severe security bugs are actually memory issues [1]
- There is a recent push for memory-safe languages from the government (last several year)
  - Safecpp.org

Example 1: raw pointers

- What is a “dangling pointer”?
  - int* ptr = new int(20);
  - delete ptr;
  - ptr = nullptr;
  - std::cout << *ptr; // dereferencing the ptr leads to undefined behaviour

Example 1: Consequences

- Memory corruption
  - If the memory is reallocated elsewhere, then it can lead to memory corruption or overwrite the data
- Information leaks
  - Can lead to access to sensitive data if it is put there, or to privilege escalation (if that memory is used in security checks)
- Malicious code execution
  - If the pointer is used to make a virtual function call, then a different address (pointing to exploit code) can be called due to vtable pointer being overwritten

Use after free

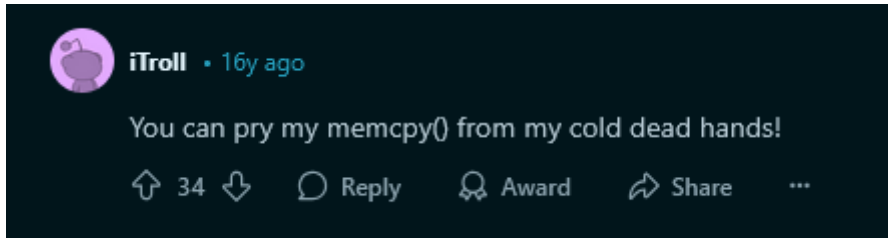
- When a dangling pointer is used after it has been freed without allocating a chunk of memory, it is known as a “use after free” vulnerability
- CVE-2014-1776 [2]: In Microsoft Internet Explorer version 6-11, this vulnerability allowed attackers to execute arbitrary code or cause DoS attack using memory corruption
- Exploited in the wild!

Example 1: fixes

- Use smart pointers
- Use audited pointers
- Be sure to always set the ptr back to null

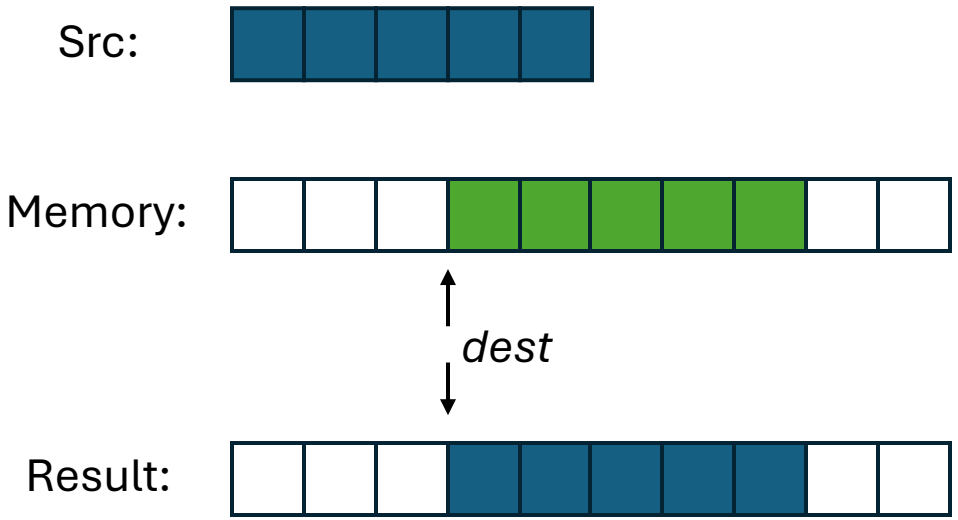
Example 2: Copying memory

- The C native functions memcpy() and strcpy() are some of the most dangerous functions
- Yet, they are very popular in the C community
- In fact, Microsoft banned their use almost 20 years ago in 2009 in their secure development cycle



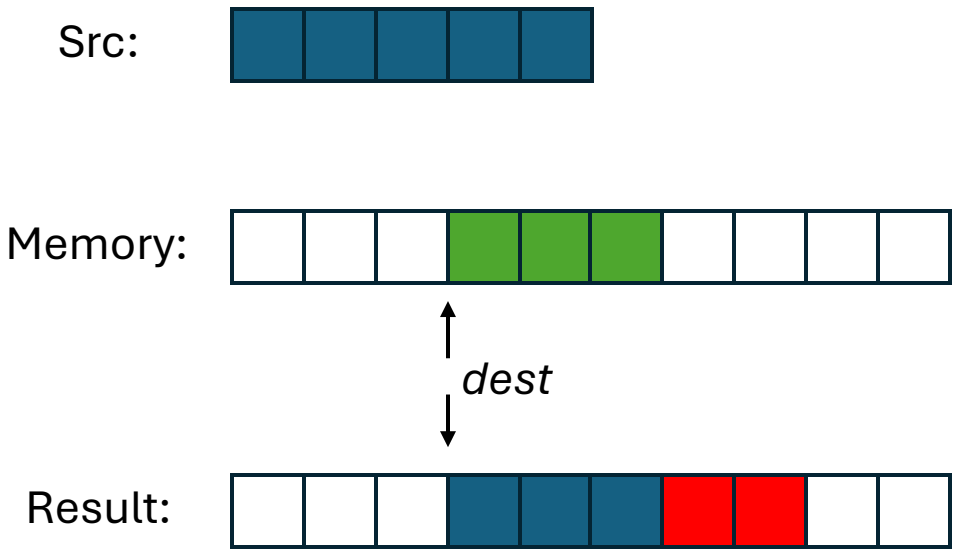
Memcpy() and strcpy()

- As the name suggest, it copies a buffer over from a *src* to a *dest*
- Assume *dest* is a buffer of char*[5]
- Ex: memcpy(*dest, src, 5)*



Buffer Overflow Attack

- If *dest* is a buffer of char*[3]
- Ex: memcpy(*dest, src, 5)*



- This becomes a really big problem if red contains code, or if sensitive information
- Imagine if red contains a bit that is True if user is authenticated

Underlying issue

- Memcpy() and strcpy() does not do bounds checking
  - It is assumed that the developer will always use it correctly
  - A contract you sign when you use C or C++

If you can write perfect code…

- Let’s assume that you have gained the ability to write perfectly efficient and safe code
- Is your code always safe?

- What kind of packages do you import?

Log4J incident

- A widely used logging tool for Java
- A bug allowed for attackers to do Remote Code Execution
- A zero-day attack was first reported in November 2021 (CVE-2021-44228)[3]
- This vulnerability has been in the tool since 2013
- An estimated 93% of enterprise cloud environment was affected [4]
- Some services include: AWS, Cloudflare, iCloud, Minecraft servers, Steam, Tencent, etc.

Summary

- Low-level languages allow developers to directly manipulate memory
  - This allows for flexibility and efficiency, but also puts the burden on the developer to write safe and good code
- Vulnerabilities are not limited to code that you write, but code that you use and import as well
- There is no silver bullet to solve security, but require diligence from us (the devs) to think about types of attacks that can affect our code

References

- [1] https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html
- [2] https://nvd.nist.gov/vuln/detail/CVE-2014-1776
- [3] https://nvd.nist.gov/vuln/detail/cve-2021-44228
- [4] https://www.wiz.io/blog/10-days-later-enterprises-halfway-through-patching-log4shell

Password and Authentication

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

04/09/2025

Kira Chan

https://cse.msu.edu/~chanken1/

A bit of motivation (and scary facts)

- As a software developer, there is something that you must remember

- **Users often will blindly trust the software to be always correct, unless proven otherwise.**
- You are the developer…

A bit of motivation (and scary facts)

- How many people use the same password for all their accounts?

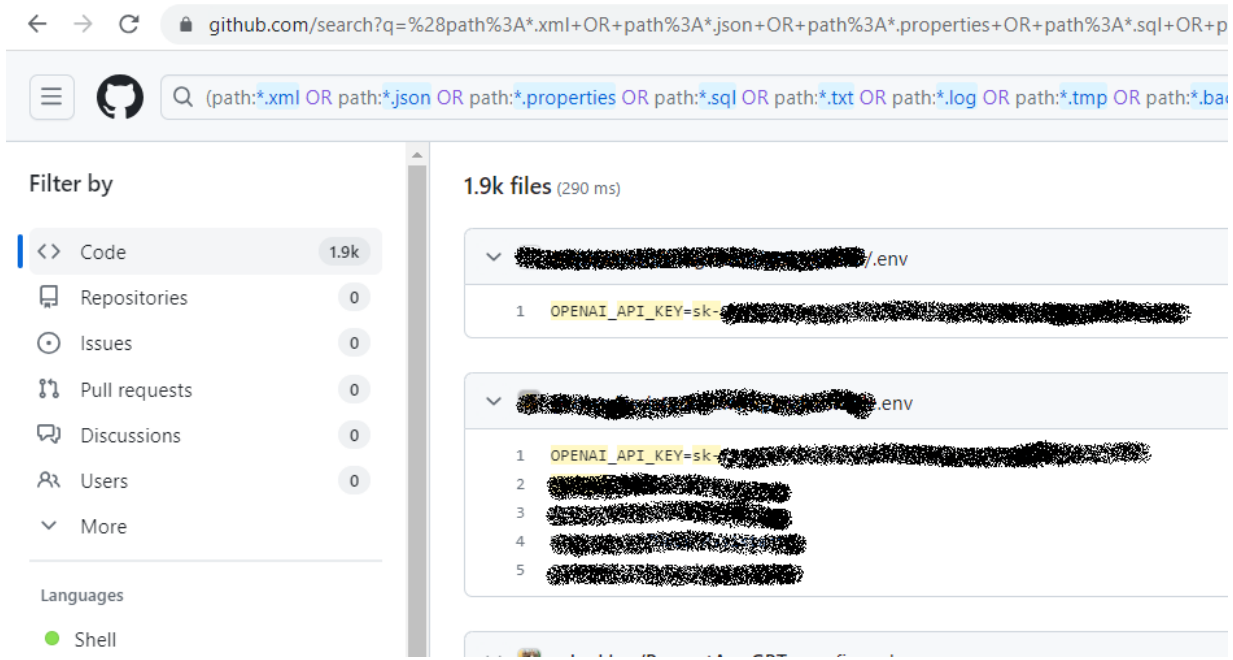
- There is no regulations for how passwords are stored
  - Stored in plaintext
  - Forwarded and sold on the black market
  - Stored securely with encryption
  - Stored *correctly*

Never store your password in plaintext

- This applies to both you as a developer, and you as a user

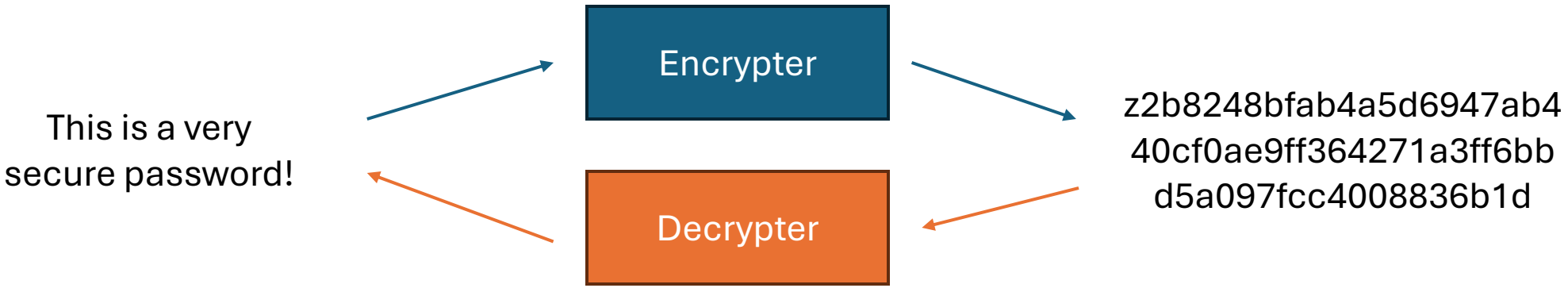
- Live example: storing them in a database, that is saved with permission 777
- Example 2: do you need an

  API key?
  - By the way, check your gitignore



How *should* you store a password?

- You should not encrypt passwords
  - Why?
- By definition, if you can encrypt something, you can decrypt something
- As a developer, do you ever need to know what a user's password is?



A better way

- Since you as the developer only has to check if two password matches, it is better to store them using a ***hash function***

- A hash function (mathematically) has to satisfy the following constraints:
  - Any input of arbitrary size maps to an output with a fixed number of char
  - Pre-image resistance: one-way function
  - Collision resistance: given two messages, hash(m1) != hash(m2)

Hash Function Example

- SHA256, SHA512
- MD5, MD6
- CRC32

- https://emn178.github.io/online-tools/sha256.html

- Example password input: This is a very secure password!
- Stored Hash: aebe5cbdd1f5dfa787baed77a5e1fe5d72ad23510b8862843e3256ec85530411

Is that it?

- One of the property of hash functions are that the same input leads to the same output

- Commonly used passwords?
  - Qwerty
  - Admin
  - 123
  - letmein
- Rainbow tables are precomputed hash tables, that you can use to match and discover the original password

Defence against the dark arts (of rainbow tables)

- We use something called a ***salt*** to strengthen these weak passwords
- A salt is just a unique, randomly generated string (16 char or more)
- Either do the following and check against input:
  - hash(password + salt)
  - hash(hash(password) + salt)
- They are stored right next to the password hash
- Simply used to increase the computation power needed to crack the password

Example database

Username = kira
Password = qwerty

Username	Password Hash	Salt
Kira	65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5	bd0ec0389e58eb22
Charles	992db5e2595725ef68eb5066e014da8c7f6baca0afa96f1021b56b66782133f2	b3b64768098d002b
Ed	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918	19b25856e1c150ca
Taylor	a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3	834cffc8b59b23ad

Just salt is a little boring…

- Why don’t we ***pepper*** the password as well.
- Similar concept to the salt, but a pepper is a separately stored string that is used across your entire login system
  - Salt is unique for each user
- Adds additional complexity for guessing passwords

- Hash(Hash(password + salt) + pepper)

Example database

Pepper: fb55fcb32 598e816

Username = kira
Password = qwerty

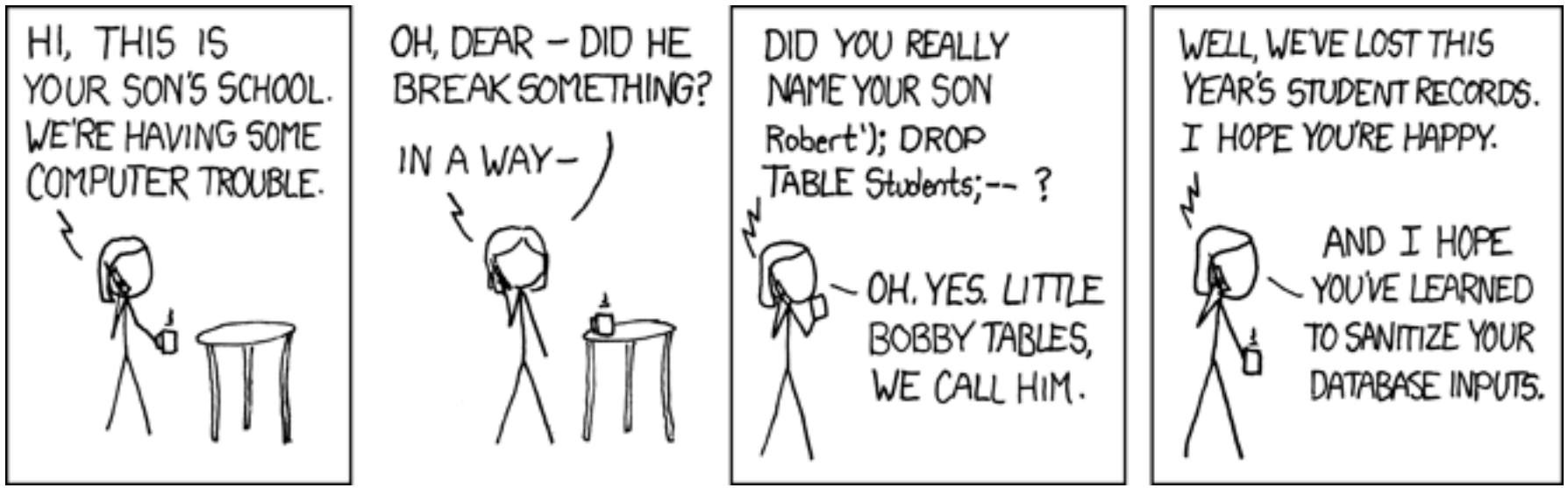
Username	Password Hash	Salt
Kira	dbaf05565ca7ee65ecb3801a37b925a50 eadef0add799e275d930c3a5976f9bb	bd0ec0389e58 eb22
Charles	d91c23455236eb2be812dea5bccfa3145 f374e4c325e258d7082d67c10668347	b3b64768098d 002b
Ed	998ed4d621742d0c2d85ed84173db569 afa194d4597686cae947324aa58ab4bb	19b25856e1c1 50ca
Taylor	8442af777e10f2da9f3b73e1daa0854b42 6cbdd833086c980698704c426ad5af	834cffc8b59b2 3ad

Hash(Hash(qwerty + bd0ec0389 e58eb22) + fb55fcb32 598e816) ?=

Sanitising inputs

- User inputs are evil
- They can do an SQL-injection attack (where they inject code at the end of their input) to bypass authentication
- Sanitise user input (strips all dangerous characters, or treats the entire input as string only)

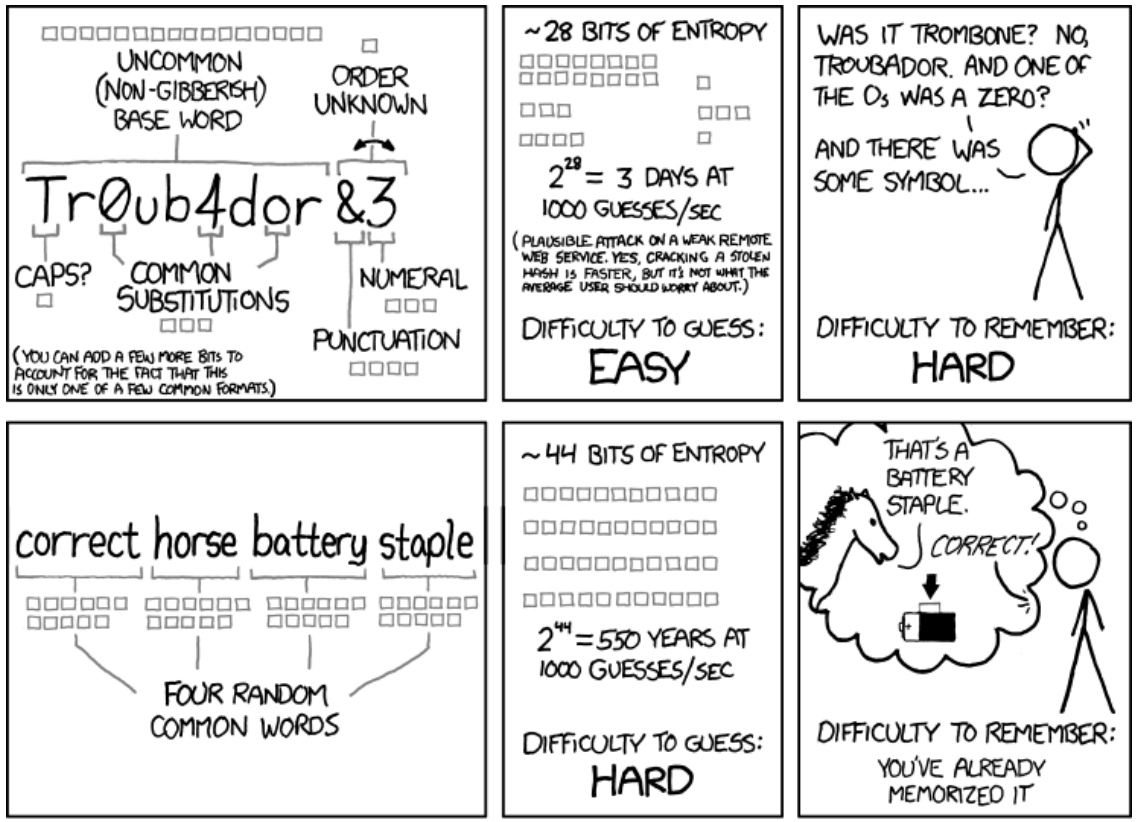
XKCD 327


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

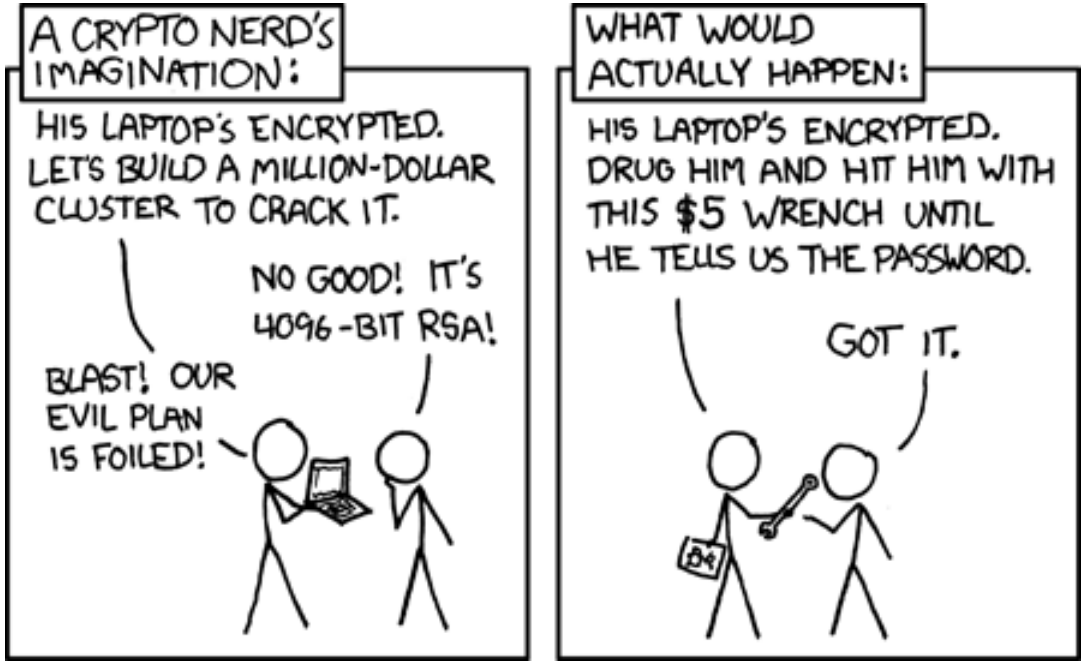
On the subject of password safety

- Do not use the same password across multiple sites, especially if the site is not reputable
- Long password with different character types are indeed better
- https://haveibeenpwned.com/
- 2 factor authentication is very good

XKCD 936


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

XKCD 538


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Person of the Day
Jack Black

And Jason Momoa… thank you for carrying the Minecraft movie

Person of the Day
Ron Rivest

- Worked extensively on cryptography and computer security
- Along with Adi Shamir and Len Adleman, invented the **RSA** algorithm
  - The most commonly used public-key cryptosystem
- Turing award winner (2002)
- Inventor of RC2, 4, 5, 6; MD2, 4, 5, 6
- Has a whole conference named after him

Byzantine General Problem

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

04/14/2025

Kira Chan

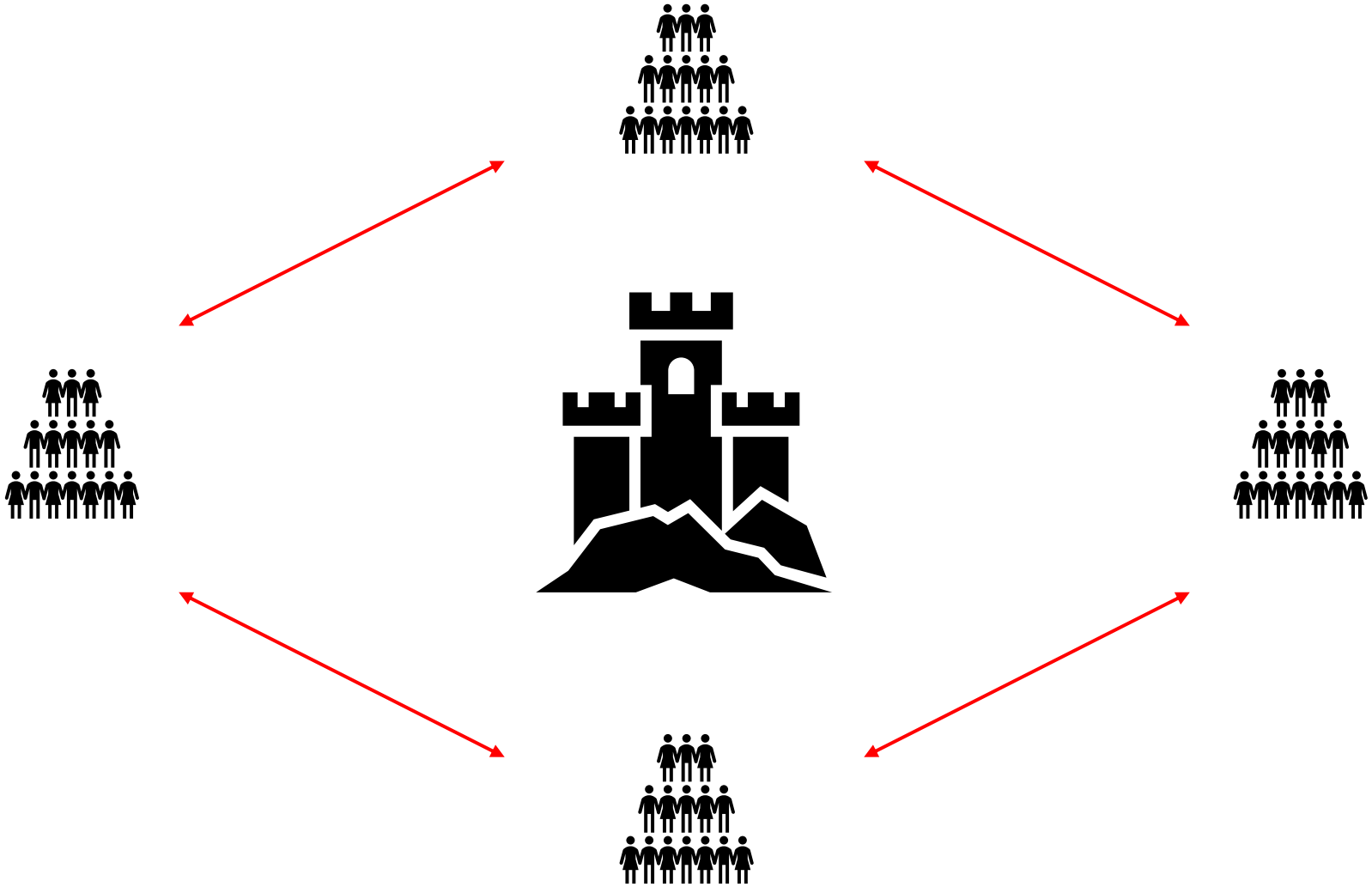
https://cse.msu.edu/~chanken1/

Problem

- In distributed computing, we have a notion of ***consensus***
- ***Consensus:*** how do we get multiple processes to agree

- Byzantine general problem demonstrates an instance of such problem
  - Created by Leslie Lamport et al. as an analogy
  - Generalisation of the two general problem

Byzantine General Problem



Byzantine General Problem

- You are the general of one of the *n* armies surrounding the city
- You have an option to attack or retreat
- If all the armies attack at the same time, then the attack is successful
- If all the armies retreat, then it’s OK but you did not conquer the city
- If some of the armies attack while some retreat, then you will fail

Complications

- What if a messenger gets captured before your message reaches its recipient?
- What if the acknowledgement message gets captured?
- What if the messenger is captured and the message is replaced to do the opposite?
- What if one of the general is a traitor and purposely sabotages your attack plans?

Unsolvability

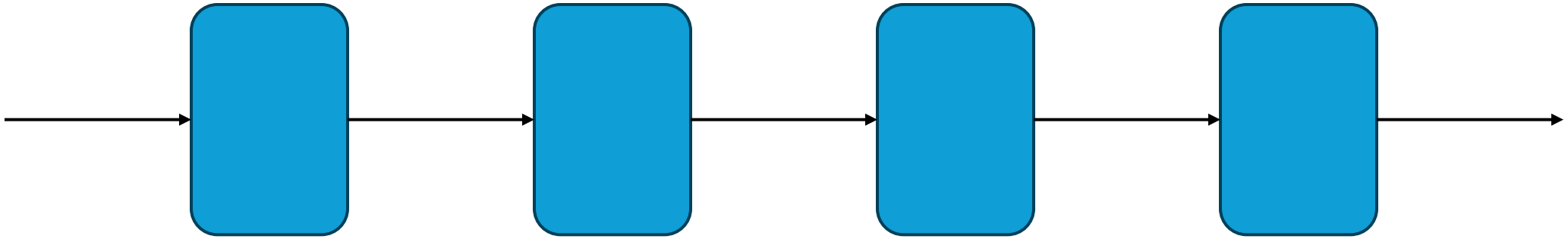
- In the general sense, this problem is unsolvable
- But it can be solved if we add some restrictions
  - If 2/3 of the generals are loyal

Byzantine Fault

- Byzantine fault is a fault that presents different symptoms to different observers
  - This is usually caused by imperfect information
- **Byzantine fault tolerance** describes the resilience of your system against Byzantine faults
  - What are some systems that is fault tolerant?

Partial solution: blockchains

- Blockchain deals with a system where the is no centralised authority
- Unlike traditional databases (e.g., a bank), blockchains maintain a distributed database called the ***ledger***
- Blockchain removes a single point of failure and ensures that data is not easily modifiable



On the mystery of bitcoin

Bitcoin: A Peer-to-Peer Electronic Cash System

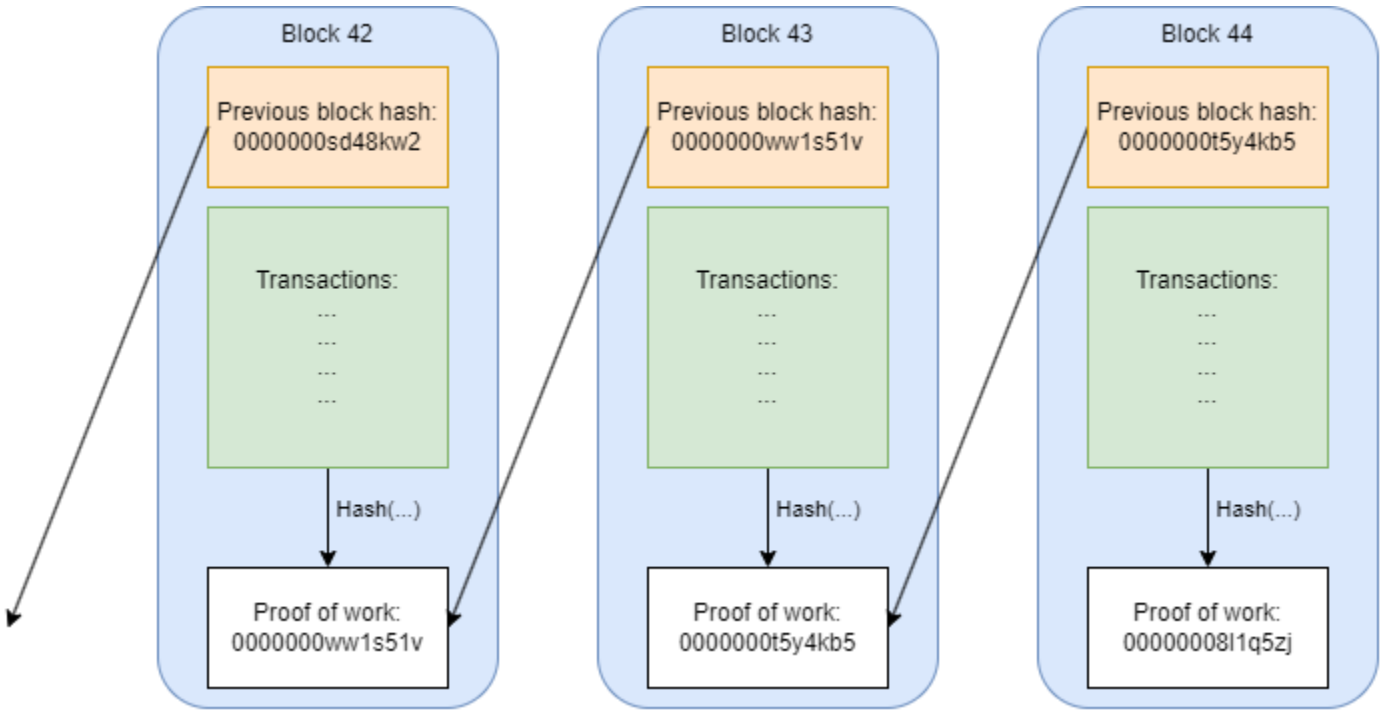
Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Blockchains

- The blockchain acts like a linked list
- Each node contains the following element:
  - 1) Hash of the previous block
  - 2) Transactions: monetary transactions or information to be stored
  - 3) Proof of work: a cryptographic puzzle
- Using blockchain, we remove the need to trust unknown nodes, rather trusting the collective system of nodes (which is assumed to be generally good)

Example blocks



The creativity of blockchain: Proof of Work

- The proof of work is a piece of data that is difficult to produce yet easy to verify
- Recall what a hash function is

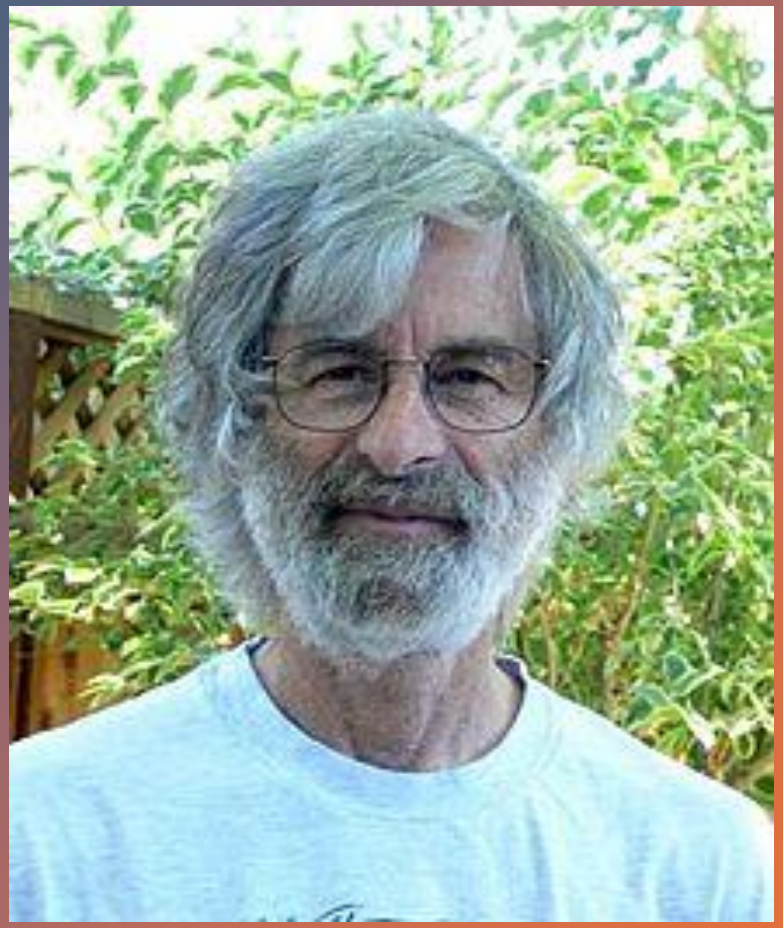
  - **Computationally hard to produce:** requires a node to correctly guess a “nonce” that when combined with the content of the block, produces a hash that starts with a given number of leading zeroes
  - **Computationally easy to verify:** verification is as simple as generating the node and ensuring it has the satisfying number of zeroes

Why it works

- The idea is blocks cannot be generated easily, and as such requires some amount of compute power to “sign” the blocks
- If an attacker wants to:
  - Modify a transaction (say add a digit to the amount transferred):
    - The new hash of the block will not have correct number of leading zeroes
    - They must recompute the hash to match
    - Increasingly harder as the blockchain gets longer
  - Add a malicious block or transaction:
    - Invalid transactions are rejected by other nodes (voting)
    - Including a block means the attacker is trying to outcompute all other nodes in the system
    - Since the longest block is used, they will have to keep outcomputing even if they successfully find a correct hash once

Summary

- From the original blockchain paper
  - The longest blockchain serves as a proof of the sequence of events witnessed, but also that it comes from the largest pool of computing power
  - Records are immutable
  - As long as majourity of the CPU power is controlled by nodes not trying to attack the system (they can each mind their own thing), they will generate the longest chain and outpace the attacker
- Applications are beyond currency, can be used to verify software, history records, voting records, etc.



Person of the Day
Leslie Lamport

- 2013 Turing award winner
- Initial developer of LaTeX
- Distributed systems (Byzantine general problem)
- Lamport’s clock
- Temporal Logic
- Lamport’s bakery algorithm

Building a safe software

- Something to keep in mind, users trust the app
- You build the app

Title

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

02/10/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Fault Tolerance

Conformity and User Expectation

A mini-lecture series

CSE498 Collaborative Design - Secure and Efficient C++ Software Development

01/15/2025

Kira Chan

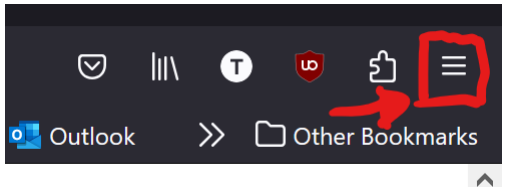
https://cse.msu.edu/~chanken1/

Pattern Recognition

- Humans are very good at pattern recognition
- Once they are used to one, it is hard to break out of a habit
- Your application should conform to user expectation or what they are used to
- When you are designing an interactive application
  - Icons must meet user expectation
  - Their location should match existing applications as well

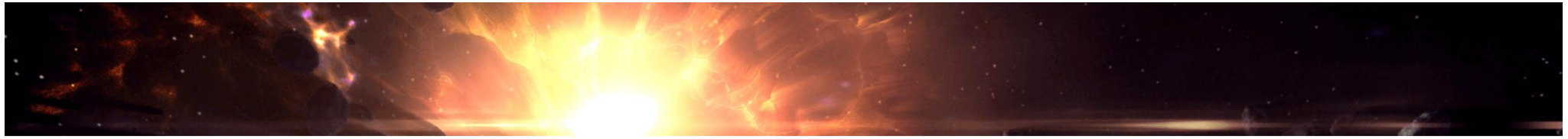
Some examples of breaking conformity

- What are these icons?



- Two lines of text, what do you expect?
  - https://www.youtube.com/watch?v=dQw4w9WgXcQ
  - https://www.youtube.com/watch?v=dQw4w9WgXcQ

What should icons do?



Kenneth (Kira) H. Chan Home Projects Teachings Academics People Resume CV About Me

Kenneth (Kira) H. Chan

ORCID

Hello, I am a Ph.D. student at the Department of Computer Science and Engineering at Michigan State University. My research field is the assurance and robustness of learning enabled systems (i.e., systems that rely on AI/ML for decision making). I apply a number of disciplines, such as evolutionary computing (e.g., novelty search), non-cooperative game theory, reinforcement learning, and other related techniques to address / improve the robustness of learning-enabled systems.

- Wow, this is a very uncomfortable slide to look at

Why It Matters

Example

- Icons should conform to what other applications are using
- Discord, Slack, Piazza:
  - The right contains a list of “channels”
- Functionality
  - Shift-click should add a new line in a text editor
  - Enter should send a message (?)
- Extends to a lot of things:
  - Consider coding:
    - Variable naming
    - Function name

MRU

What is the correct way to store elements in an array?

- FIFO?
- LIFO?
- Random?
- HEAP?

Theory of Most Recently Used (MRU)

- Keep items used frequently closer to the ends of the vector
- It is likely that they are used again
- Consider this
  - User 1: last accessed 4 years ago
  - User 2: last accessed 2 years ago
  - User 3: last accessed 1 day ago

LSP

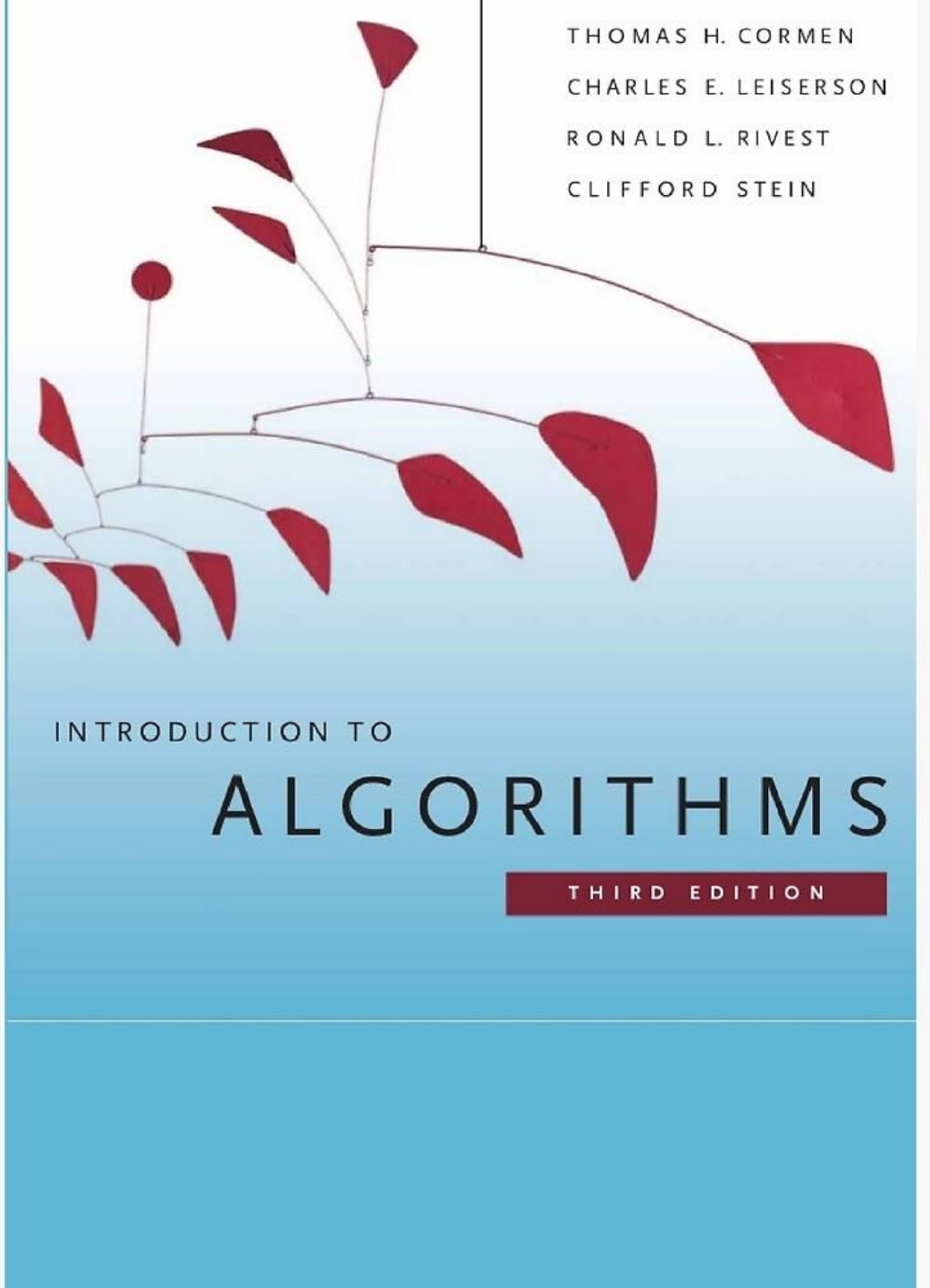
- https://stackoverflow.com/questions/56860/what-is-an-example-of-the-liskov-substitution-principle

Circle-ellipse problem

Board game example

Person (?) of the day
CLRS

- A very famous book by
  - Cormen
  - Leiserson
  - Rivest
  - Stein
- A very good read and reference
- Seriously, buy this book



Systematic and rigrous programming

- Height of fortran, spaghetti code
- Aliveness Halting problem

Steve Wozniak (Woz)

- “Mind” behind Apple

Mary Shaw

Person of the Day
“Uncle Bob”

- Robert C. Martin
- Known for introducing / promoting:
  - SOLID Principles
  - Test-driven Development
  - Software Craftsmanship

Topics or principles to consider

- Separation of concerns
- Use of AI in industry
- Principle of least privledge
- Single Responsibility
- Law of demeter or principle of least knowledge
- **Liskov Substitution Principle (LSP)**
- Solid principles
- https://ilegra.com/en/solid-understand-the-5-principles-of-object-oriented-programming/
- Byzantine sieging problem
- Markov State Machines
- Error, faults, violation of properties
- AB Testing
- Password storing, sanitation
- Circle programming lang (Sean Baxter)

Future people

- Leslie Lamport: Concurrency, latex, distributed systems, byzantine fault tolerance
- Watts Humphrey
- Barbara Liskov
- Von Neumann
    - Game theory, Linear programming, stability analysis, von neumann architecture, merge sort
- John Nash
- James Gosling (Java)
- Marvin Minksy (AI)
- Larry Page and Sergey Brin (Google)
- **Guido van Rossum (python)**
- Ron Rivest / Adi Shamir **(Crypthography)**
- **Radia Perlman (Spanning Tree, networking, mother of internet)**
- **Hedy Lamarr (Technology that formed basis for Wifi, GPS, bluetooth**
- Richard Hamming (Hamming distance)
- Richard Bellman (DP)
- Claude Shannon Father of information theory
- Euler
- https://en.wikipedia.org/wiki/Kurt_G%C3%B6del
- Ludwig Boltzman
- Audrey Markov
- Jason Turner (videos)
- Hurbs sutter
- Zero knowledge proof – Goldwasser
- Tom ray Digital evolution
- John Holland John kowzah
- John conway