CS202: Software Tools and Techniques for CSE

Lecture 1

Shouvick Mondal

shouvick.mondal@iitgn.ac.in
August 2025

Learning Outcomes

The students will understand the fundamentals and gain **hands-on experience** with **software tools** used by **academicians and researchers** in their day-to-day activities.

The course will prepare the students with the **necessary groundwork** and **exposure to the right mix of tools** before working with **computer systems** and undertaking **mainstream research** in domains such as analytical and experimental research.

About the Course

Instructor: **Shouvick Mondal**

TAs: Isha Jain, Drishti Bhandari

Course email (*all announcements will be made*):
cs202_stt_cse-aug-dec-2025.pvtgroup@iitgn.ac.in

Google classroom (*all lab assignments will be posted*):
https://classroom.google.com/c/Nzc3NDYzNjEyMTY2?cjc=2j4r42iw

Lectures: (Slot **D1**): Tue, 10:00–11:20 AM @ AB10/103
Lab sessions: (Slot **I1+K1**): Mon, 2:00–4:50 PM

- Batch 1.1 (@ AB10/104)
- Batch 1.2 (@ AB10/105)

Overall course load: {**14L** (x1)} + {**26P** (x1)}

- **Blue (lec.)**: AB10/103 (250x)
- **Red (lab.)**: AB10/104 (70x), AB10/105 (70x)

Month	Day
August	4, 11, 18, 25 2, 5, 12, 19, 26
September	1, 8, 15 2, 9, 16
October	6, 13, 27 7, 14, 21, 28
November	3, 10, 17 4, 11, ~~18~~

Course Evaluation

[**A1**: 12 marks] Assess'm. I on reports/deliverables (labs cond. Aug 4, 11, 18, 25) **[Deadline: Sep 2]**

[**E1**: 20 marks] Exam I **[Sep 19–26]**

[**A2**: 12 marks] Assess'm. II on reports/deliverables (labs cond. Sep 1, 8, 15) **[Deadline: Oct 10]**

[**A3**: 12 marks] Assess'm. III on reports/deliverables (labs cond. Oct 6, 13, 27) **[Deadline: Nov 6]**

[**A4**: 12 marks] Assess'm. IV on reports/deliverables (labs cond. Nov 3, 10, 17) **[Deadline: Nov 19]**

[**E2**: 20 marks] Exam II **[Nov 21–28]**

[**B**: 12 marks] Attendance (TAs must record **attendance in physical signature** for both lectures and lab. sessions)

All deadlines are firm, set at 23:59:59 hrs IST

Deliverables: Structured Reports + Codes

Report for assessment Ai must be submitted (<=80 pages single column) as **`<rollno>_Ai.pdf`** *//No other extension allowed*

Codes for assessment Ai must be submitted as **`<rollno>_Ai.zip`** *//No other extension allowed*

For all submitted reports/codes, TAs will generate similarity reports from either Turnitin, moss, or other plagiarism detection tools.

Allowable similarity ranges: {reports (to be checked by TAs): <=30%, codes (to be checked by TAs): need-specific}

Deliverables: Structured Reports + Codes

Assessment criteria on structured reports (+ codes submitted)	Marks (total 12)	Description of components
Introduction, Setup, and Tools	2	Overview, objectives, environment setup, tools, and versions used.
Methodology and Execution	5	Step-by-step procedure, code snippets, outputs, screenshots, and error handling.
Results and Analysis	2	Outputs, observations, key insights, and comparisons.
Discussion and Conclusion	2	Challenges, reflections, lessons learned, and summary.
Format and Writing Style	1	Report structure, grammar, and writing style (coherent, unambiguous).
Plagiarism Compliance		**Revise and resubmit if similarity exceeds** (reports: >30%, codes: need-specific).

Software Environment

Linux based softwares/packages are available through the following Virtual Machine (VM) image.

- Mondal, S. (2024). ***Virtual Machine for Software Engineering and Testing Group - IIT Gandhinagar***. Zenodo. https://doi.org/10.5281/zenodo.10467159.

Windows based softwares/packages are available from https://visualstudio.microsoft.com

- Download and install ***Visual Studio Community 2022***.
- *Troubleshooting*: No more support for ASP.NET Web Forms (.aspx) in Visual Studio 2022?
- *Troubleshooting*: A network-related or instance-specific error occurred while establishing a connection to SQL Server.

Course Contents

(tentative and subject to change at the sole discretion of the instructor)

Introduction to version controlling: local, centralized, distributed.

Agile development methods: Continuous integration/delivery/deployment. e.g., Git workflow, actions etc. Source code differencing algorithms in Git: myers, minimal, patience, and histogram.

Makefiles and build tools: Dependency rules, macros, suffix rules etc. build tools e.g., Gradle, Apache Maven etc.

Rapid application development strategy: introduction to event-driven programming and its structure, object based programming using windows form controls and event handling.

Data and connectivity mechanisms: Data binding in windows forms, connecting controls to data sources, displaying, and updating data.

Concept of Debugging/profiling: Visual studio debugger, GNU debugger, Linux perf – stat, trace, sampling and off-cpu profiling, hotspot UI frontend.

Learning Resources

Online:

- Continuous Integration and Delivery (***CircleCI***: https://circleci.com)

Textbook:

- Sharp, J. (2022). ***Microsoft Visual C# Step by Step***, 10th edition, Microsoft Press.
- Watson, K., Nagel, C., Pedersen, J. H., Reid, J. D., & Skinner, M. (2008). ***Beginning Microsoft Visual C# 2008***. John Wiley & Sons.
- Mark J. Price (2024). ***C# 13 and .NET 9 – Modern Cross-Platform Development Fundamentals***, 9th edition, Packt Publishing Ltd.

Reference:

- Soni, M. (2016). ***DevOps for Web Development***. Packt Publishing Ltd.
- Yusuf Sulistyo Nugroho, Hideaki Hata, and Kenichi Matsumoto. 2020. ***How different are different diff algorithms in Git? Use*** `--histogram` ***for code changes***. Empirical Softw. Engg. 25, 1 (Jan 2020), 790–823.

Introduction to version controlling: local, centralized, distributed.

We all use some software for our daily needs

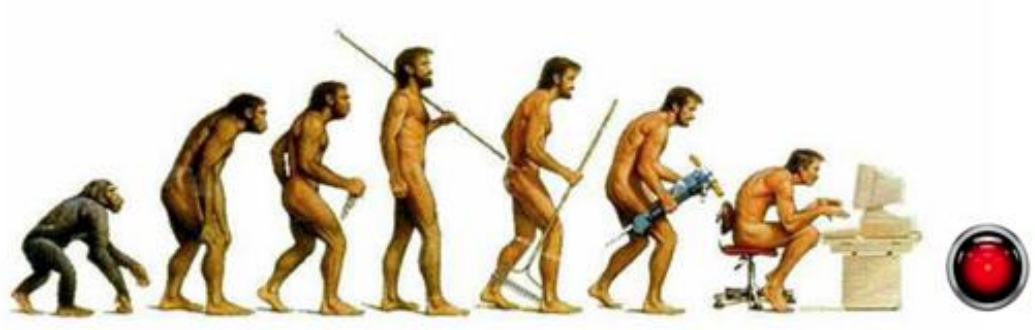
But is my software okay?
How do I know that I can trust it?

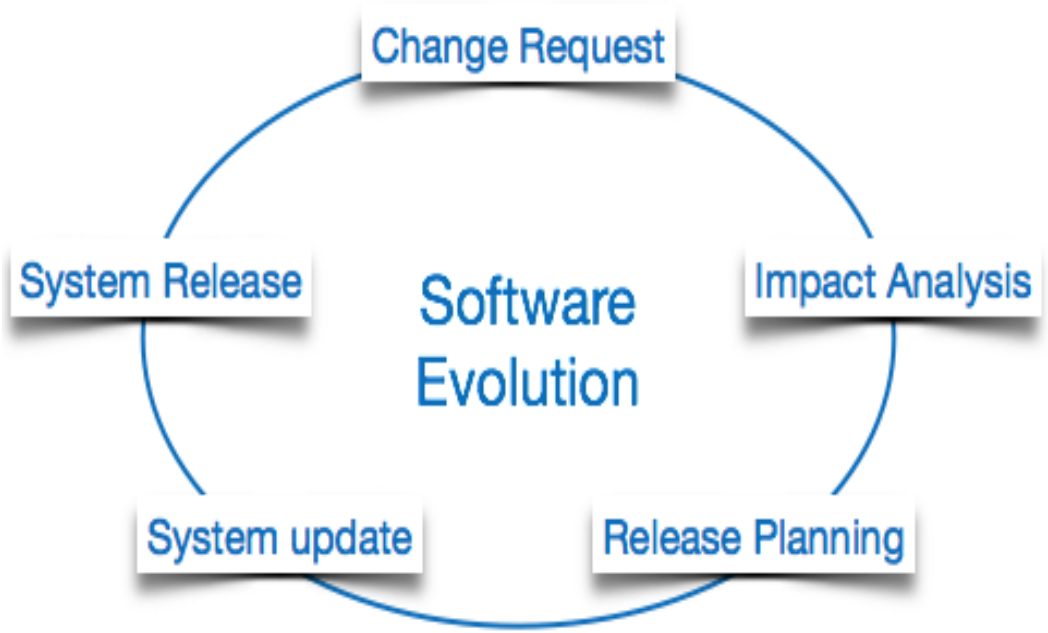
Software measurement
Testing is one way to do it...

Quality assurance: once is NOT enough!

Our needs change over time. To synchronize, software also undergoes changes. Testing them is a repetitive process…

Evolution and the process to deal with it...

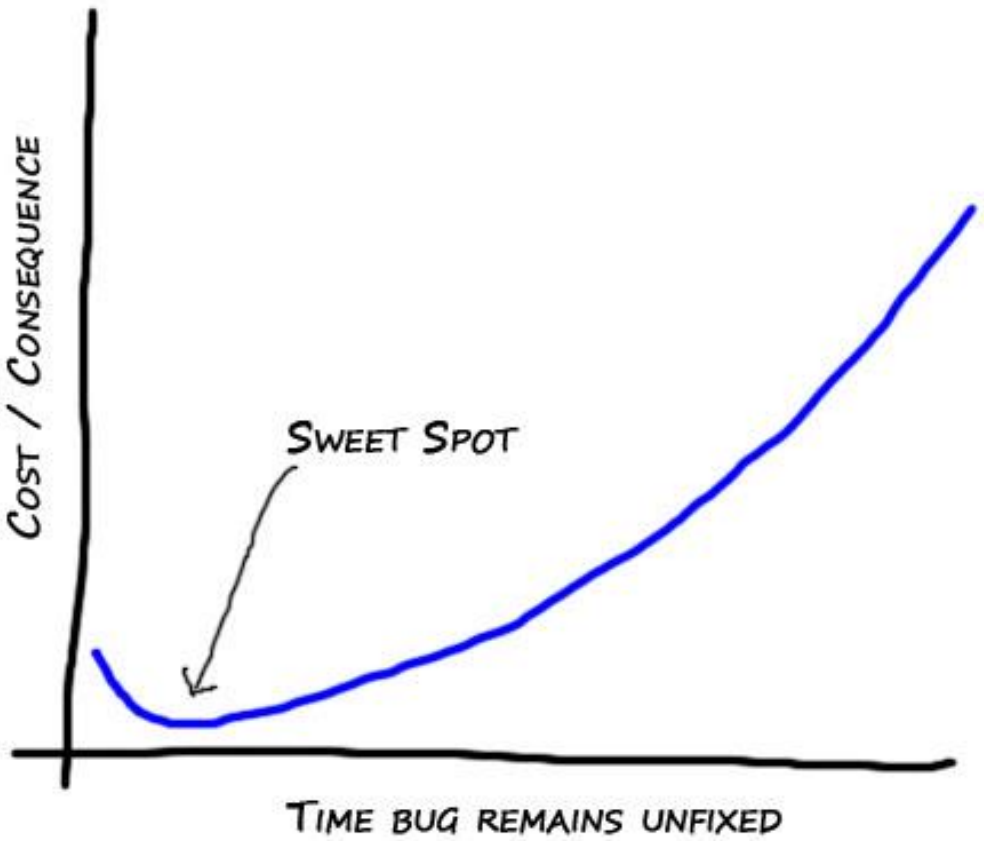


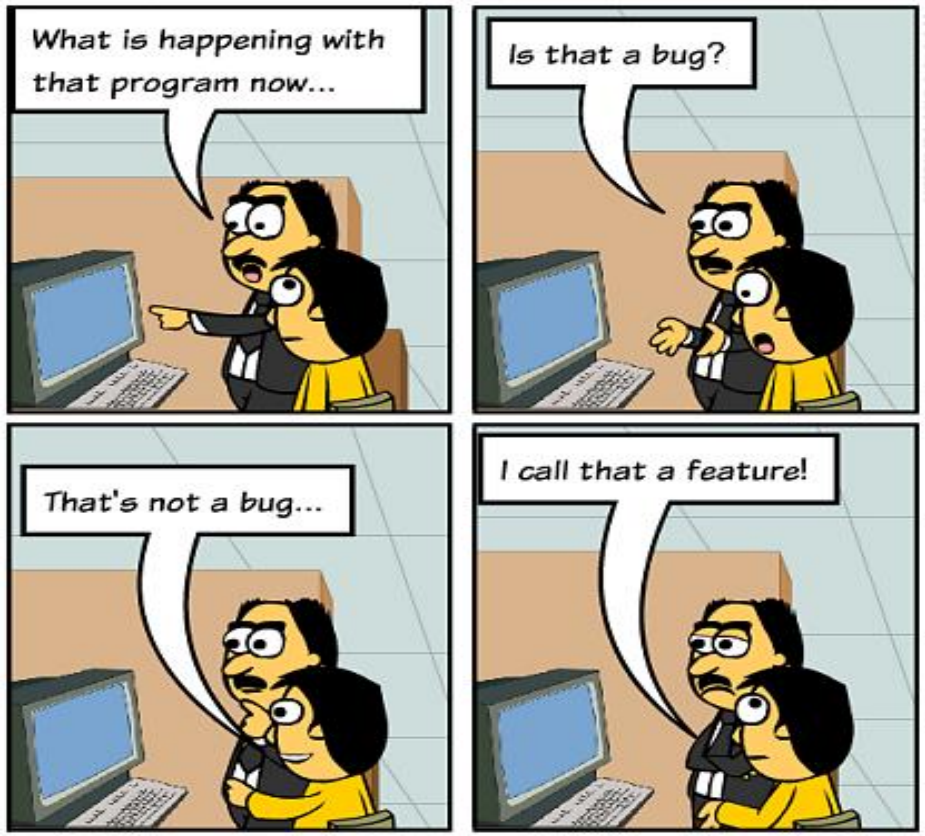


What are we testing for? BUGS

Bugs incur loss of assets...

Timing is important: the earlier the better…





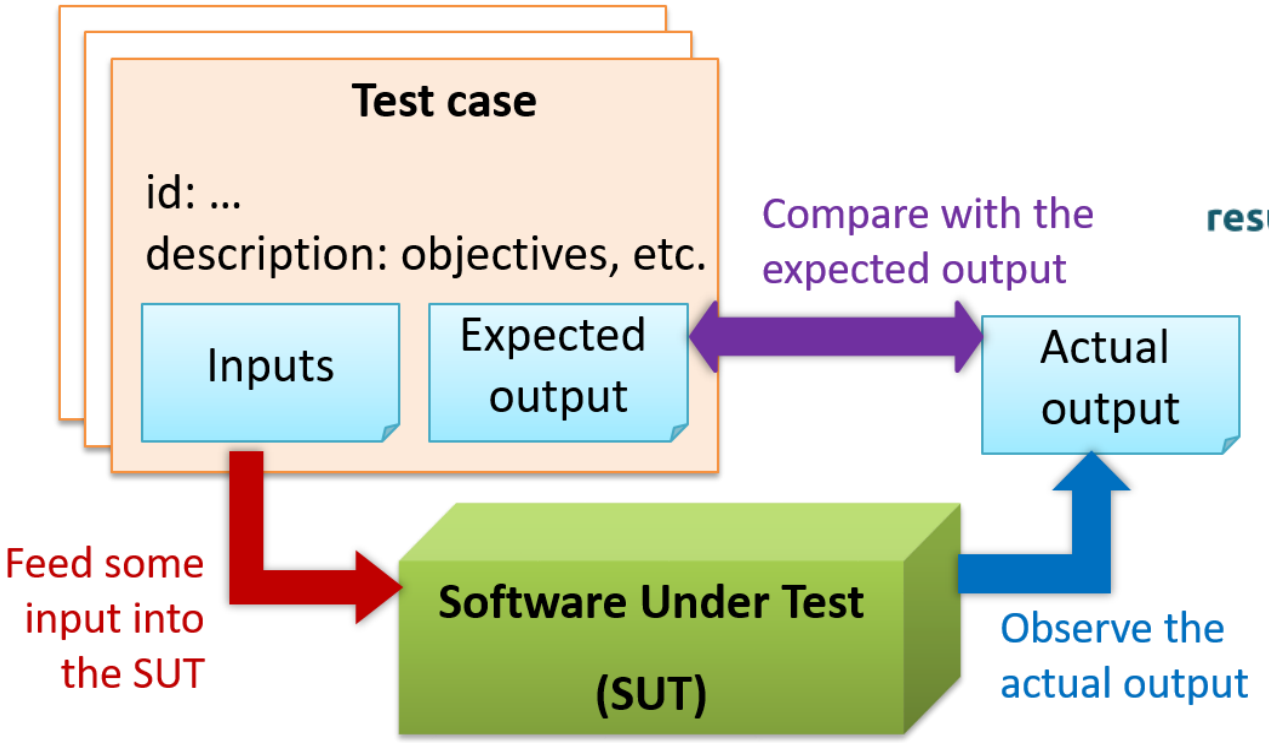
Usual software bugs

- **Functional bugs.** These are failures in the general workflow of the app caused by **improper system behavior** or enabled product features.
- **Visual bugs**. Layout and user interface **distortions** or mistakes.
- **Syntactic bugs.** The **grammar mistakes** or misspelled words and sentences used in product GUI.
- **Performance bugs**. Problematic **slowness**, **hanging**, or sluggish interface.
- **Crash bugs**. The **unexpected failure** of the program to work and function at all.

Unusual software bugs

- **Heisenbug**. The greatest trick of this bug is that it **disappears or alters its behavior when one attempts to fix it**.
- **Bohr Bug**. This unusual error **replicates itself** many times and **manifests reliably under a possibly unknown** but well-defined set of conditions.
- **Mandelbug**. Usually, it underlines **causes that are so complex** and obscure to predict. Mandelbug has **chaotic behavior**.
- *There are other species too!*

How do we test for software bugs? **Test cases**



Test case: a scenario/use case/input which **executes** the software to *observe some interesting events*.

Example of testing-and-debugging a C program (for division): *version **0** with **4** test cases*

```c
//original program
#include<stdio.h>
int main(void)
{
    int x,y;
    float res;
    scanf("%d %d",&x, &y);

    res=x/y;
    printf("%d/%d =%.2f\n",x,y,res);

    return (0);
}
```

```c
//statement of interest
#include<stdio.h>
int main(void)
{
    int x,y;
    float res;
    scanf("%d %d",&x, &y);

    res=x/y;
    printf("%d/%d =%.2f\n",x,y,res);

    return (0);
}
```

Example of testing-and-debugging a C program (for division): *version **0** with **4** test cases (using gcc 9.3.0)*

```c
//statement of interest
#include<stdio.h>
int main(void)
{
    int x,y;
    float res;
    scanf("%d %d",&x, &y);

    res=x/y;
    printf("%d/%d =%.2f\n",x,y,res);

    return (0);
}
```

```
$ ./a.out < t1
1/1 = 1.00

$ ./a.out < t2
5/4 = 1.00

$ ./a.out < t3
3/4 = 0.00

$ ./a.out < t4
15/5 = 3.00
```

Example of testing-and-debugging a C program (for division): *version **1** with **4** test cases (using gcc 9.3.0)*

```c
//statement of interest
#include<stdio.h>
int main(void)
{
    int x,y;
    float res;
    scanf("%d %d",&x, &y);

    res=(float)(x/y);
    printf("%d/%d =%.2f\n",x,y,res);

    return (0);
}
```

```
$ ./a.out < t1
1/1 = 1.00

$ ./a.out < t2
5/4 = 1.00

$ ./a.out < t3
3/4 = 0.00

$ ./a.out < t4
15/5 = 3.00
```

Example of testing-and-debugging a C program (for division): *version **2** with **4** test cases (using gcc 9.3.0)*

```c
//statement of interest
#include<stdio.h>
int main(void)
{
    int x,y;
    float res;
    scanf("%d %d",&x, &y);

    res=(float)x/y;
    printf("%d/%d =%.2f\n",x,y,res);

    return (0);
}
```

```
$ ./a.out < t1
1/1 = 1.00

$ ./a.out < t2
5/4 = 1.25

$ ./a.out < t3
3/4 = 0.75

$ ./a.out < t4
15/5 = 3.00
```

Example of testing-and-debugging a C program (for division): *version **2** with **7** test cases (using gcc 9.3.0)*

```c
//statement of interest
#include<stdio.h>
int main(void)
{
    int x,y;
    float res;
    scanf("%d %d",&x, &y);

    res=(float)x/y;
    printf("%d/%d =%.2f\n",x,y,res);

    return (0);
}
```

```
$ ./a.out < t1
...

$ ./a.out < t5
1/0 = inf

$ ./a.out < t6
0/0 = -nan

$ ./a.out < t7
0/1 = 0.00
```

Fault model of software testing

Failure: the current *output deviates* from the expected output.

– the C function `int excessTwo(int x);` ***returns 6 instead of 5***, for a test-case with input value 3.

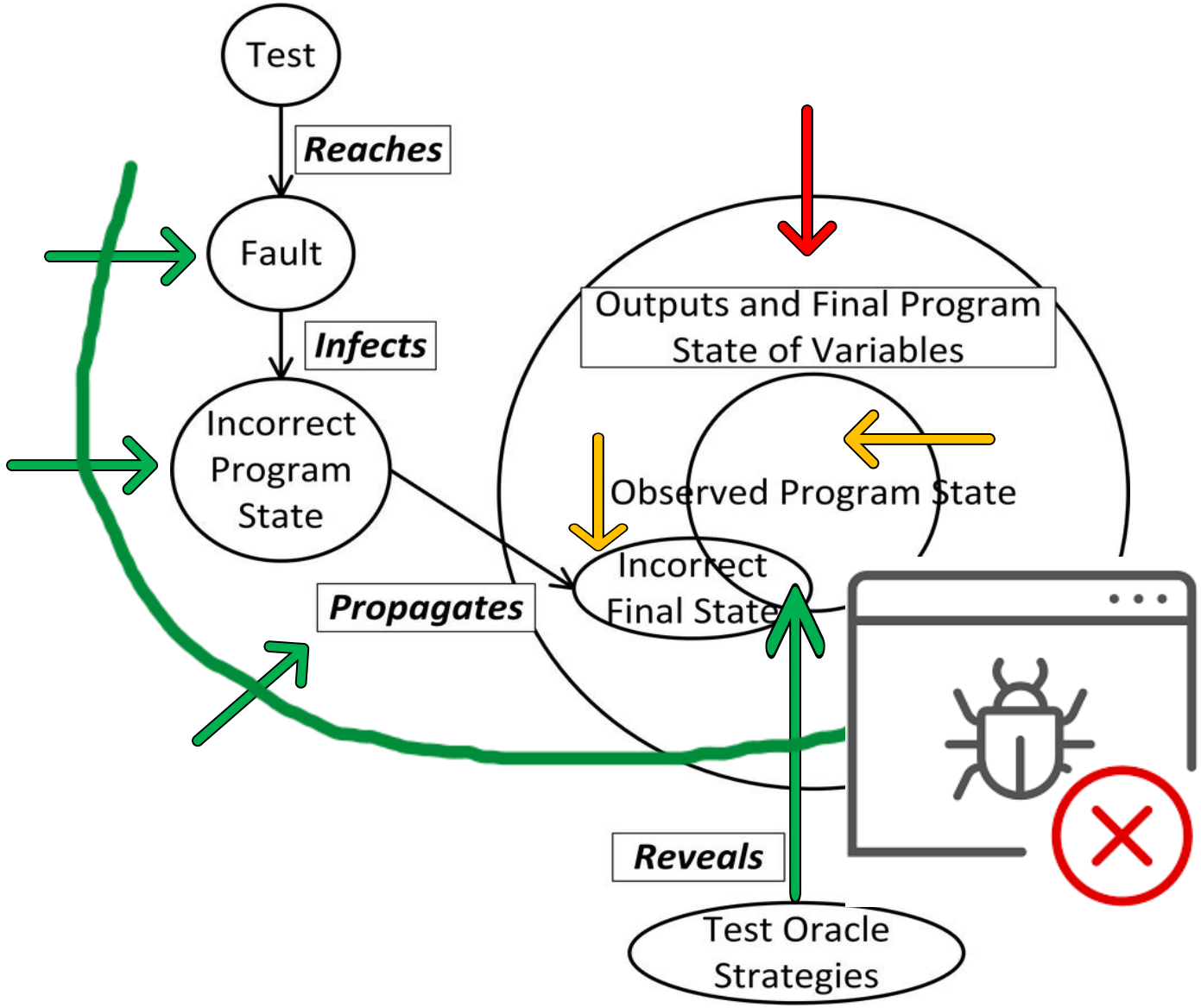
Error: a *system state* that might cause a failure.

– the definition `int excessTwo(int x) {y=2*x; return(y);}` has the ***intermediate program state***, `2*x`, in ***error***.

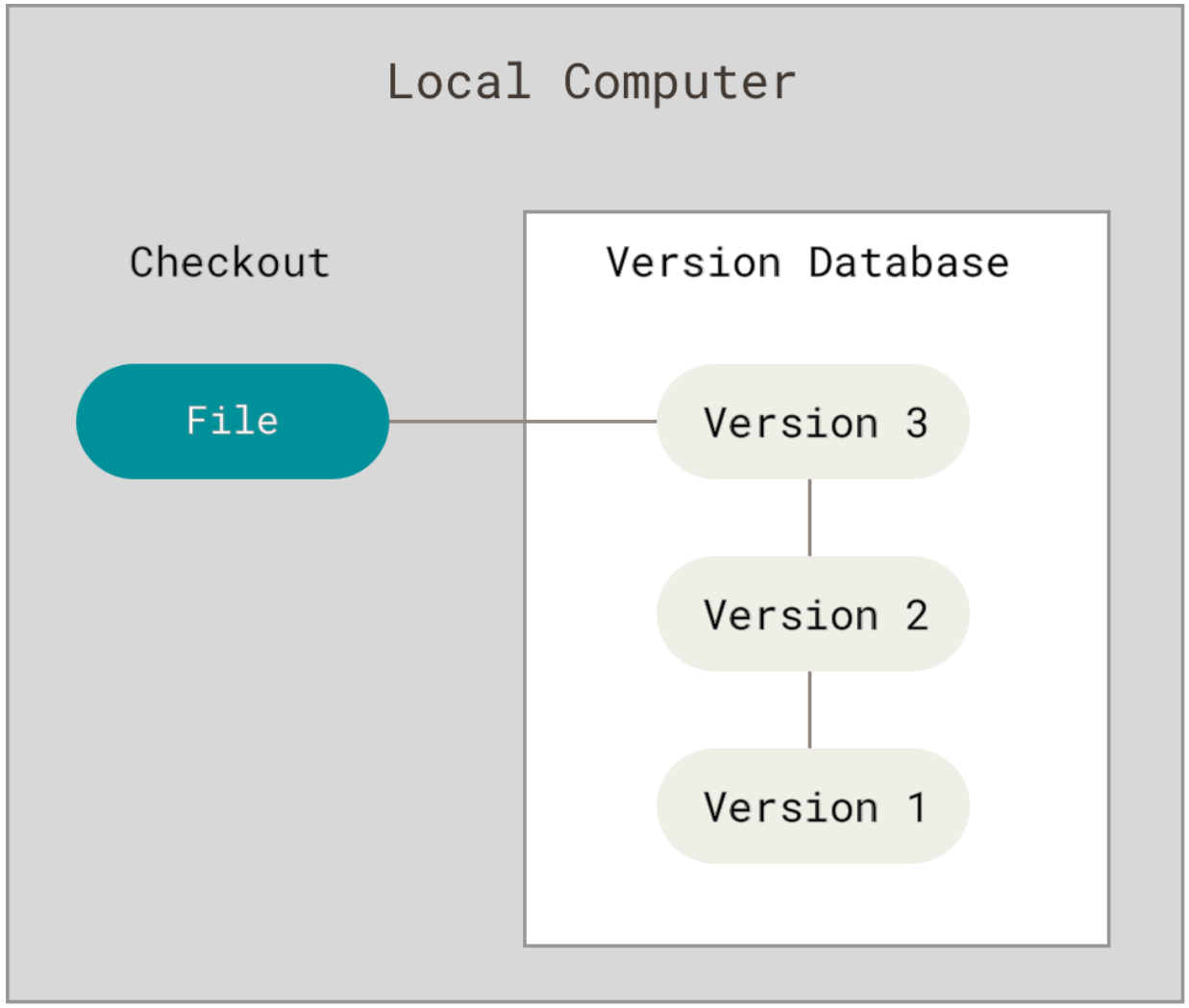
Fault: the *root cause* which gives rise to an error.

– the change `{...y=2+x;...}` → `{...y=2*x;...}` introduces the ***binary operation fault***: multiplication (`*`).

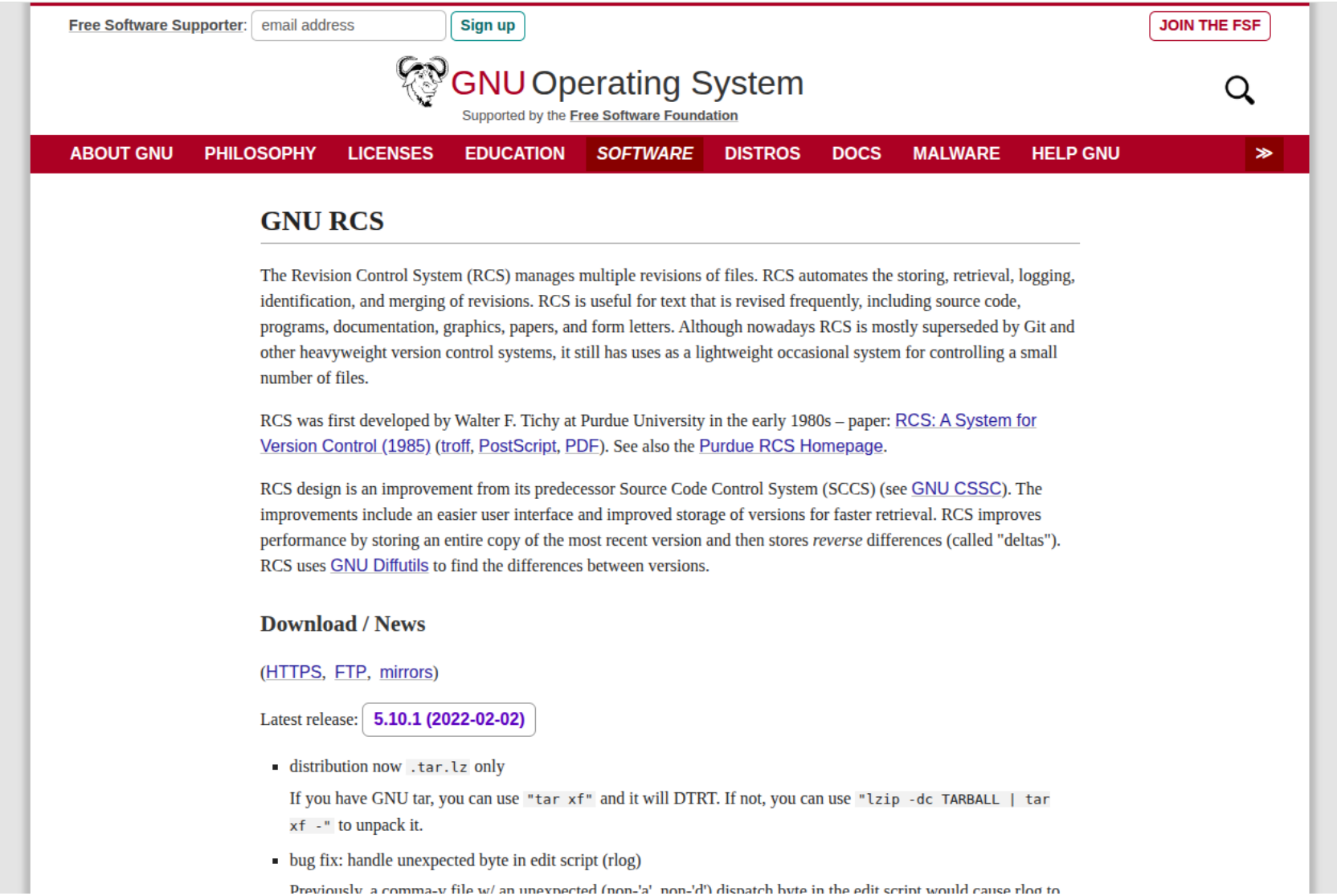
RIPR: Reachability, Infection, Propagation, Revelation



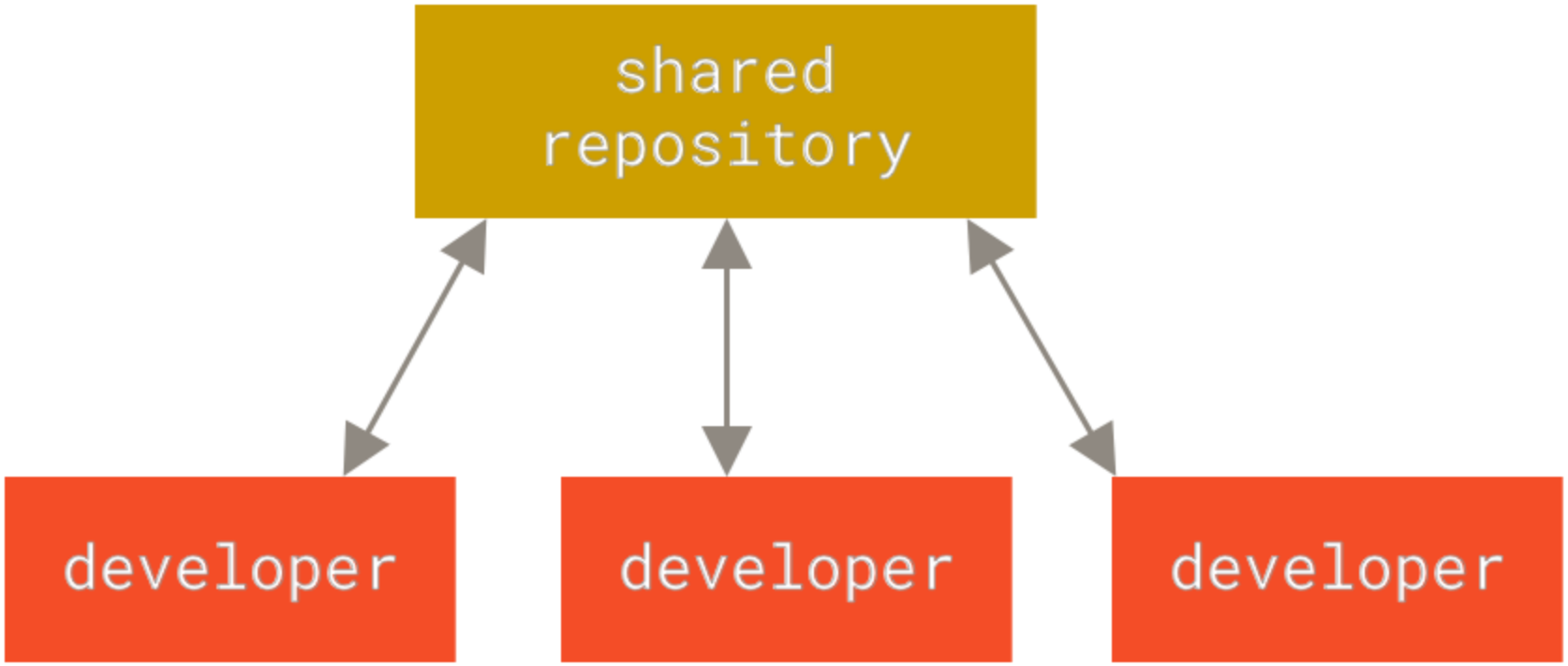
Local Version Control System (L-VCS)



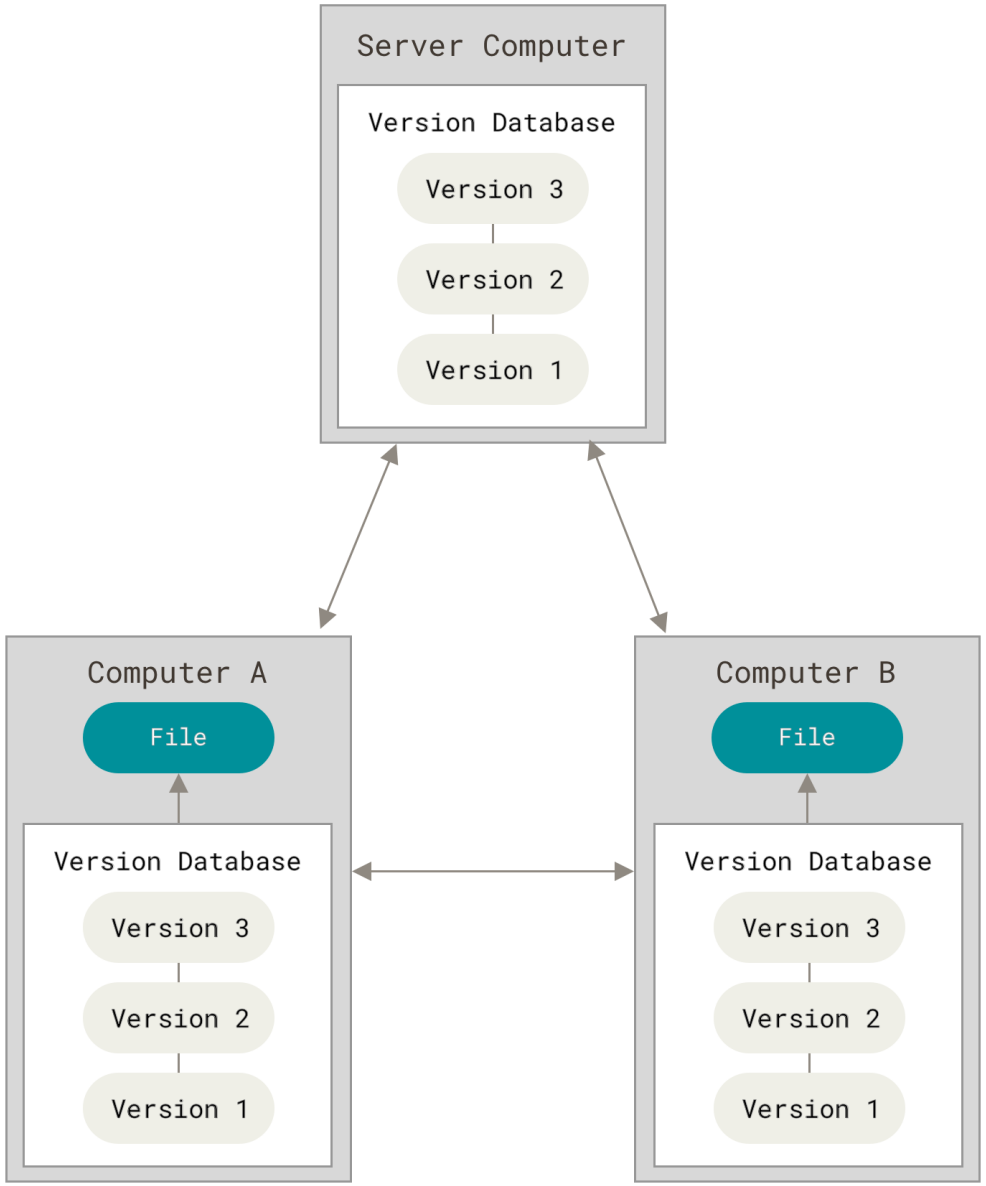
One of the Most Popular VCS Tools: **GNU RCS**



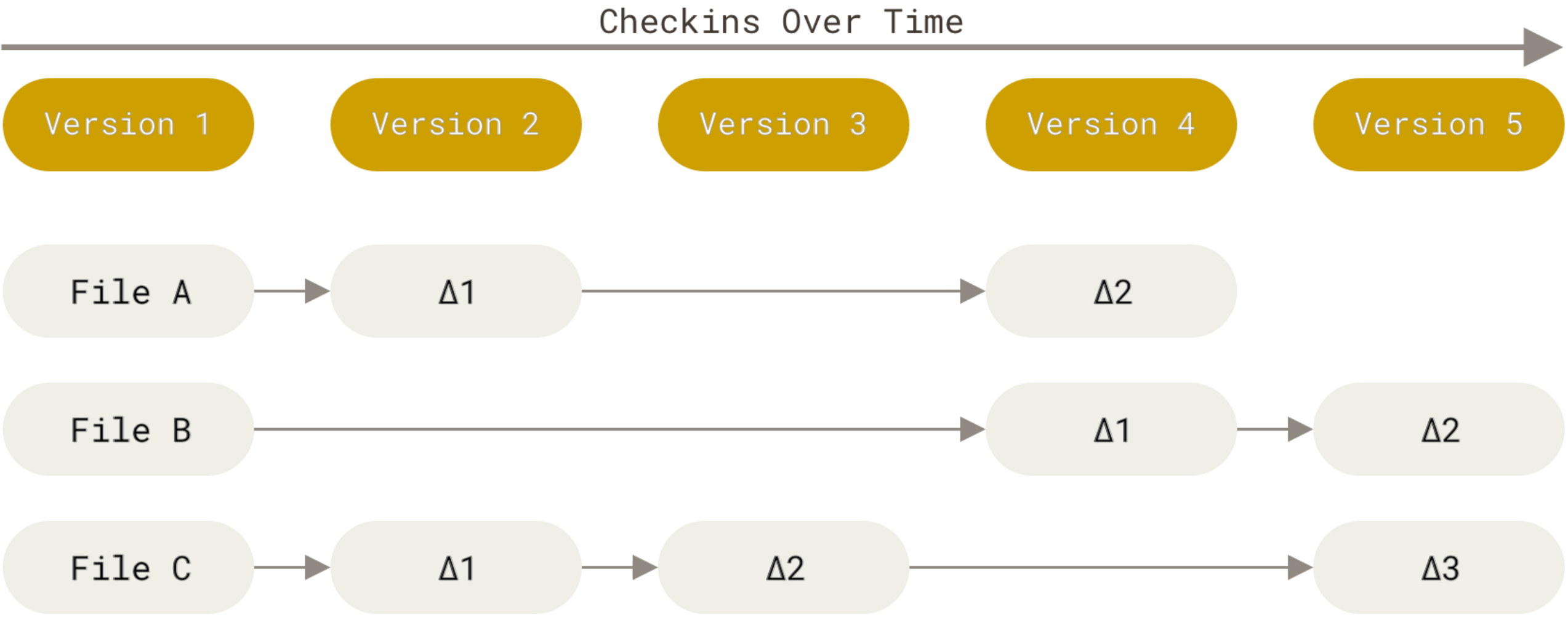
Centralized Version Control System (C-VCS)



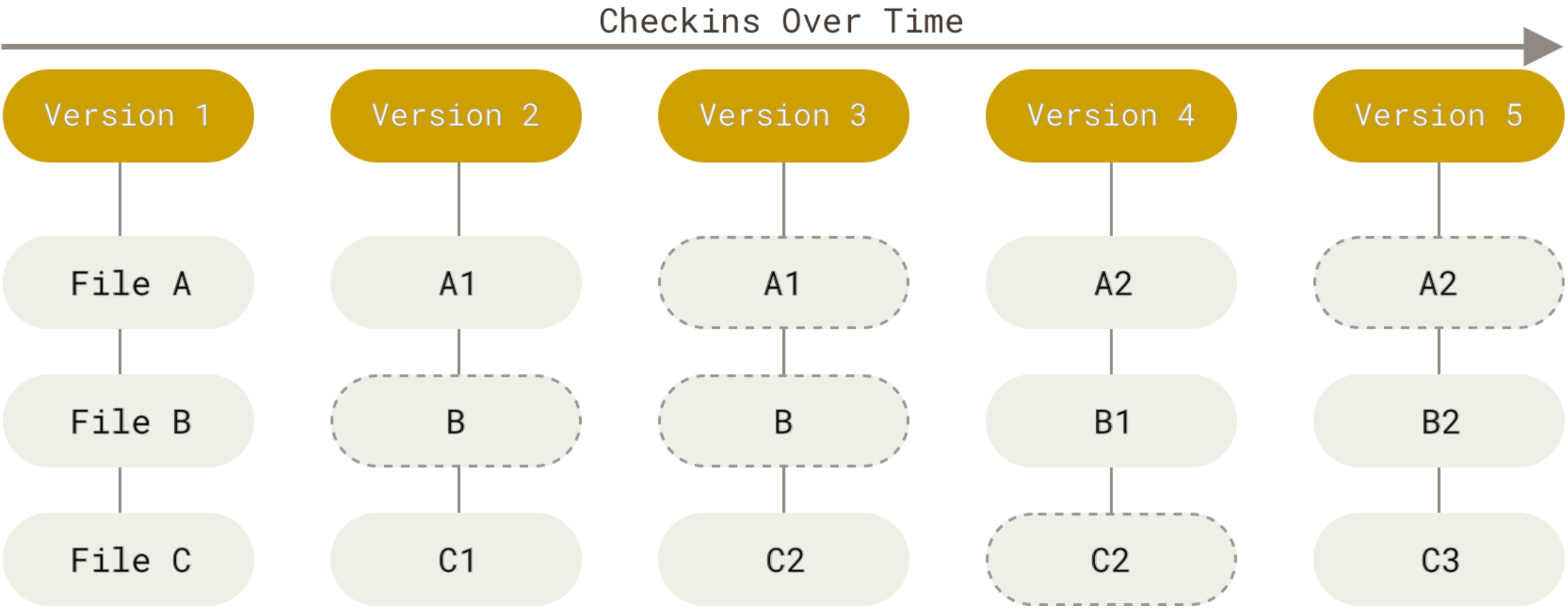
Distributed Version Control System (D-VCS)



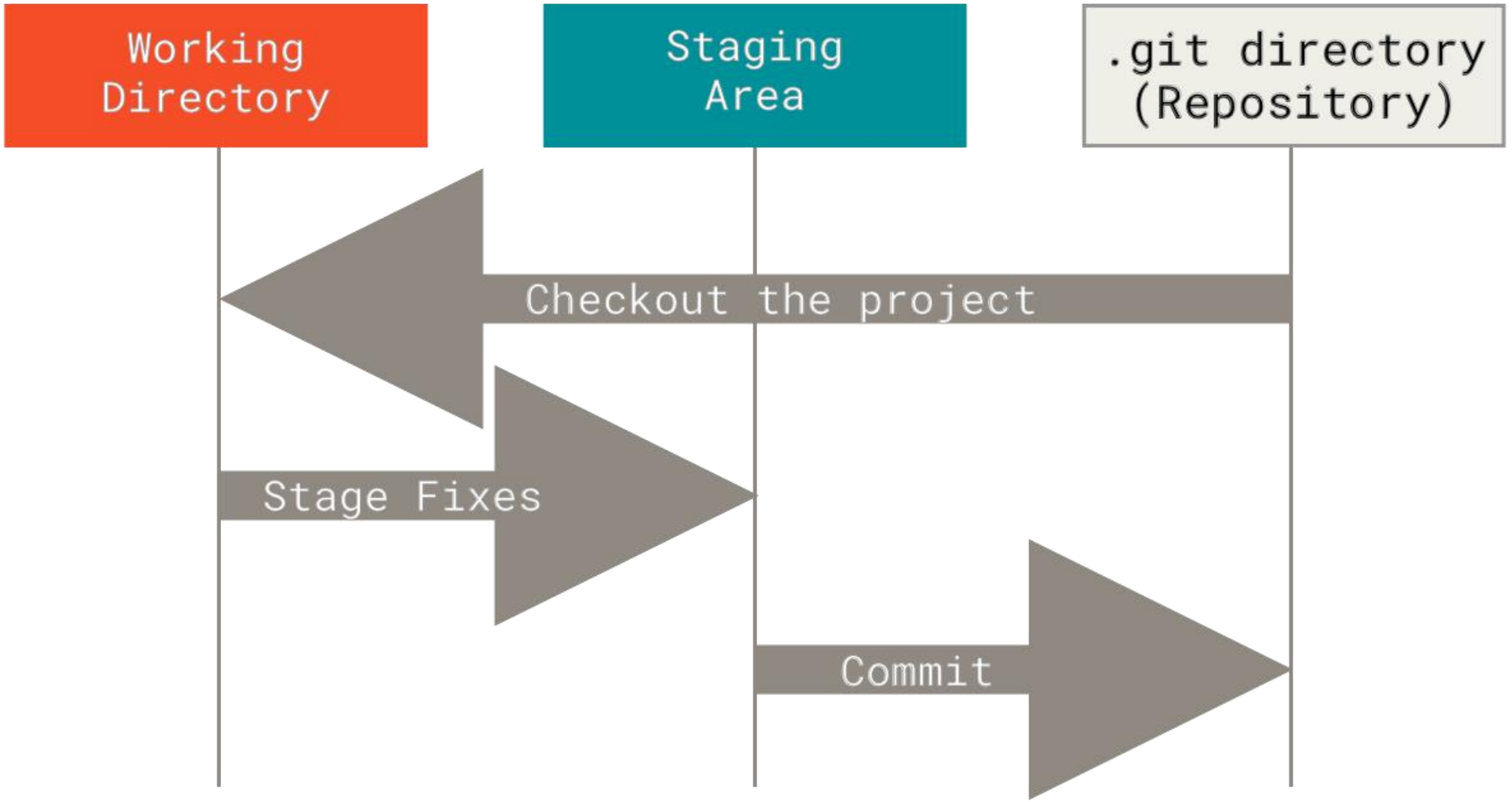
svn: storing data as changes to a base version of each file



`git`: storing data as snapshots of the project over time



Working tree, staging area, Git directory



Agile development methods: **Continuous** integration/delivery/deployment. e.g., Git workflow, actions etc. Source code differencing algorithms in Git: myers, minimal, patience, and histogram.

DevOps is a combined approach of Software Development (Dev) and IT Operations (Ops)

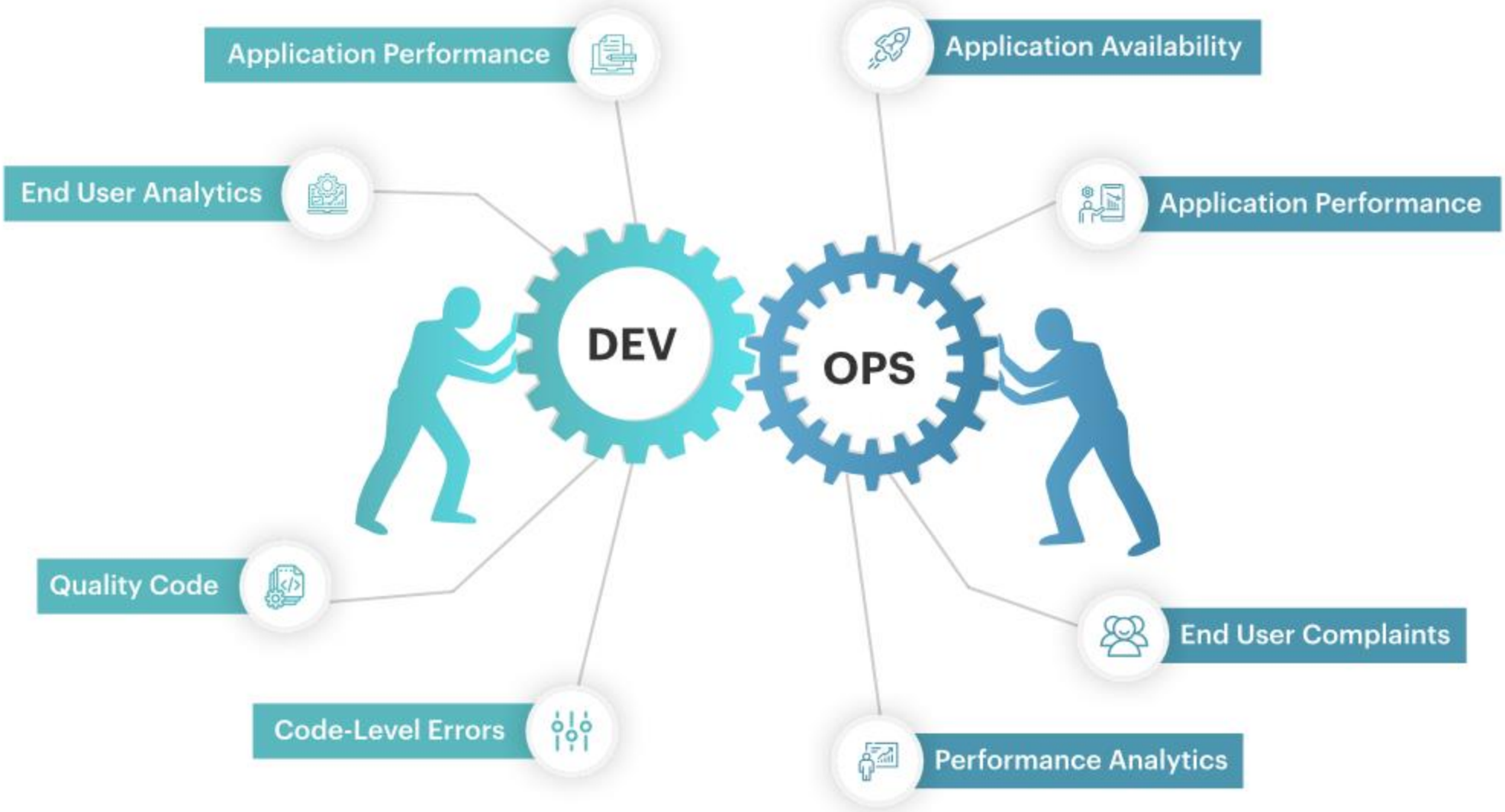


Image source: https://www.openxcell.com/devops

The DevOps Lifecycle

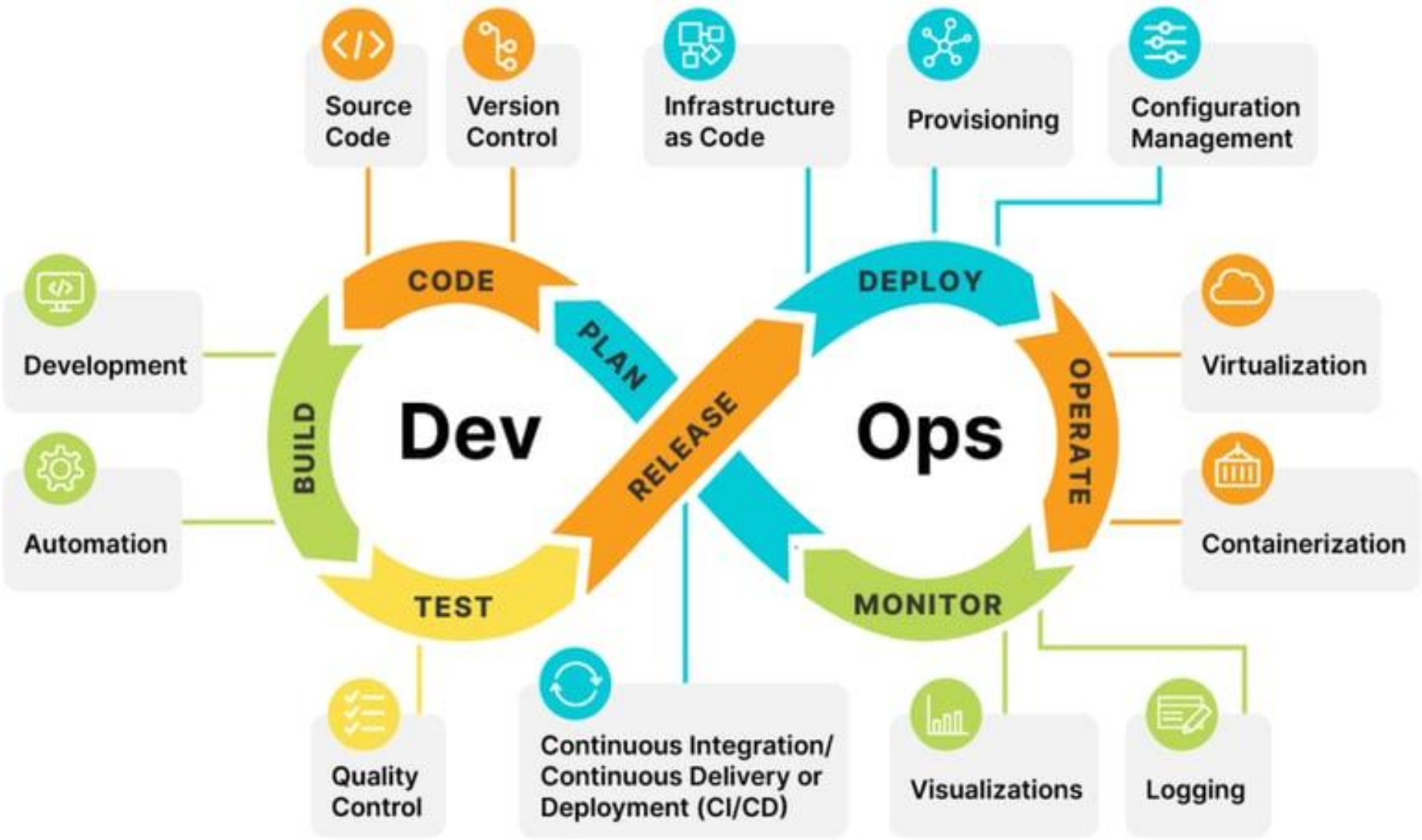


Image source: https://dev.to/rubixkube/the-future-of-devops-how-ai-is-shaping-infrastructure-management-3b95

DevOps Tools list according to DevOps Lifecycle

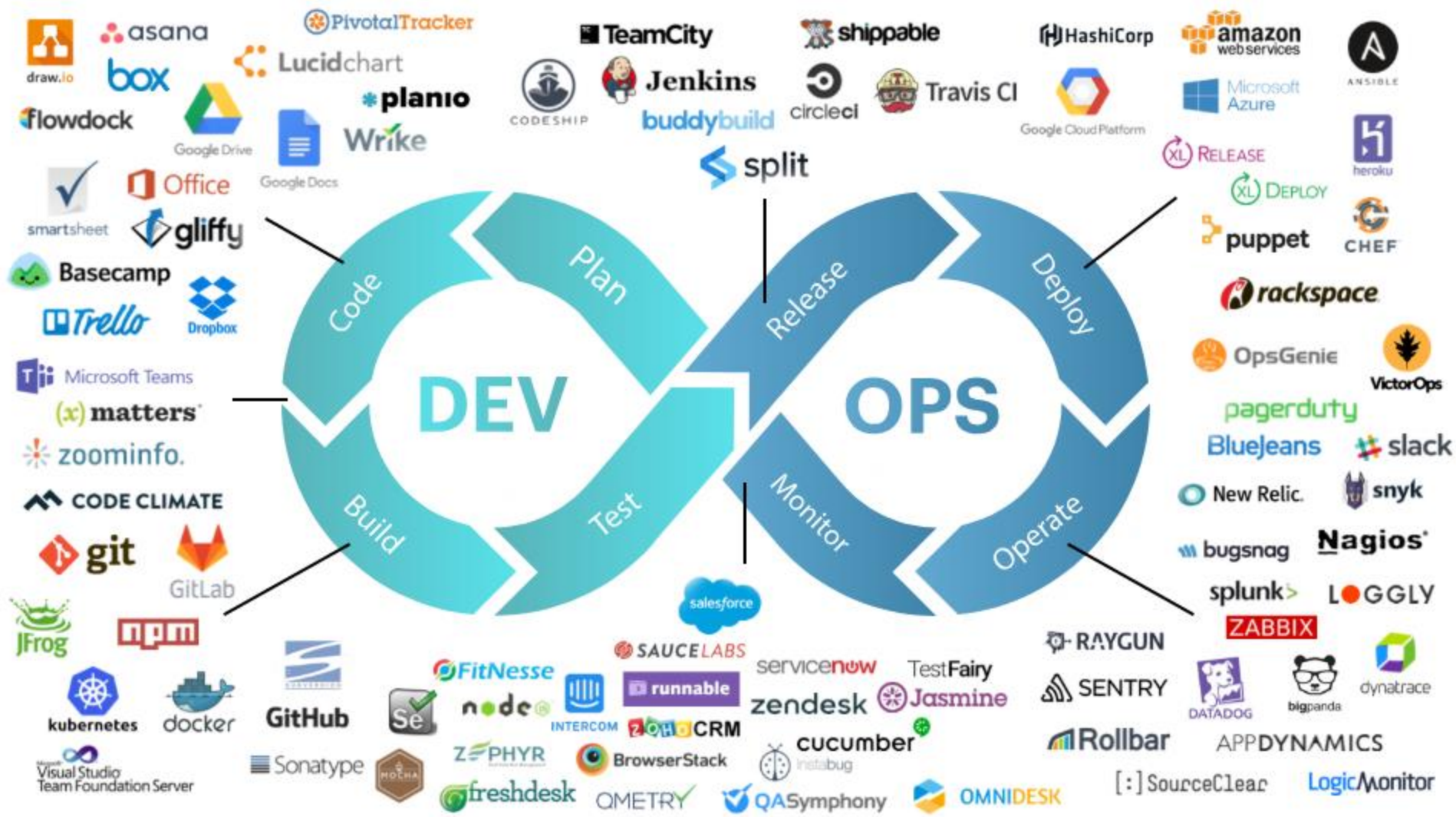
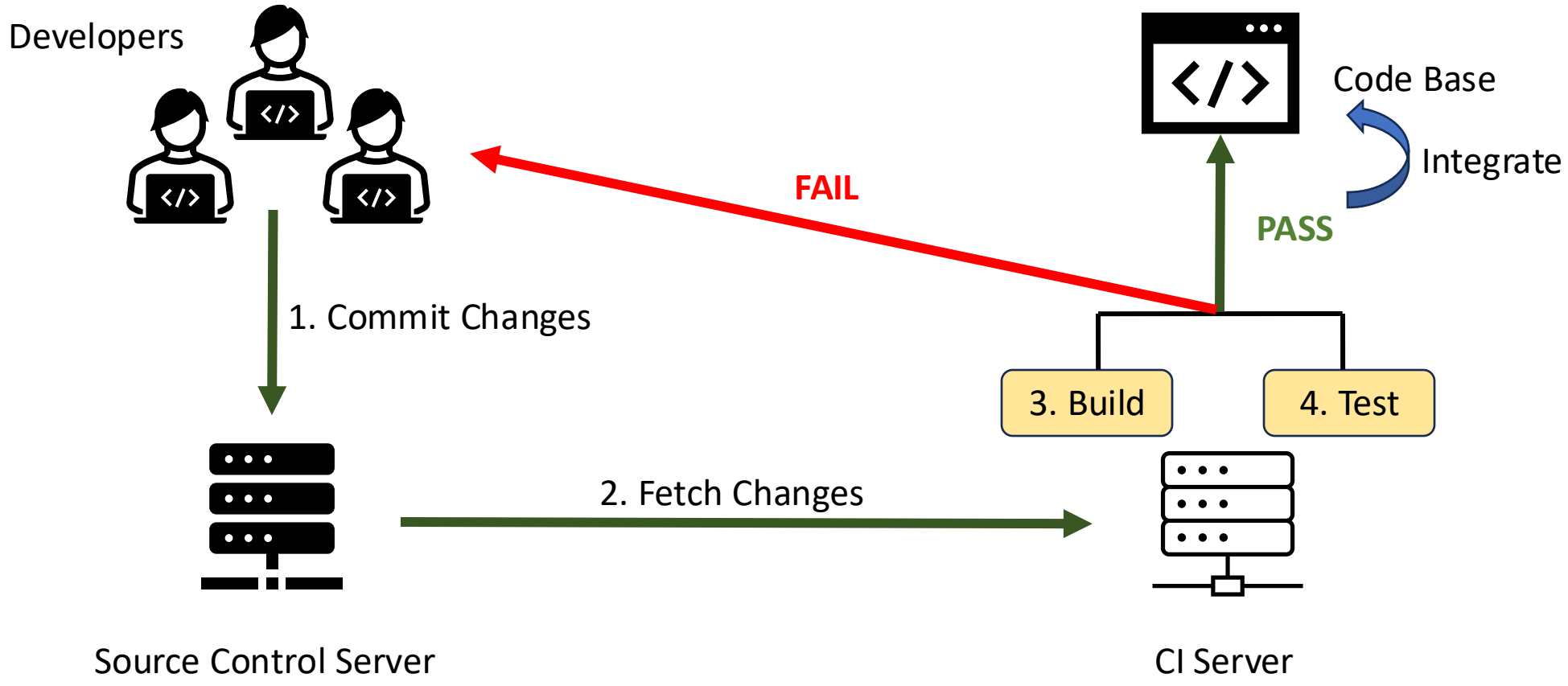


Image source: https://www.openxcell.com/devops

Continuous
Integration/Delivery/Deployment

Continuous Integration (CI)



Benefits of CI/CD

Faster Development Cycles:
Rapid integration and deployment lead to quicker releases.
Early Bug Detection:
Automated testing catches bugs early in the development process.
Consistent and Reliable Builds:
Ensures consistency across different environments.
Reduced Manual Intervention:
Automation minimizes the need for manual processes.

Continuous Integration/Deployment

Continuous Integration (CI):

- The practice of regularly merging code changes into a shared repository.
- Automated builds and tests to detect and fix integration issues early.

Continuous Deployment (CD):

- The automatic deployment of code changes to production or staging environments.
- Ensures a streamlined and efficient release process.

CI/CD Pipeline

Definition: A series of automated steps that code changes go through from development to deployment.

Stages:

- Code Compilation
- Automated Testing
- Artifact Generation
- Deployment to Staging
- User Acceptance Testing (UAT)
- Deployment to Production

Key Components of CI/CD

Version Control System (VCS):
e.g., Git, SVN - Enables collaborative development.

Build Automation Tools:
e.g., Jenkins, Circle CI - Automates building and packaging.

Automated Testing:
Unit tests, integration tests, and end-to-end tests.

Artifact Repository:
e.g., Nexus, Artifactory - Stores and manages build artifacts.

Continuous Integration

- **Purpose:** Ensure that code changes made by developers are integrated into the main codebase regularly and automatically.
- **Process:** Developers frequently merge their code changes into a shared repository, triggering an automated build and test process.
- **Benefits:** Early detection of integration problems, reduced integration risk, faster feedback on code changes, and increased collaboration among team members.

Continuous Deployment

- **Variation of Continuous Delivery:** Continuous Deployment is an even more automated extension of Continuous Delivery where every successful build that passes automated tests is automatically deployed to production without manual intervention.

- **Benefits:** Enables the most rapid and frequent release cycles, reducing time between development and delivery. **However, it requires a high level of confidence in automated testing and deployment processes**.

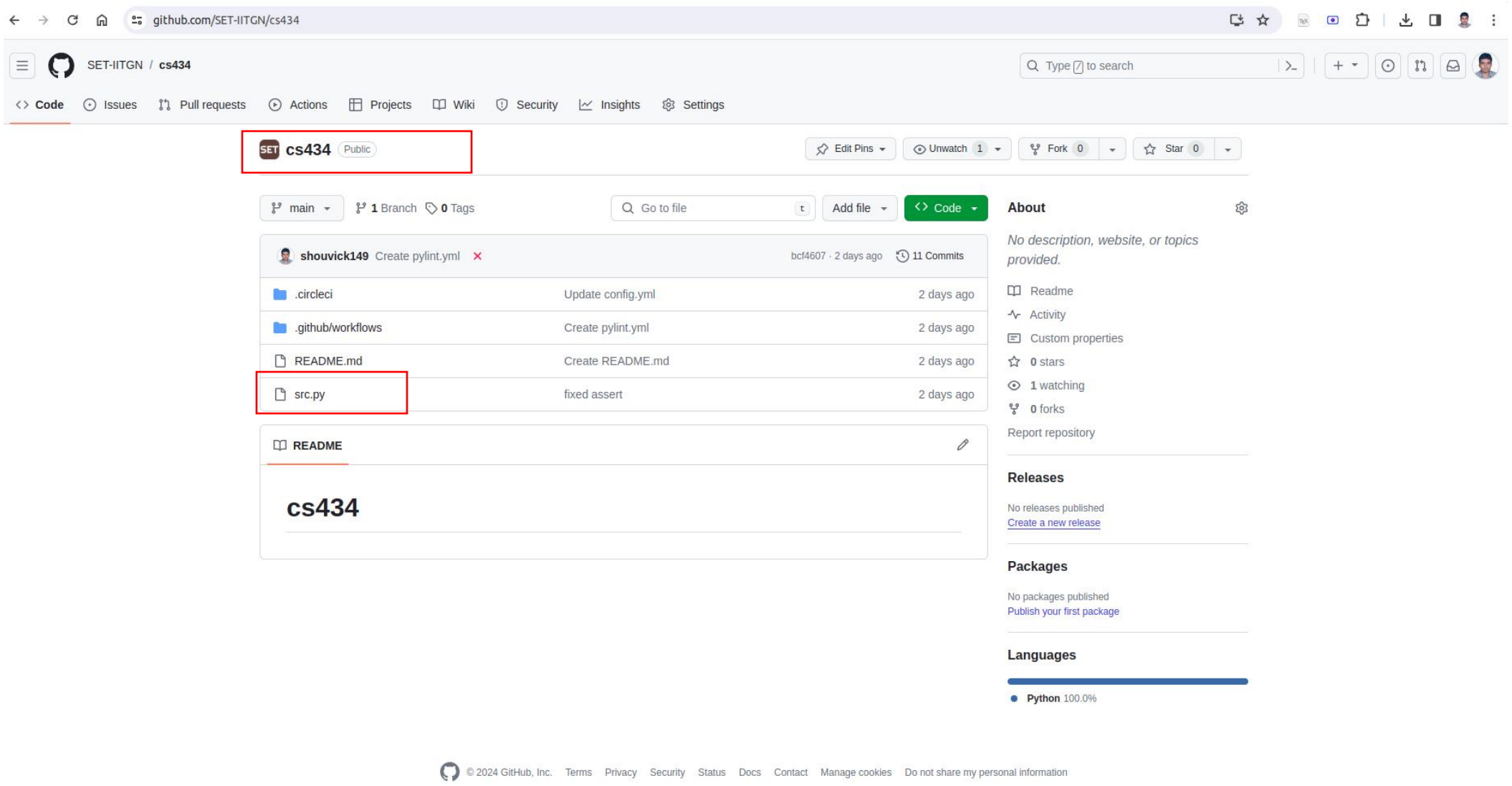
Continuous Delivery

- **Purpose:** Extend CI principles to ensure that the software can be released to production at any time by automating the entire software release process up to the point of deployment.

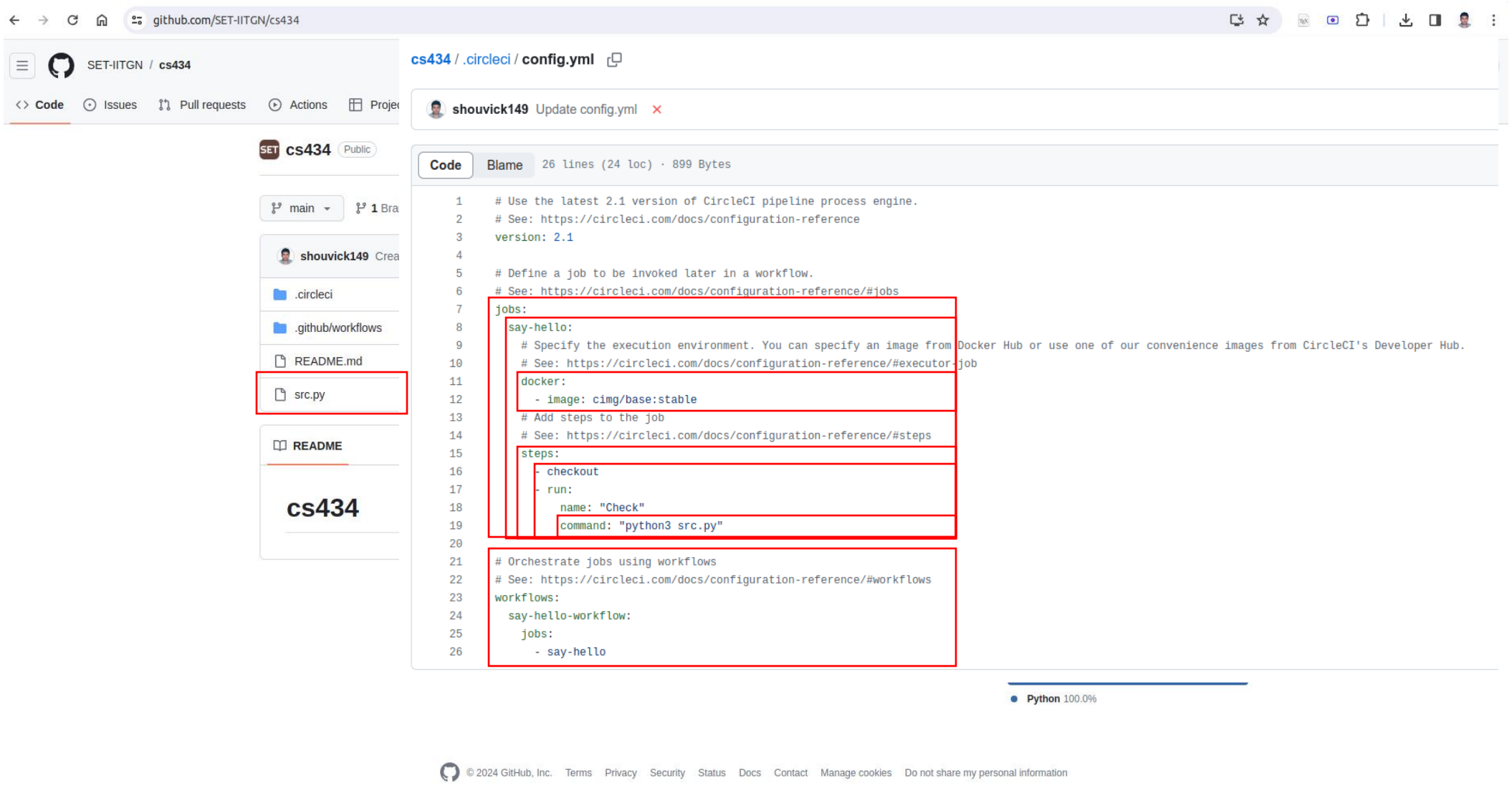
- **Process:** After successful CI, the software undergoes automated testing, and if all tests pass, it can be automatically deployed to a staging environment. It can then be manually or automatically promoted to production.

- **Benefits:** Reliable and repeatable software releases, faster time to market, reduced manual intervention in the release process, and the ability to release new features more frequently.

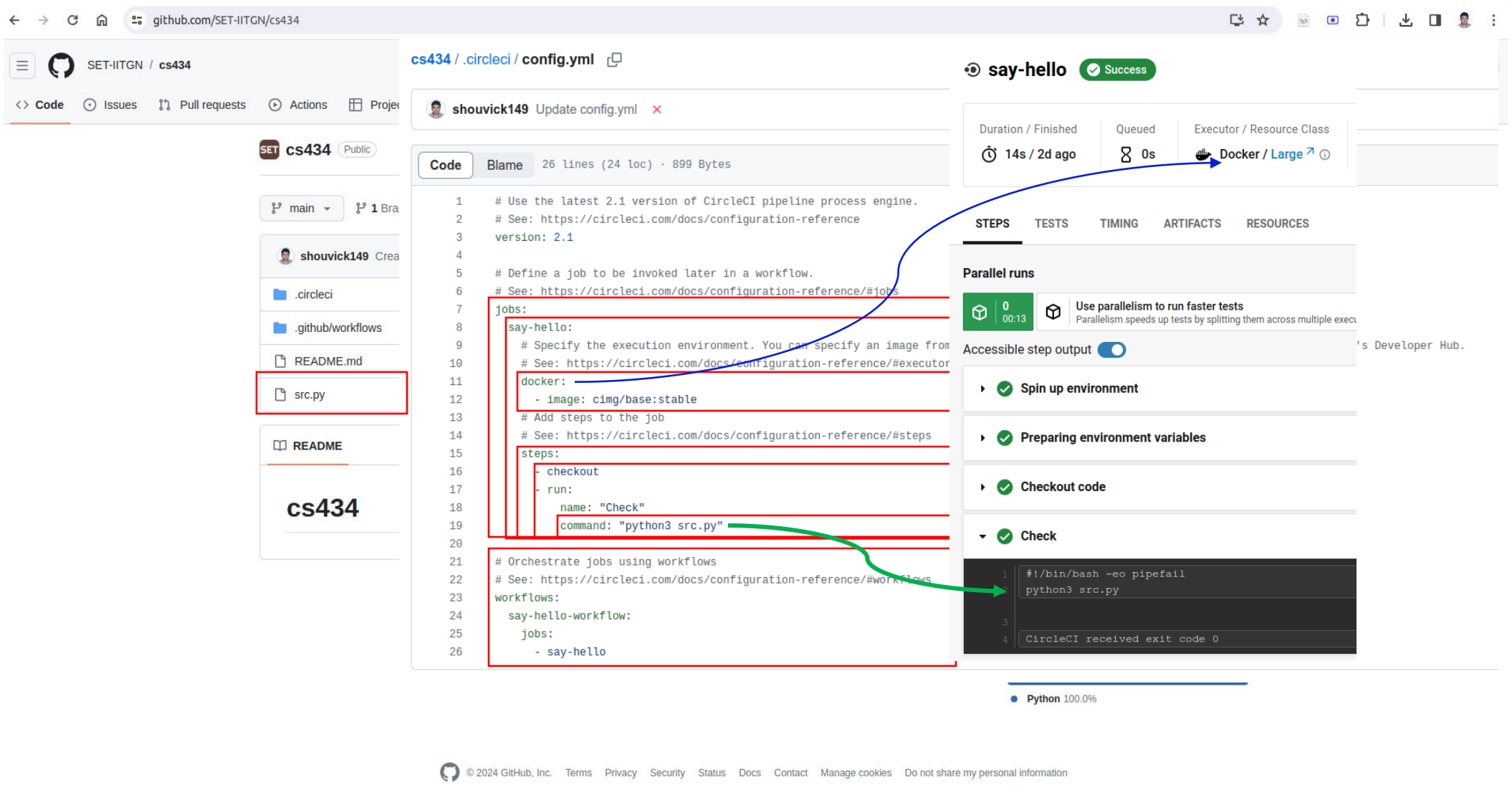
Setting up CI for a GitHub project (cs434)



Specify CI workflows/jobs in YAML (`config.yml`)



Specify CI workflows/jobs in YAML (`config.yml`)



Texts, References, and Acknowledgements

Online:

- Continuous Integration and Delivery (**CircleCI**: https://circleci.com)

Textbook:

- Sharp, J. (2022). *Microsoft Visual C# Step by Step*, 10th edition, Microsoft Press.
- Watson, K., Nagel, C., Pedersen, J. H., Reid, J. D., & Skinner, M. (2008). *Beginning Microsoft Visual C# 2008*. John Wiley & Sons.
- Mark J. Price (2024). *C# 13 and .NET 9 – Modern Cross-Platform Development Fundamentals*, 9th edition, Packt Publishing Ltd.

Reference:

- Soni, M. (2016). *DevOps for Web Development*. Packt Publishing Ltd.
- Yusuf Sulistyo Nugroho, Hideaki Hata, and Kenichi Matsumoto. 2020. *How different are different diff algorithms in Git? Use* `--histogram` *for code changes*. Empirical Softw. Engg. 25, 1 (Jan 2020), 790–823.