

DevOps

Antonio Brogi

Department of Computer Science
University of Pisa



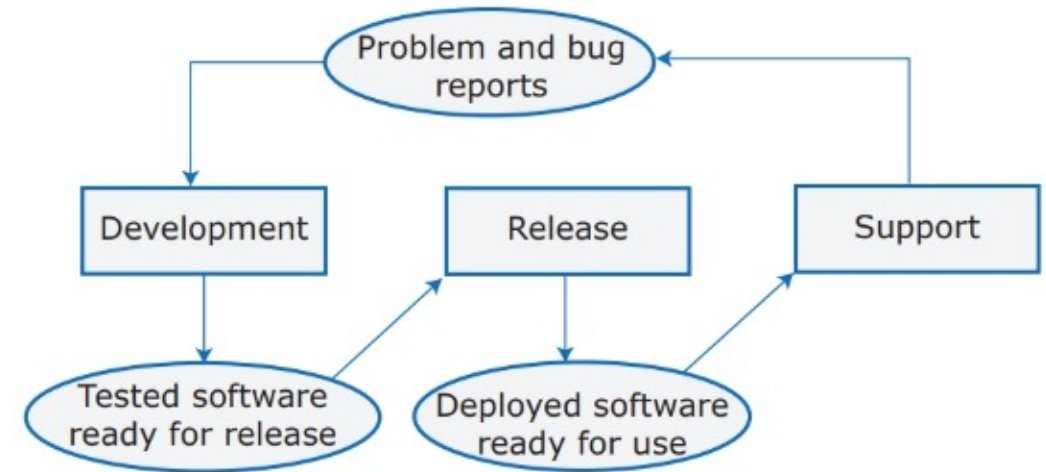
DevOps?

Development | Deployment | Support

Separate teams responsible for software development, release, and support

- development team passes “final” version of software to release team
- release team builds release version, tests it, prepares releases documentation, and releases software to customers
- support team provides customer support

Development team (or separate maintenance team) responsible for implementing software changes



Development | Deployment | Support

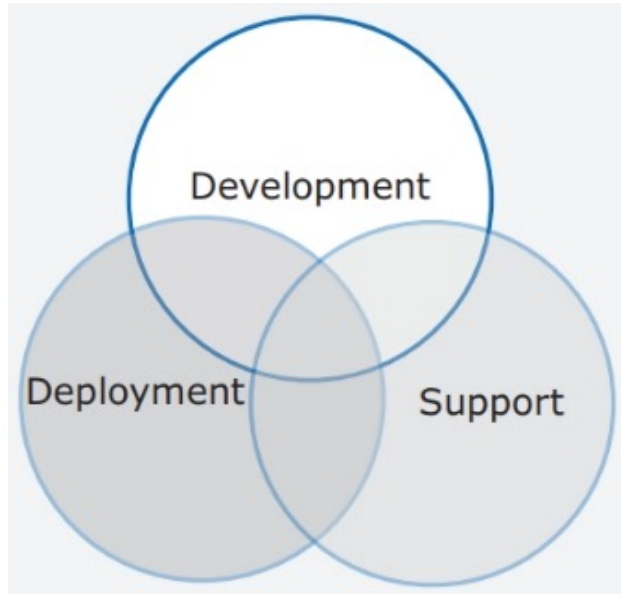
Issues

- Communication delays between teams
- Separate teams use different tools, have different skills, often don't understand the other's problems
- Days needed to fix urgent bugs or security vulnerabilities

DevOps approach

Three factors enabling a change:

1. Agile software engineering **reduced software development time**
→ traditional release process became a bottleneck
2. **Amazon** re-engineered its software into (micro)services, **assigning** both **service development and service support to same team**
3. **SaaS release of software** became possible on public/private clouds



DevOps (**Dev**elopment + **Ops**erations) integrates development + deployment + support in a single team

DevOps principles

Everyone is responsible for everything

All team members have joint responsibility for developing, delivering, and supporting the software

Everything that can be automated should be automated

All/most activities involved in testing, deployment, and support should be automated

Measure first, change later

DevOps should be driven by measuring collected data about the system and its operation

DevOps benefits

Faster deployment

Dramatic reduction of human communication delays → faster deployment to production

Reduced risk

Small functionality increment in each release → less chance of feature interactions and system failures/outages

Faster repair

No need to discover which team is responsible for fixing problem and to wait for them to fix it

More productive teams

DevOps teams more productive than teams involved in separate activities

DevOps culture

Creating a DevOps team means bringing together a number of different skill sets, including

- software engineering
- UX design
- security engineering
- infrastructure engineering
- customer interaction

Successful DevOps team

- culture of mutual respect and sharing
 - everyone on the team should be involved in Scrums and other team meetings
 - team members encouraged to share their expertise with others and to learn new skills
- developers should support the software services that they have developed
 - if service fails in weekend, developer is responsible for getting it up and running again
 - if developer is unavailable, other team members take over the task
 - team priority: ~~blame team member(s)~~ fix failures as quickly as possible

DevOps?
Code management

Code management

During development of a software product

- tens of thousands of lines of code and automated tests created, organized into hundreds of files
- dozens of libraries used, different programs to create/run code

→ Impossible to keep track of changes made to software without an automated support

Code management = software-supported practices to manage evolving codebase

- to ensure that changes made by different developers do not interfere with each other and to create different product versions
- to simplify creating executable product from source code files and to run automated tests

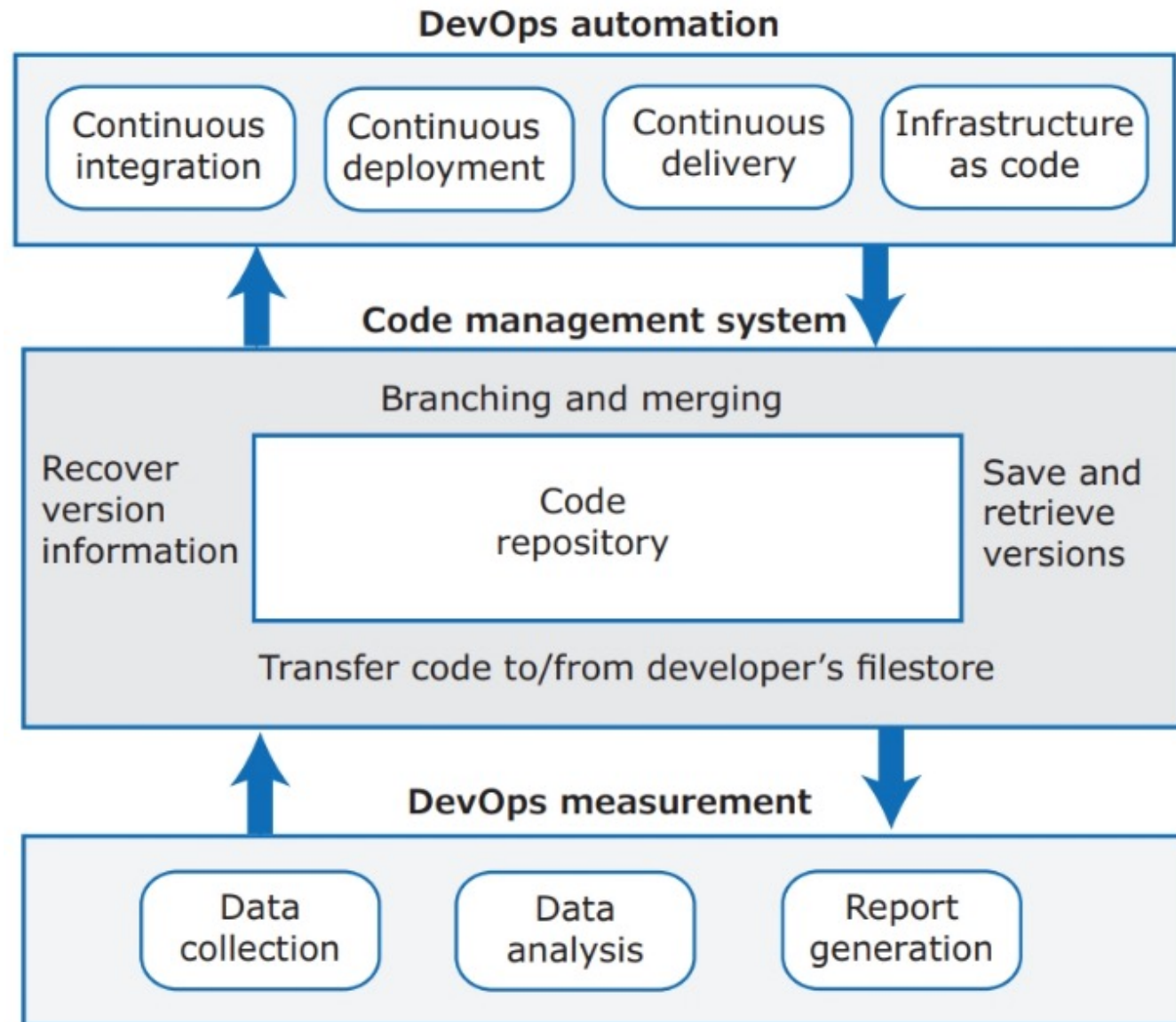
Motivation: example

Alice and Bob worked for a company called FinanceMadeSimple and were team members involved in developing a personal finance product. Alice discovered a bug in a module called TaxReturnPreparation. The bug was that a tax return was reported as filed but sometimes it was not actually sent to the tax office. She edited the module to fix the bug. Bob was working on the user interface for the system and was also working on TaxReturnPreparation. Unfortunately, he took a copy before Alice had fixed the bug and, after making his changes, he saved the module. This overwrote Alice's changes, but she was not aware of this.

The product tests did not reveal the bug, as it was an intermittent failure that depended on the sections of the tax return form that had been completed. The product was launched with the bug. For most users, everything worked OK. However, for a small number of users, their tax returns were not filed and they were fined by the revenue service. The subsequent investigation showed the software company was negligent. This was widely publicized and, as well as a fine from the tax authorities, users lost confidence in the software. Many switched to a rival product. FinanceMadeSimple failed and both Bob and Alice lost their jobs.

- A code management would have detected conflict between Bob's and Alice's changes
- bug would have been fixed
- company could have continued in business

Code management needed for DevOps automation and measurement



Source code management systems

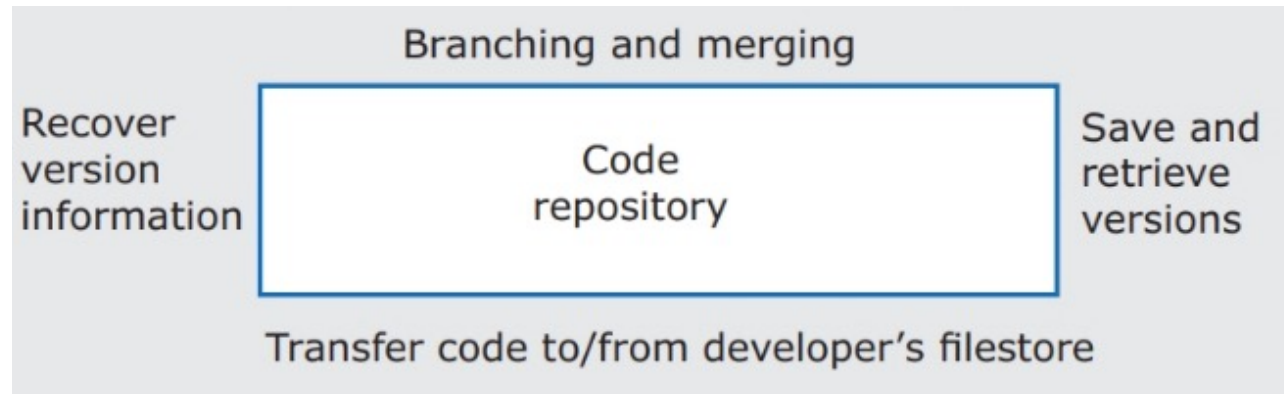
Objective

- manage evolving project codebase to allow different versions of components and entire systems to be stored and retrieved
- developers can work in parallel without interfering with each other and integrate their work with that from other developers

By featuring

- Code transfer
Developers download code into personal file store, work on it, return it to shared code management system
- Version storage and retrieval
Files may be stored in several different versions, specific versions can be retrieved
- Branching and merging
Parallel development branches can be created for concurrent working
Changes made in different branches may be merged
- Version information
Information on different versions may be stored and retrieved

Source code management systems



Shared repository

- All source code files and file versions are stored in the repository, together with other artifacts (e.g. configuration files, build scripts, shared libraries, versions of tools)
- The repository includes a database of information about the stored files (e.g. version information, author of changes, time of changes)

Files can be transferred to/from the repository, information about different versions of files and their relationships can be updated

- Specific versions of files and information about these versions can always be retrieved from the repository

Features of code management systems

Version and release identification

Managed versions of a code file are uniquely identified when submitted to the system
(→ managed files can never be overwritten)

They can be retrieved using their identifier and other file attributes

Change history recording

When a change to a code file is submitted, submitter must add a string explaining the reasons of the change

(→ helps developers understand why new version was created)

Independent development

Several developers can work on the same code file at the same time

When this is submitted to the code management system, a new version is created so that files are never overwritten by later changes.

(→ avoids the file overwriting problem of previous example)

Project support

All files associated with a project may be checked out at the same time

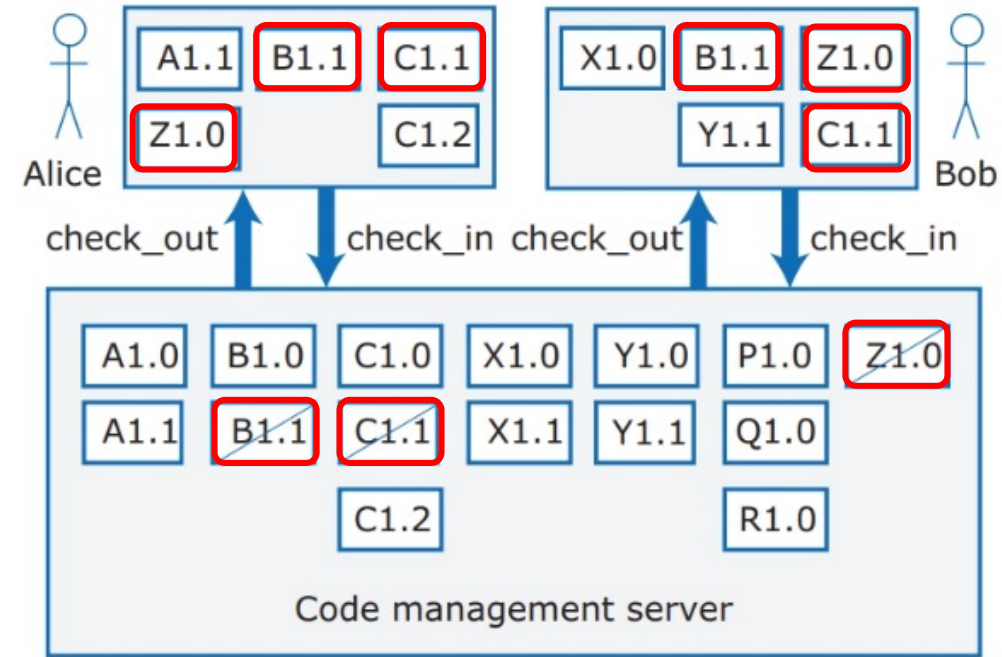
Storage management

Code management system includes efficient storage mechanisms not to keep multiple copies of files that have only small differences (less dramatic today with cheap storage)

Centralized source code management

Centralized repository architecture

- users check in and check out files
- warning when checking out file already in use
- at check in: new version of file is created, system ensures that file copies do not conflict



Still used in some code management systems

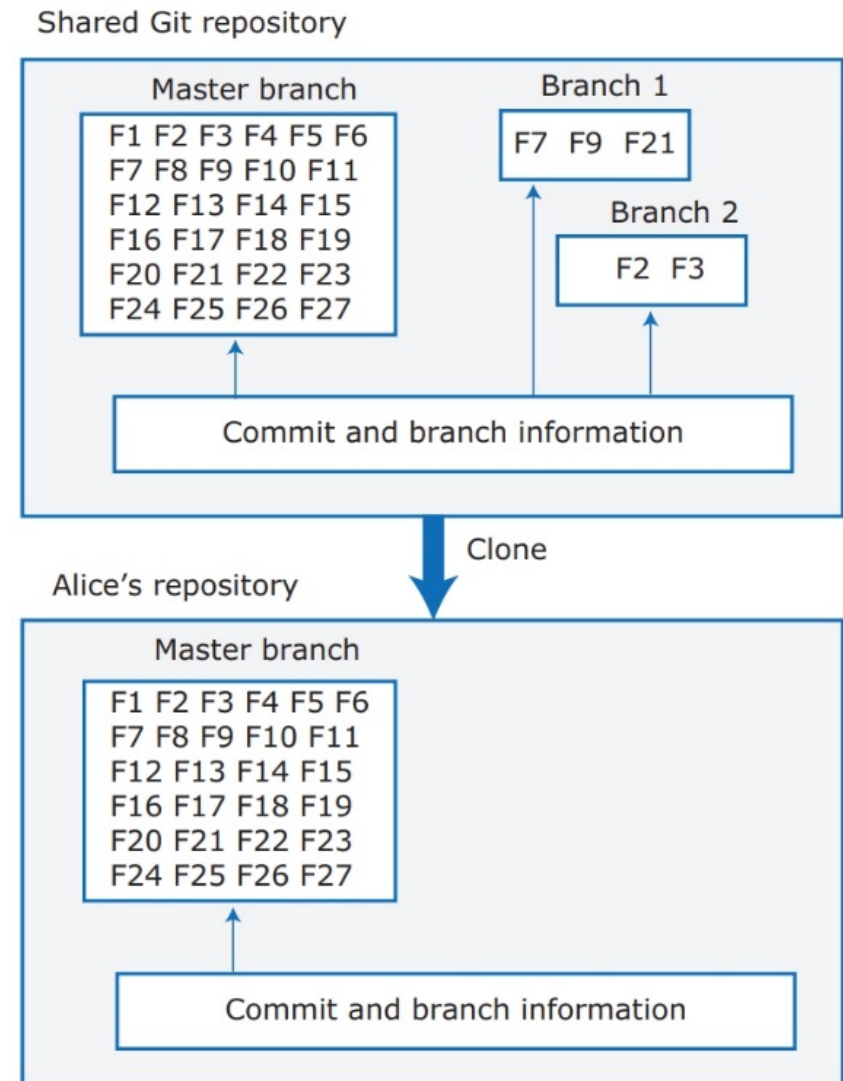
e.g. <https://subversion.apache.org/>

Decentralized source code management

Most commonly used today for software product development

Git (Linux's Distributed Version Control System)

- to support large-scale open-source development (originally to manage Linux kernel's code)
- taking advantage of low storage costs, Git maintains a clone of the repository on every user's computer
- "master branch": current master version of software that the team is working on
- new versions can be created by creating new branches
- when users request a repository clone, they get a copy of the master branch that they can work on independently



Advantages of *decentralized* systems

Resilience

- Everyone working on a project has own copy of repository
- If shared repository damaged/attacked, work can continue, clones can be used to restore shared repository
- People can work offline

Speed

- Committing changes to the repository is a fast, local operation
- It does not need data to be transferred over the network

Flexibility

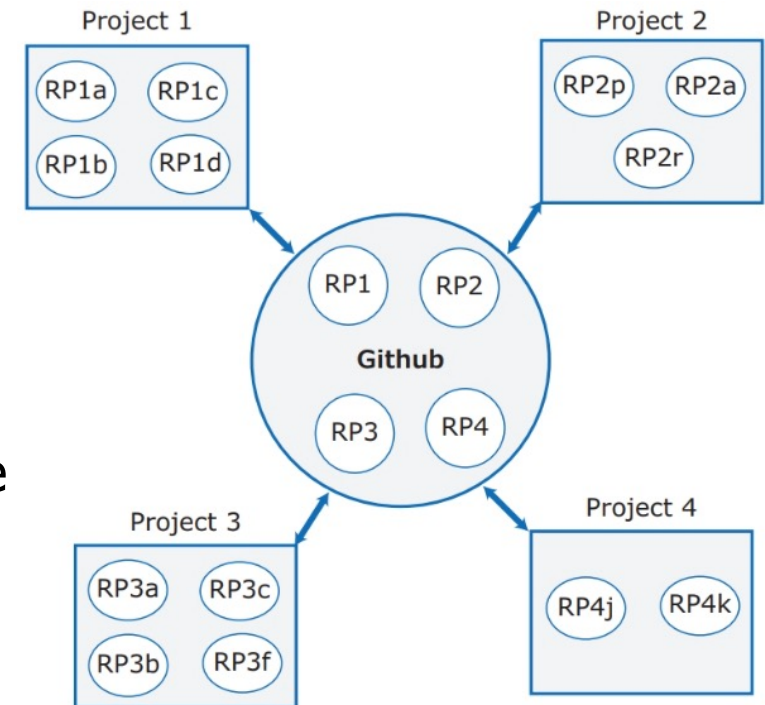
- Local experimentation is much simpler
- Developers can safely try different approaches without exposing experiments to others

Git for teamwork

- Shared project repository (running in server's company or in the cloud by hosting company e.g. Github, Gitlab)
- Private clones of shared repository held on each developer's computer

Example

- 4 project repositories on Github (RP1–RP4)
- Each developer on each project identified by a letter
- Each developer has individual copy of project repository
- Each developer may work on more than one project at a time



Using Git

When you join a project, you set up a project directory on your computer that acts as your workspace for that project

```
cd myproject
```

The command

```
git init
```

sets up a local Git repository (repo) in project directory “myproject”, and

```
git clone <URL of external repository>
```

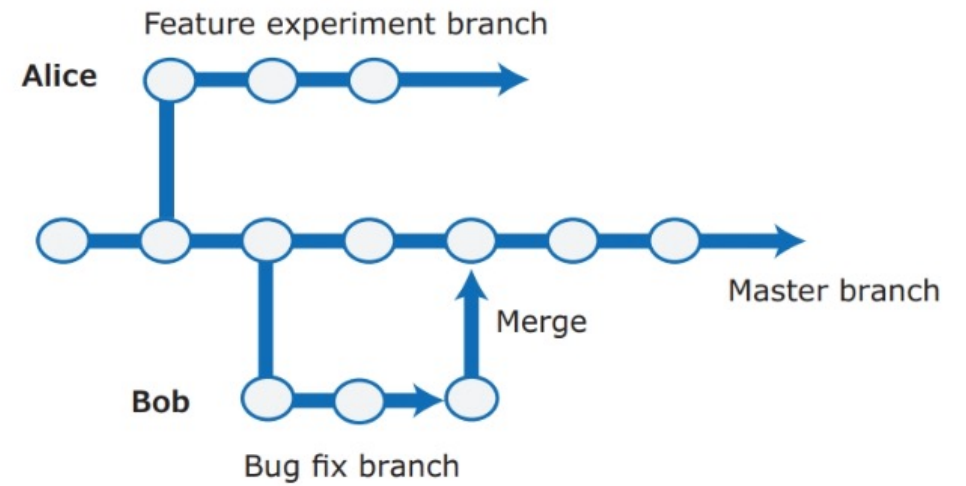
copies master files (usually most recent version of project files) from remote repository to working directory. It also copies repository information so that you can see what other developers have done - you can query this information and download other branches from the project repo if you need them.

You can then start to work on the files in your project directory, adding files and committing changes to local repo

```
git add <list of files to be controlled>
```

```
git commit
```

Branching



A branch is an independent, stand-alone version that is created when a developer wishes to change a file

- The changes made by developers in their own branches may be merged to create a new shared branch
- The repository ensures that branch files that have been changed cannot overwrite repository files without a merge operation
- If a developer makes mistakes on the branch she is working on, she can easily revert to master file
- If she commits changes while working, she can revert to earlier versions of the work she has done

Git (efficiently) compares versions of files on a line-by-line basis

- There is no conflict if developers have changed different lines in the file.
- If they have made changes to the same lines, however, then a merge conflict is signalled. Git highlights the lines where the conflict occurs, and it is up to the developers to resolve these conflicts.

Using Git

The command

```
git checkout -b fix-header-format
```

creates new branch (called fixed-header-format) based on the files in your working directory

You can now for instance edit two files, add them and commit

```
git add report_printer.py generate_header.py
```

```
git commit
```

The changes are committed to current branch (fix-header-format)

If you want to merge your changes with with the master branch in your directory, with

```
git checkout master
```

you check out the master branch from the project repo (to make sure that you have most recent version of all master files), and with

```
git merge fix-header-format
```

issue a merge command.

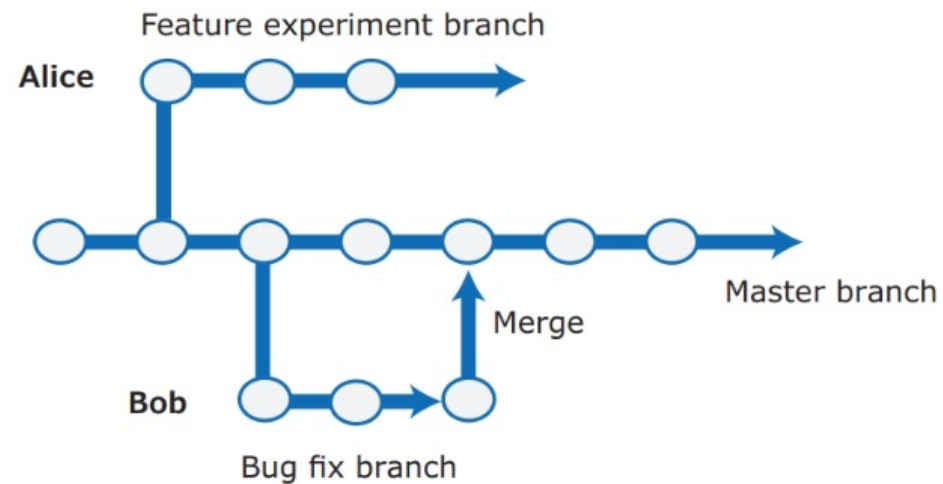
So far, you have updated the master branch in your local repository but not the external repository. With

```
git push
```

you incorporate the changes that you have made in the master branch of the external repository

Branching (example)

- Alice and Bob are working on the same file in the Git repo
- Each created a branch for working on a copy of that file
- Bob finished and tested his bug fix, he **merges** his changes and **pushes** them to the shared repo
- Bob's update is accepted
- Alice finishes her feature experiment and tries to **push** her changes
- Git rejects them (as Bob already issued a **push** command with possible conflicts)
- Alice **pulls** Bob's changes to update her repository, fixes the conflict, and **pushes** the updated branch to the external repo



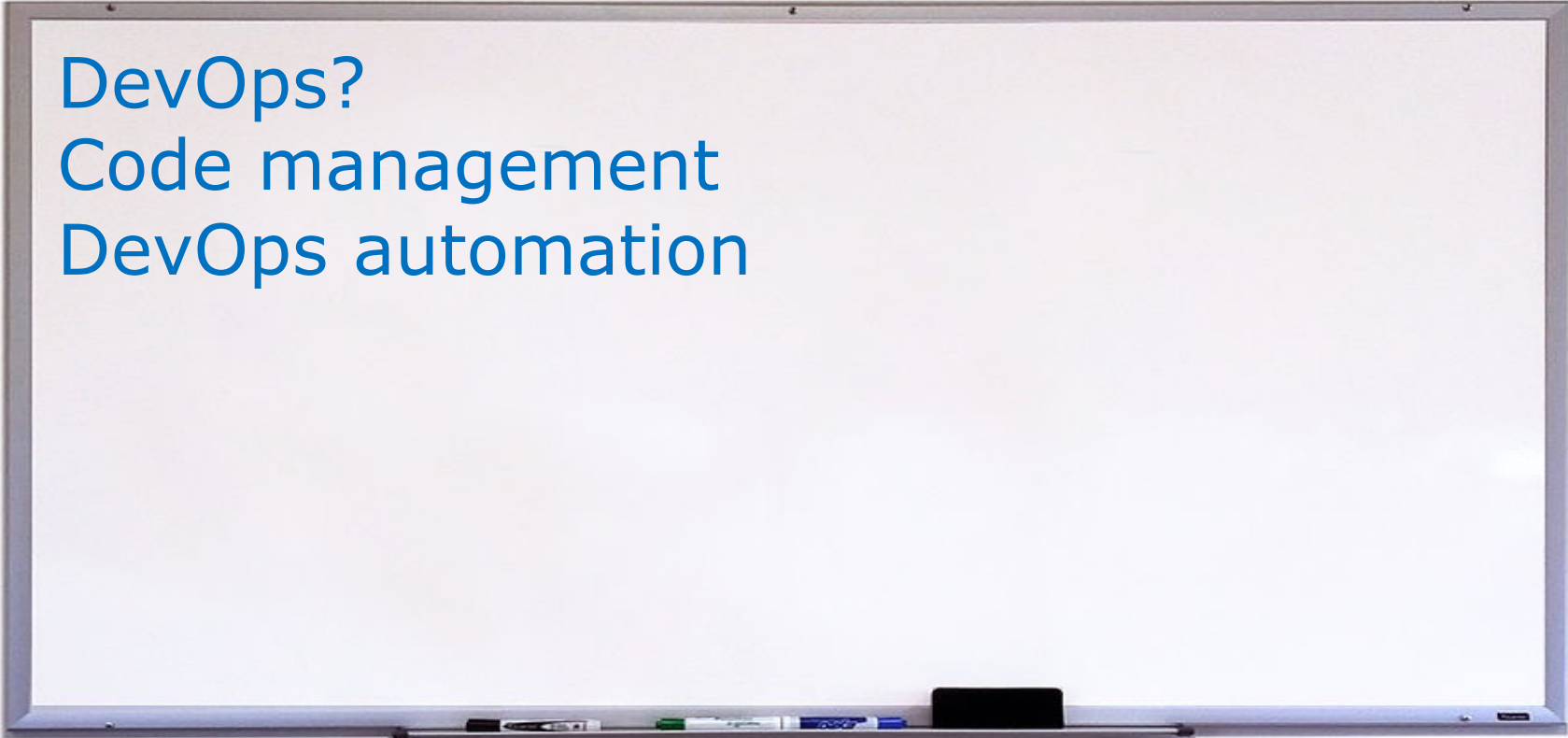
Open-source software development with Git

Different people can work independently on code without knowledge of what others are doing

Master version of the open-source software is managed by individual/group that decides what changes should be incorporated

Github uses a mechanism called Webhooks to trigger DevOps automation tools in response to updates to the project repository



A whiteboard with a silver frame and a white surface. The text 'DevOps?', 'Code management', and 'DevOps automation' is written in blue marker on the top left. At the bottom, there are several markers (black, green, blue) and a black eraser.

DevOps?
Code management
DevOps automation

“Everything that can be should be automated”

Continuous integration

Each time a developer commits a change to the project's master branch, an executable version of the system is built and tested.

Continuous delivery

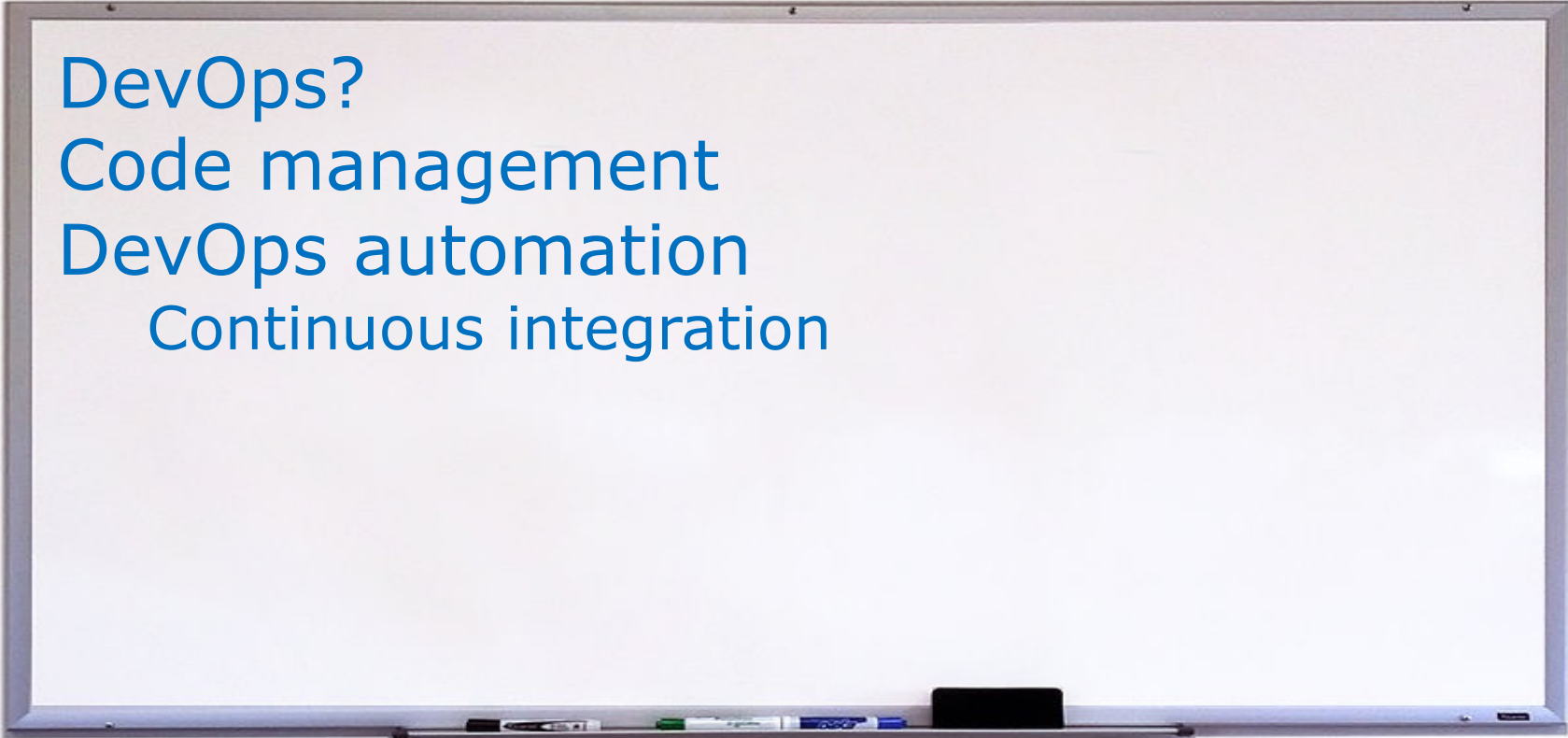
Executable software is tested in a simulated product's operating environment.

Continuous deployment

New release of system made available to users every time a change is made to the master branch of the software.

Infrastructure as code

Machine-readable models of infrastructure (network, servers, routers, etc.) on which the product executes are used by configuration management tools to build the software's execution platform. The software to be installed, such as compilers and libraries and a DBMS, are included in the infrastructure model.

A whiteboard with a silver frame and a white surface. On the left side, four lines of text are written in blue marker. At the bottom of the whiteboard, there is a black eraser and several markers (black, green, blue, and white) lying horizontally.

DevOps?
Code management
DevOps automation
Continuous integration

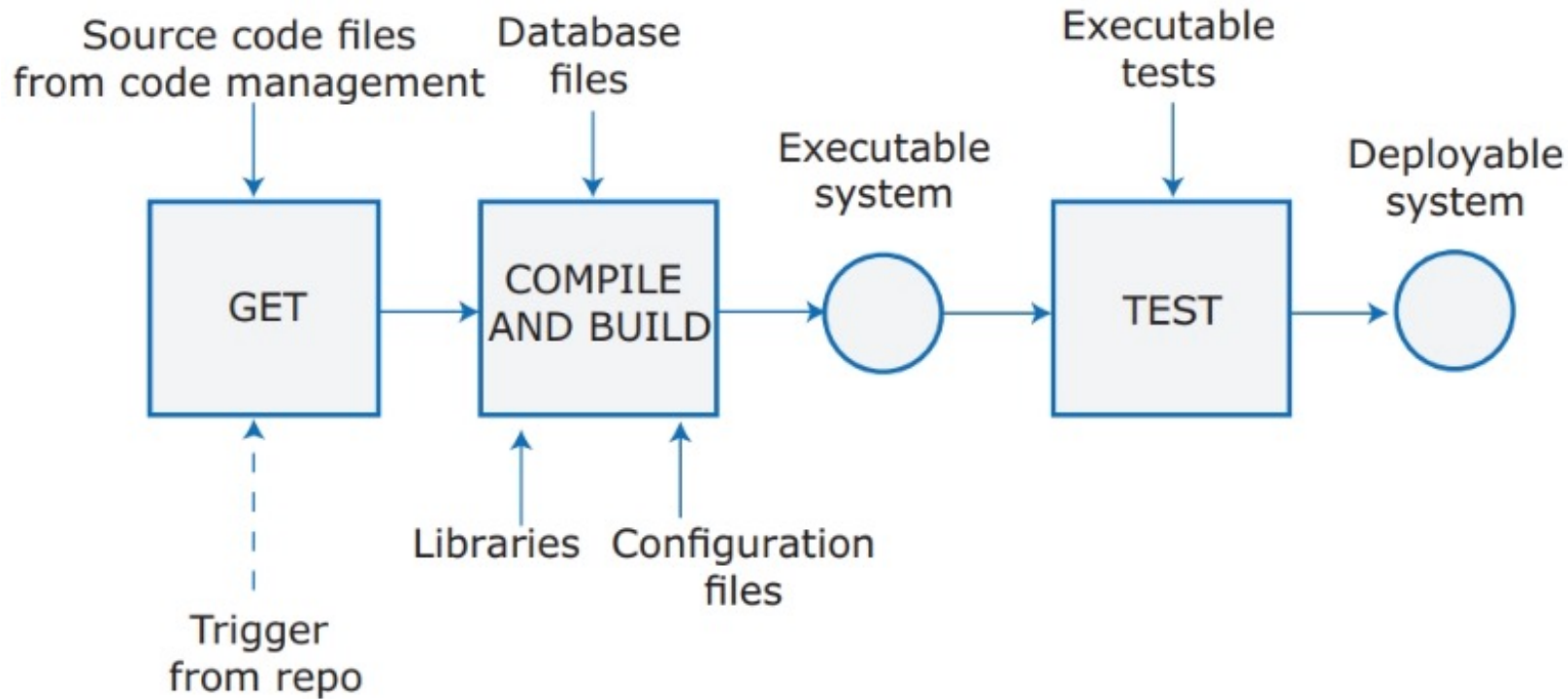
Continuous Integration

System integration (system building) is more than compiling:

- install db software and set up db with appropriate schema
- load test data into db
- compile the files that make up the product
- link compiled code with libraries and other components used
- check that external services used are operational
- move configuration files to correct locations (and delete old ones)
- run system tests to check that integration has been successful

If a system is *infrequently* integrated, problems can be difficult to isolate, and fixing them slows down system development → continuous integration should be used (Extreme Programming)

Continuous Integration



An integrated version of the system is created and tested every time a change is pushed to the system's shared code repository. On completion of the push operation, repository sends message to integration server to build a new version of the product.



Don't "break the build"





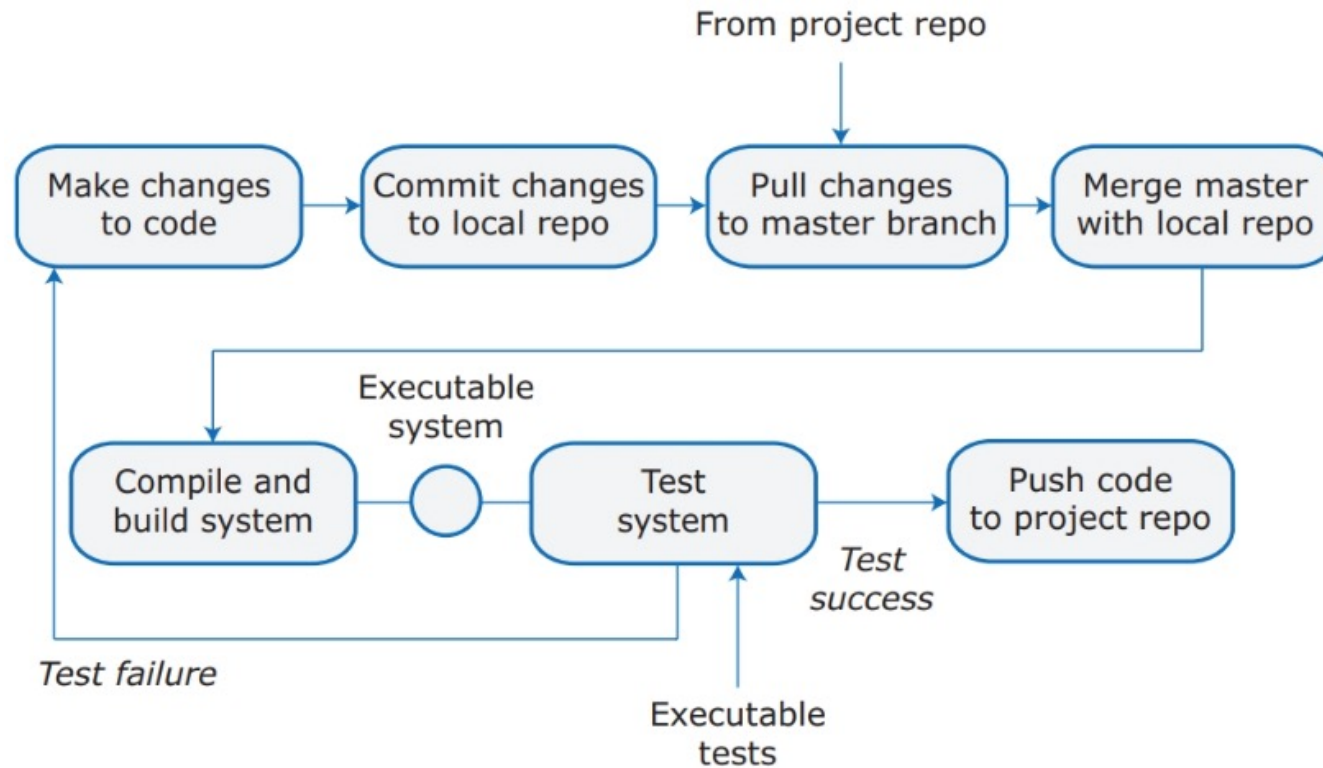
Don't "break the build"



Adopt an "integrate twice" approach to system integration

1- integrate and test on your own computer

2- push code to project repository to trigger integration server



Advantage of Continuous Integration

It is faster to find and fix bugs in the system

if you make a small change and some system test fails, the problem almost certainly lies in the new code that you pushed

A working system is always available to the whole team

It can be used to test ideas and to demo system to management and customers

“Quality culture” in development team

No team members wants the stigma of breaking the build

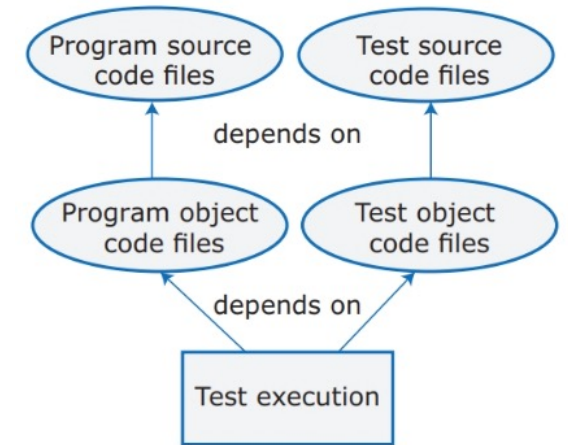
Everybody checks her work carefully before pushing it to project repo

Incremental build process

Code integration tools only repeat actions if dependent files have been changed

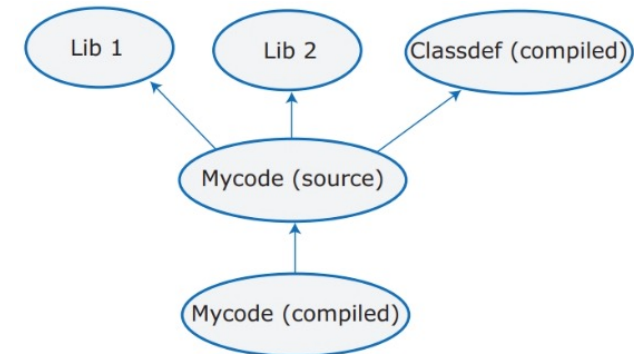
Example

- At first integration, all source code files and executable test files are compiled
- Subsequently, object code files are created only for modified tests and for modified source code files



Example (modification timestamps)

- $\text{date}(\text{compiled code}) > \text{date}(\text{source code}) \rightarrow \text{do nothing}$
- $\text{date}(\text{compiled code}) < \text{date}(\text{source code}) \rightarrow \text{recompile}$
- $\text{date}(\text{compiled code}) < \text{date}(\text{class definition}) \rightarrow \text{recompile}$



Tools

System building tools

Unix make (oldest)

Ant, Maven (Java)

Rake (Ruby)

...

Continuous integration tools

Jenkins (open-source)

Travis, Bamboo (proprietary)

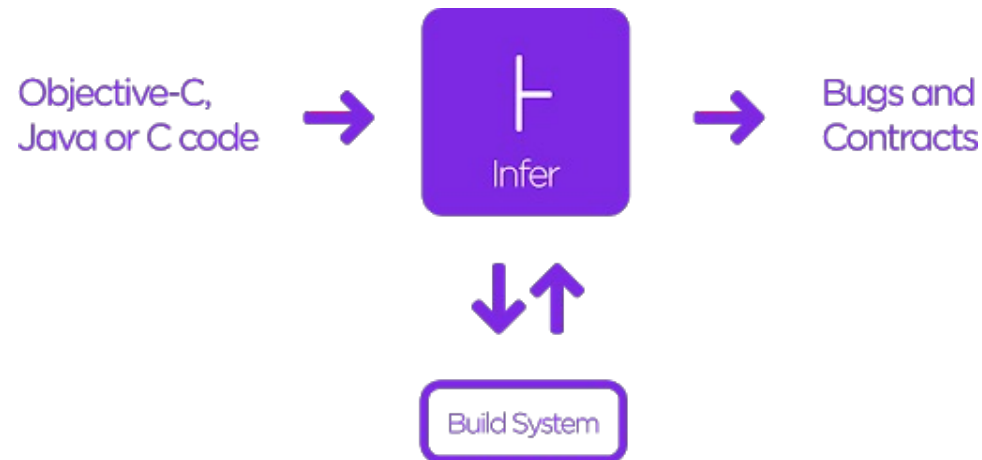
...

Continuous Reasoning

Exploit compositionality to differentially analyse a large-scale system:

- by mainly **focussing on the latest changes** introduced in the system, and
- by **re-using previously computed results** as much as possible

Successful in supporting iterative software development at large IT companies, e.g. [FB Infer](#)



Separation logic



Extension of Hoare logic

$\{precondition\}code\{postcondition\}$

to model in-place update of memory during execution in terms of preconditions and postconditions on the heap

$$\begin{array}{l} \{x \mapsto 0 * y \mapsto 0\} \\ [x] = y; \\ [y] = x \\ \{x \mapsto y * y \mapsto x\} \end{array}$$

(*Concurrent* separation logic for modular reasoning about threads)

DevOps?

Code management

DevOps automation

Continuous integration

Continuous delivery and deployment

Continuous delivery

Continuous Integration creates an executable version of a software system by building system and running tests on your computer or project integration server

Real production environment will differ from development environment

- production server may have different filesystem organization, access permissions, installed applications
- new bugs may show up

Continuous delivery ensures that changed system is ready for delivery to customers

- feature tests in production environment to make sure that environment does not cause system failures
- load tests to check how software behaves as number of users increases

Containers are the simplest way to create a replica of a production environment

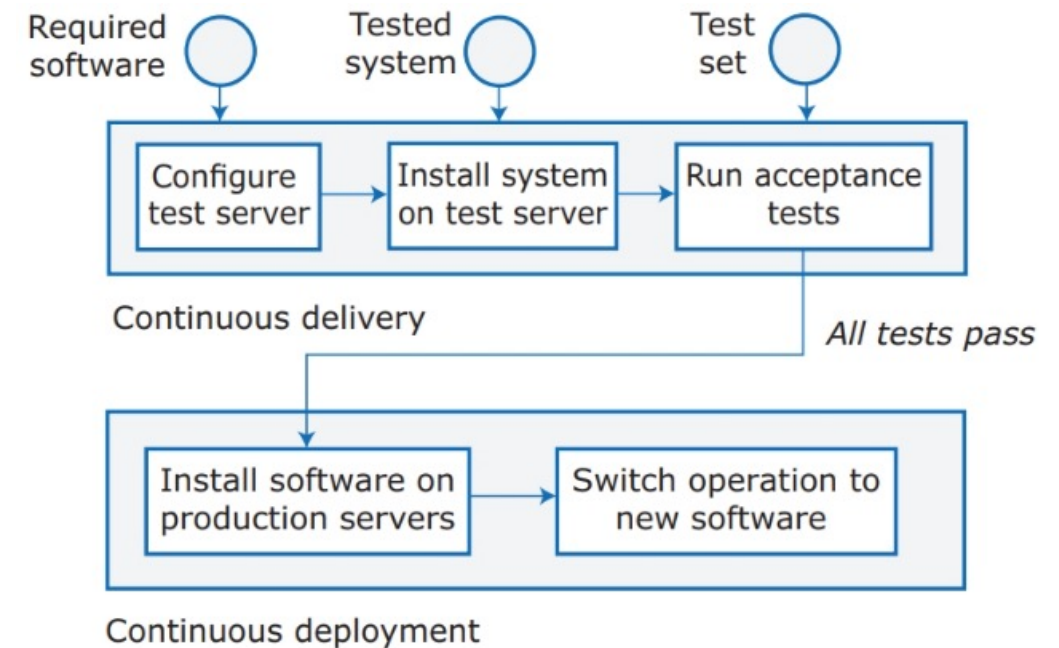
Continuous delivery and deployment

To deliver

- after initial integration testing, configure staged test environment (replica of production environment)
- run system acceptance tests (functionality, load, performance)

To deploy

- software and data are transferred to production servers
- new service requests stopped, older version processes outstanding transactions
- switch to new system version and restart processing



Benefits of continuous deployment

Reduced costs

Fully automated deployment pipeline

Faster problem solving

If a problem occurs, it will probably affect only a small part of the system and the source of that problem will be obvious

Faster customer feedback

You can deploy new features when they are ready for customer use
Users' feedback can be used to identify improvements

A/B testing

If you have many customers and several servers

- deploy new software version only on some servers
- use load balancer to divert some customers to the new version
- measure and assess how new features are used

(Continuous deployment only practical for cloud-based systems)

Why not deploying every single change?

Business reasons not to deploy every software change to customers:

1. You have incomplete features that could be deployed, do not want to disclose this to competitors until implementation is complete
2. Customers may be irritated by continually changing software, especially if this affects UI
3. You may wish to synchronize your releases with known business cycles
e.g. start of academic year for education market

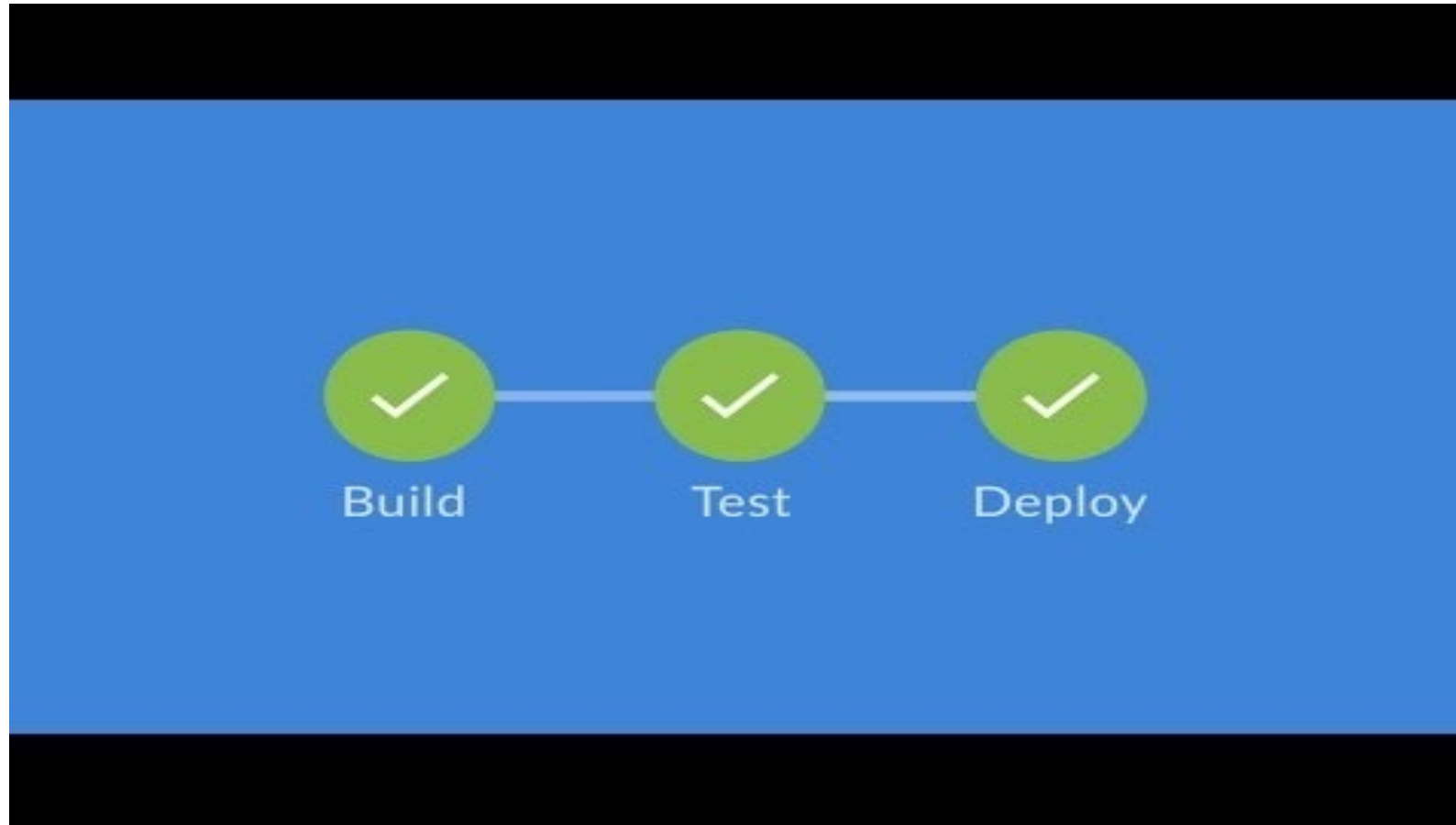
Tools

CI tools such as Jenkins and Travis may also be used to support continuous delivery and deployment.

These tools can integrate with infrastructure configuration management tools such as Chef and Puppet and Ansible to implement software deployment.

For cloud-based software, it is often simpler to use containers in conjunction with CI tools rather than use infrastructure configuration management software.

CI&CD: Jenkins



https://www.youtube.com/watch?v=k_fVIU1FwP4

DevOps?

Code management

DevOps automation

- Continuous integration

- Continuous delivery and deployment

- Infrastructure as code

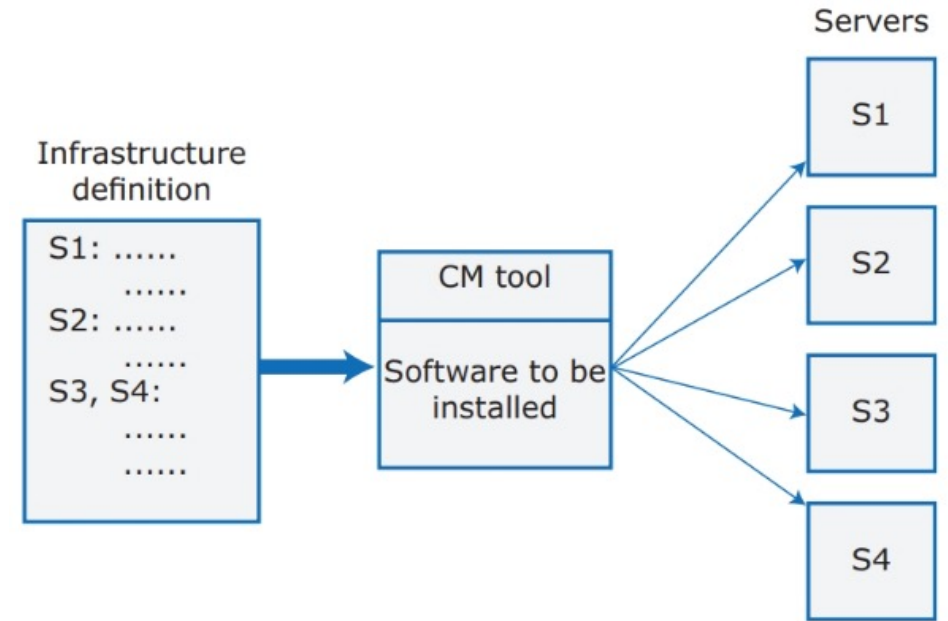
Motivation

Manually maintaining a computing infrastructure with tens or hundreds of servers is expensive and error-prone

- many different physical/virtual servers that have different configurations and run different software packages
- when new versions of software become available, some servers may have to be updated, others may not because of dependencies with older versions of software
- difficult to manually keep track of software installed on each server (e.g. emergency changes are not always documented)

IaC

- Automate the process of updating software on servers, by using a machine readable model of the infrastructure
- Configuration Management (CM) tools, like Puppet and Chef and Ansible, can automatically install software and services on servers according to the infrastructure definition
- When changes have to be made, infrastructure model is updated and CM tool makes the changes to all servers



Advantages of IaC

Visibility

Infrastructure defined as stand-alone model that can be read/understood/reviewed by whole DevOps team.

Reproducibility

Installation tasks will always be performed in the same sequence, same environment will be always created. You do not have to rely on people remembering the order that they need to do things.

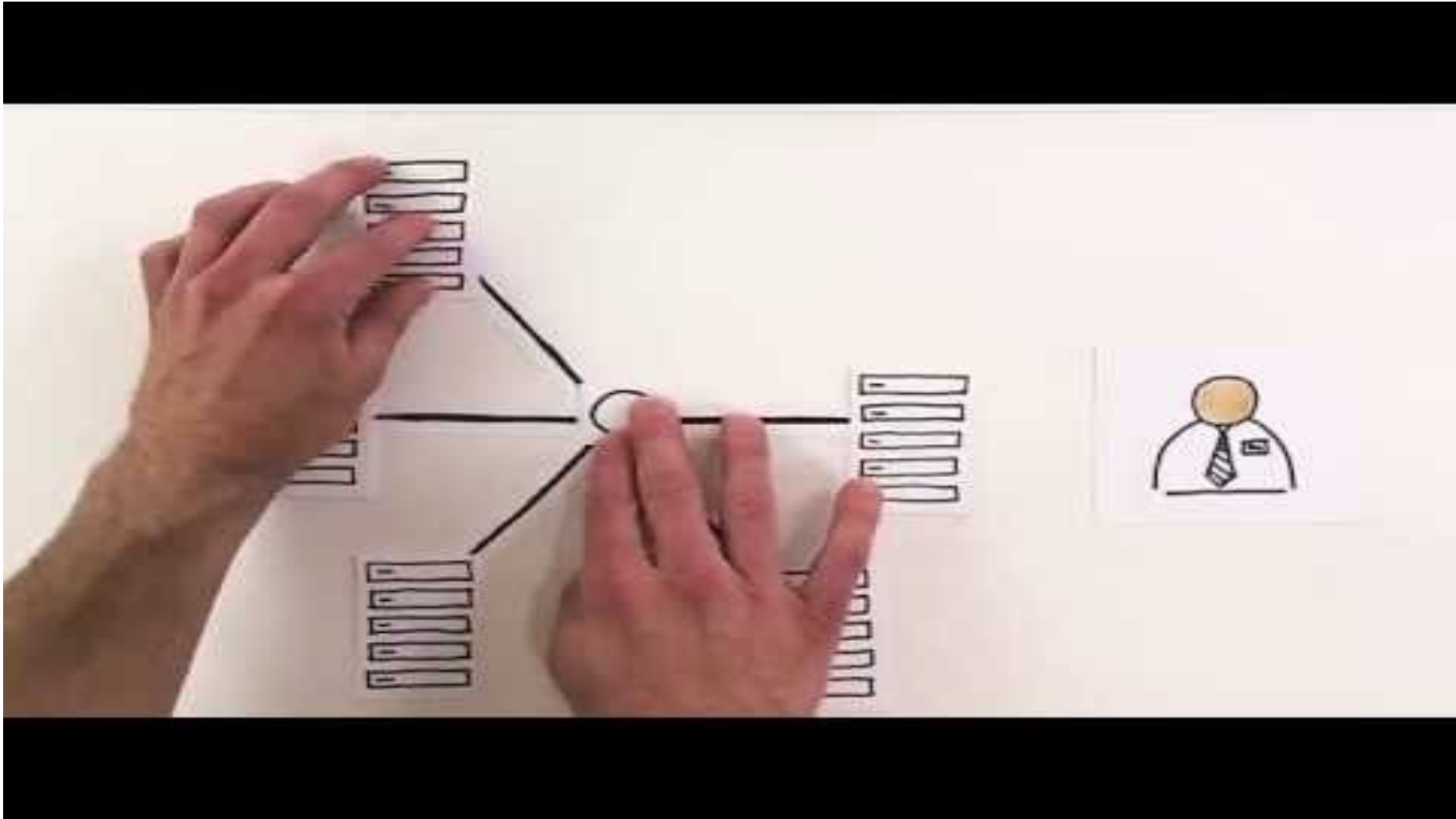
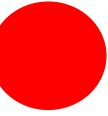
Reliability

Automation avoids simple mistakes made by system administrators when making same changes to several servers.

Recovery

Infrastructure model can be versioned and stored in a code management system. If infrastructure changes cause problems, you can easily revert to older version and reinstall the environment that you know works.

Iac: Puppet



<https://www.youtube.com/watch?v=QFcqvBk1gNA>

Containers

Containers are a very effective way to deploy cloud-based products

- Simple to provide identical execution environments
 - for each type of server you use, you define the environment you need and build an image for execution
 - when you update your software, you create new image that includes the modified software
- You can employ use a container management system, such as Kubernetes, for scaling/resiliency ...

DevOps?

Code management

DevOps automation

- Continuous integration

- Continuous delivery and deployment

- Infrastructure as code

DevOps measurement

DevOps measurement

You should try to continuously improve your DevOps process to achieve faster deployment of better quality software

→

Need to measure and analyse product and process data

- **Process measurements**
to collect & analyze data on development/testing/deployment processes
- **Service measurements**
to collect & analyze data on software's performance/reliability/acceptability to customers
- **Usage measurements**
to collect & analyze data on how customers use your product
(help you identify issues and problems with the software itself)
- **Business success measurements**
to collect & analyze data on how your product contributes to overall business success
(hard to isolate contribution of DevOps to business success - that may be due e.g. to DevOps introduction or to better managers)

DevOps measurement

Measuring software and its development is a complex process

- need to identify suitable metrics that give you useful insights
- need to find reliable ways of collecting and analyzing metrics data
- some measures (e.g. customer satisfaction) must be inferred from other metrics (e.g. number of returning customers)

Software measurement should be automated as far as possible

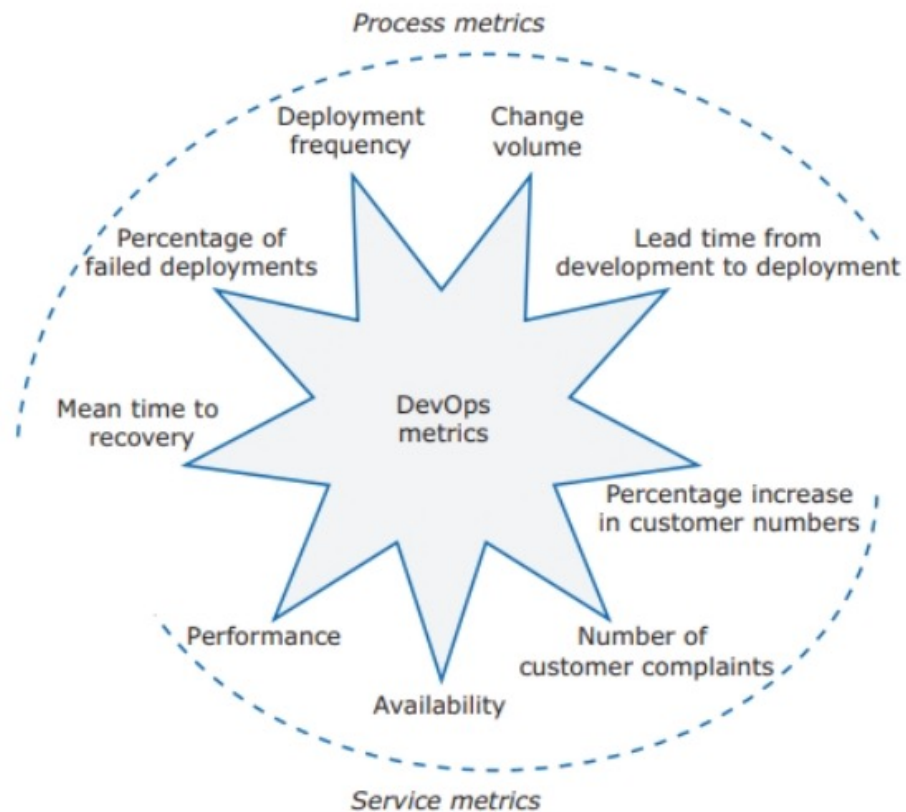
- instrument your software to collect data about itself
- use monitoring system to collect data about software's performance and availability

Process measurement is not simple

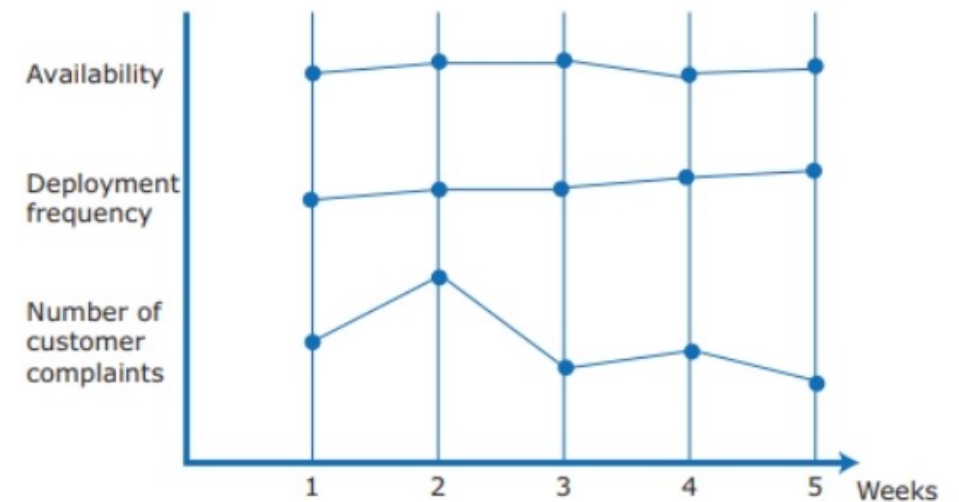
- different people record information differently
- e.g. to record lead time for implementing a change
 - Is the "lead time" the overall elapsed time or the time spent by developers?
 - How to take into account higher priority changes whose implementation slowed down other changes?

DevOps measurement

Example: Measuring the achievement of the goal of shipping cloud-based code frequently without causing customer outages



Metrics



Weekly presentation of results of analyses of collected data and trends

DevOps measurement

To collect data

- you may use **continuous integration tools** like Jenkins to collect data on deployments, successful tests, etc.
- you may use **monitoring services** featured by cloud providers like Amazon's Cloudwatch to collect data on availability and performance
- you may collect customer-supplied data from **issue management systems**
- you may **add instrumentation to your product** to gather data on its performance and how it is used by customers
 - use log files, where log entries are timestamped events reflecting customer actions and/or software responses
 - log as many events as possible
 - use available log analysis tools to extract useful information about on how your software is used

APM: New Relic



https://www.youtube.com/watch?v=J5tzl-8k_Q

DevOps?

Code management

DevOps automation

- Continuous integration

- Continuous delivery and deployment

- Infrastructure as code

DevOps measurement



Reference



Chapter 10 – DevOps and Code Management