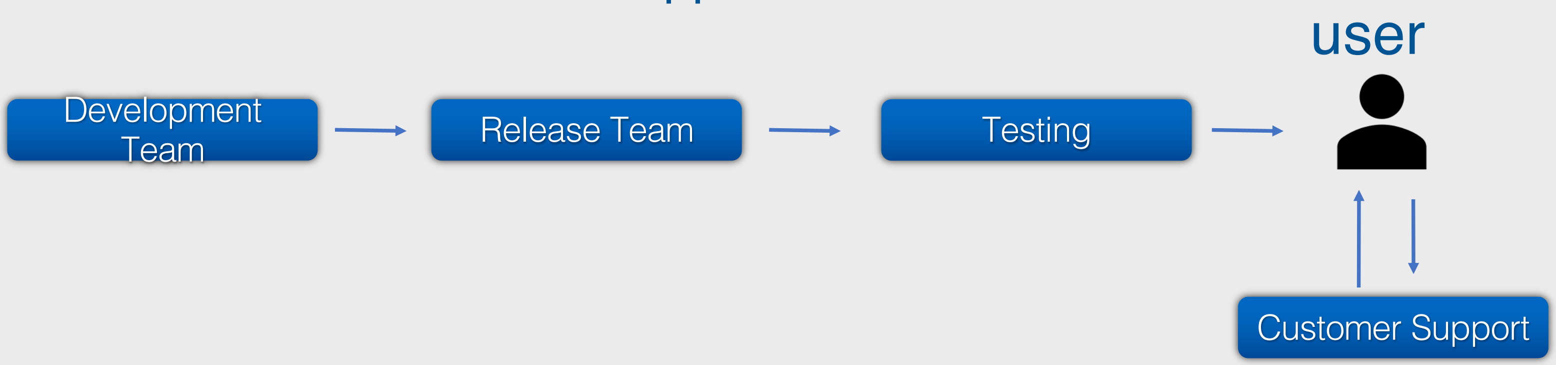
DevOps and Code Management

Software support

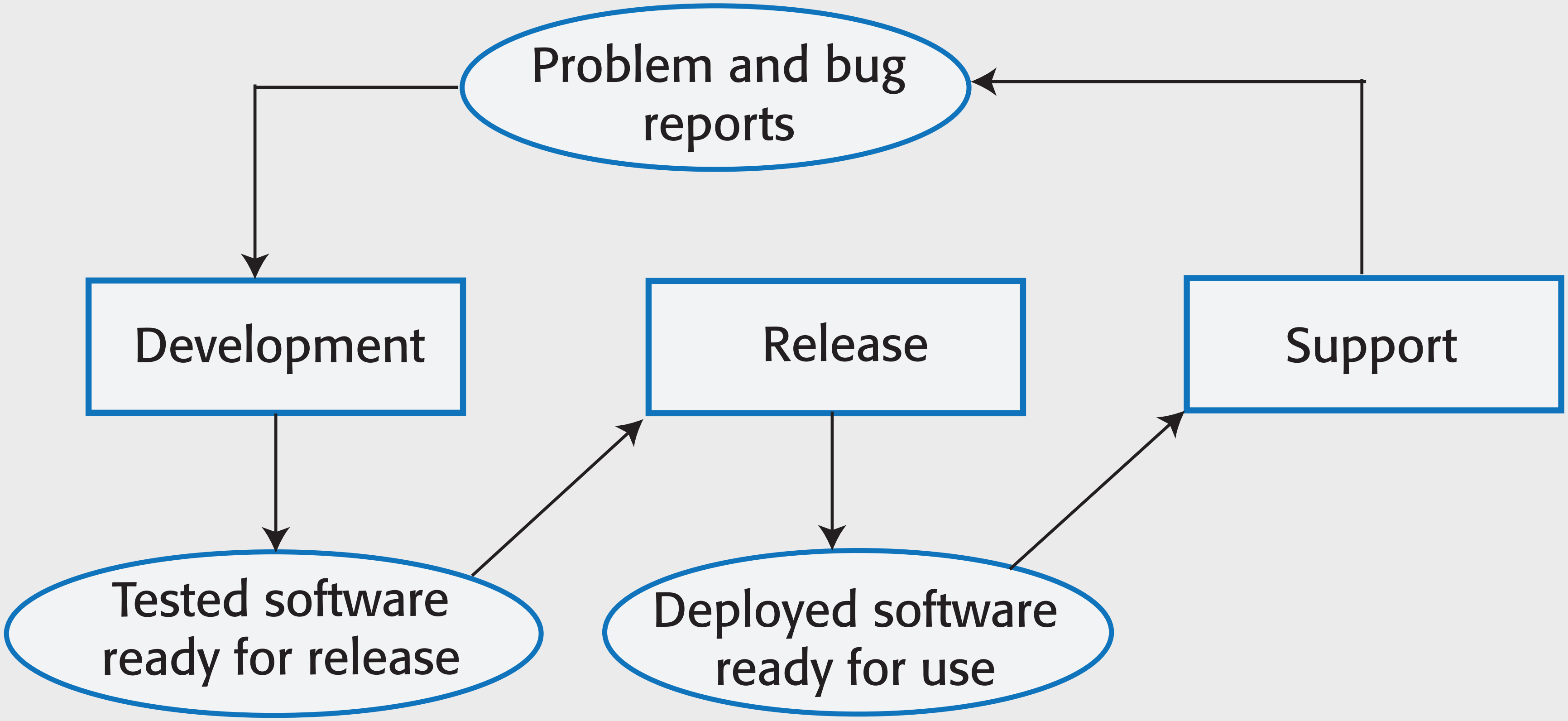
- Traditionally, separate teams were responsible software development, software release and software support.



- The original development team were sometimes also responsible for implementing software changes.

- Alternatively, the software may have been maintained by a separate ‘maintenance team’.

Development, release and support


Problem and bug reports
Development
Release
Support
Tested software ready for release
Deployed software ready for use

- inevitable delays and overheads in the traditional support model.

DevOps: Development+Operations

- To speed up the release and support processes, an alternative approach called **DevOps** has been developed.

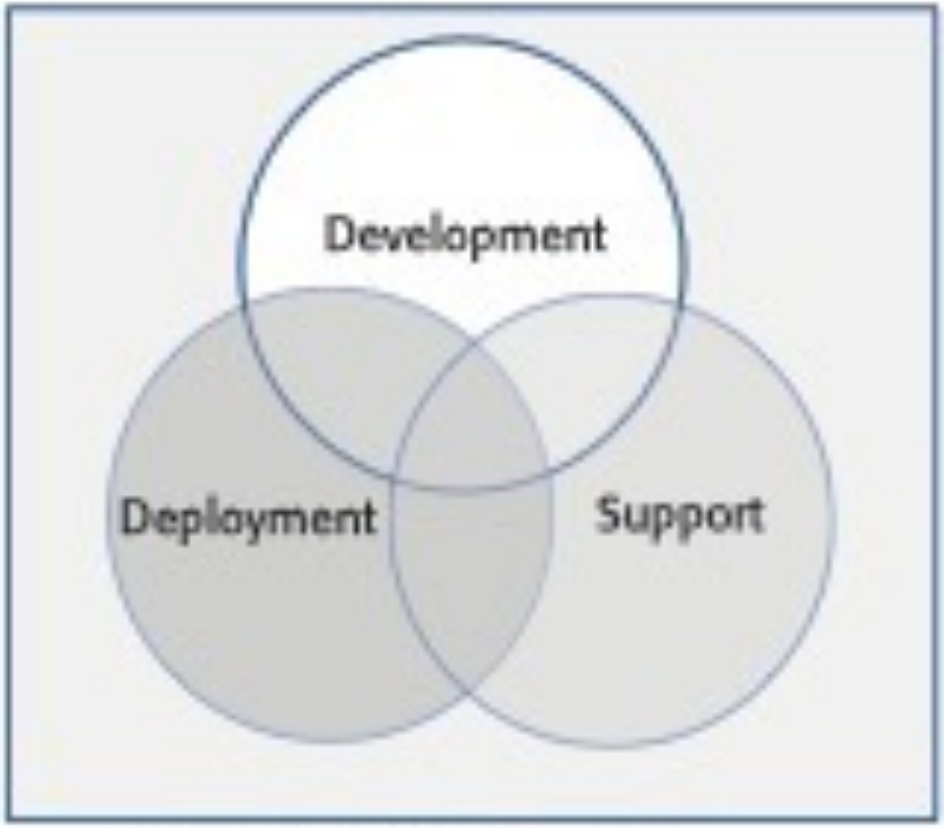
- Three factors led to the development and adoption of DevOps:

1. **Agile** methodology that reduced the development time

2. Amazon re-engineered their software around services
   - They suggested the same team develop and support each service
   - Amazon’s claim that this improves the reliability was widely publicized.

3. It became possible to release **software as a service**, running on a cloud.

DevOps



Multi-skilled DevOps 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 activities involved in testing, deployment and support should be automated if it is possible to do so.
- There should be minimal manual involvement in deploying software.

Measure first, change later

DevOps should be driven by a measurement program where you collect data about the system and its operation. You then use the collected data to inform decisions about changing DevOps processes and tools.

Benefits of DevOps

Faster deployment

- Software can be deployed to production more quickly because communication delays between the people involved in the process are dramatically reduced.

Reduced risk

- The increment of functionality in each release is small so there is less chance of feature interactions causing failures and outages.

Faster repair

- DevOps teams ***cooperate*** to get the software up and running again ***asap***.
- ***No*** *need to* ***wait and find the*** *responsible* ***team for*** *the problem and* ***waiting*** *for them to fix it.*

More productive teams

- DevOps teams are happier and more productive than the teams involved in the separate activities.

Code management

- During the software development, the team creates tens of thousands of lines of code and automated tests.
  - hundreds of code files, dozens of libraries, and several different programs that are involved in creating and running the code.
- **Code management** is a set of software-supported practices that is used to manage an evolving codebase.
  - They ensure that changes made by different developers do not interfere with each other, and to create different product versions.
- Code management tools make it easy to:
  - Create an executable product from its source code files
  - Run automated tests on that product.

Code management problem example

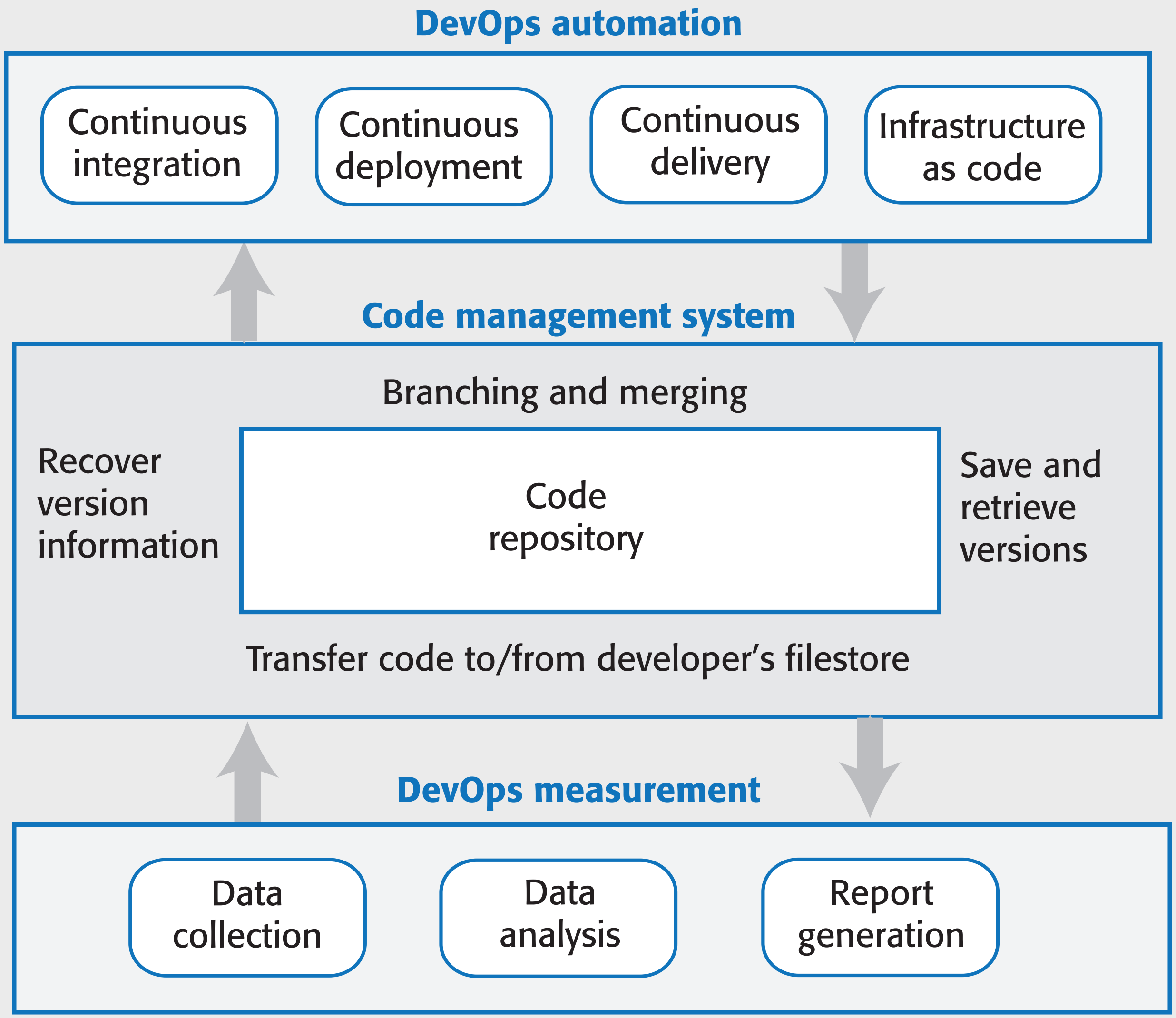
Alice and Bob worked for a company called FinanceMadeSimple and were team members involved in developing a personal finance product. Alice discovered a bug in a module called TaxReturnPreparation. The bug was that a tax return was reported as filed but, sometimes, it was not actually sent to the tax 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. FinanceMade Simple failed and both Bob and Alice lost their jobs.

Code management and DevOps

- Source code management, combined with automated system building, is essential for professional software engineering.
- In companies that use DevOps, a modern code management system is a fundamental requirement for ‘automating everything’.
- Not only does it store the project code that is ultimately deployed, it also stores all other information that is used in DevOps processes.
- DevOps automation and measurement tools all interact with the code management system

Code management and DevOps



Code management fundamentals

- Code management systems provide features that support four areas:
  - **Code transfer** Developers take code into their personal file store to work on it then return it to the shared code management system.
  - **Version storage and retrieval** Files may be stored in several different versions and specific versions of these files can be retrieved.
  - **Merging and branching** Parallel development branches may be created for concurrent working. Changes made by developers in different branches may be merged.
  - **Version information.** Information about different versions maintained in the system may be stored and retrieved

Code repository

- Source code management systems provide a shared repository and a set of features to manage the files in that repository:
  - All source code files and file versions are stored in the repository, as are other artifacts such as configuration files, build scripts, and shared libraries
  - Repository includes a database about the stored files: e.g., version information, who has changed the files, what changes were made at what times…
- Files can be transferred to and from the repository and information about the different versions of files and their relationships may 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 they are submitted to the system and can be retrieved using their identifier and other file attributes.

Change history recording
The reasons why changes to a code file have been made are recorded and maintained.

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.

Storage management
The code management system includes efficient storage mechanisms so that it doesn't keep multiple copies of files that have only small differences.

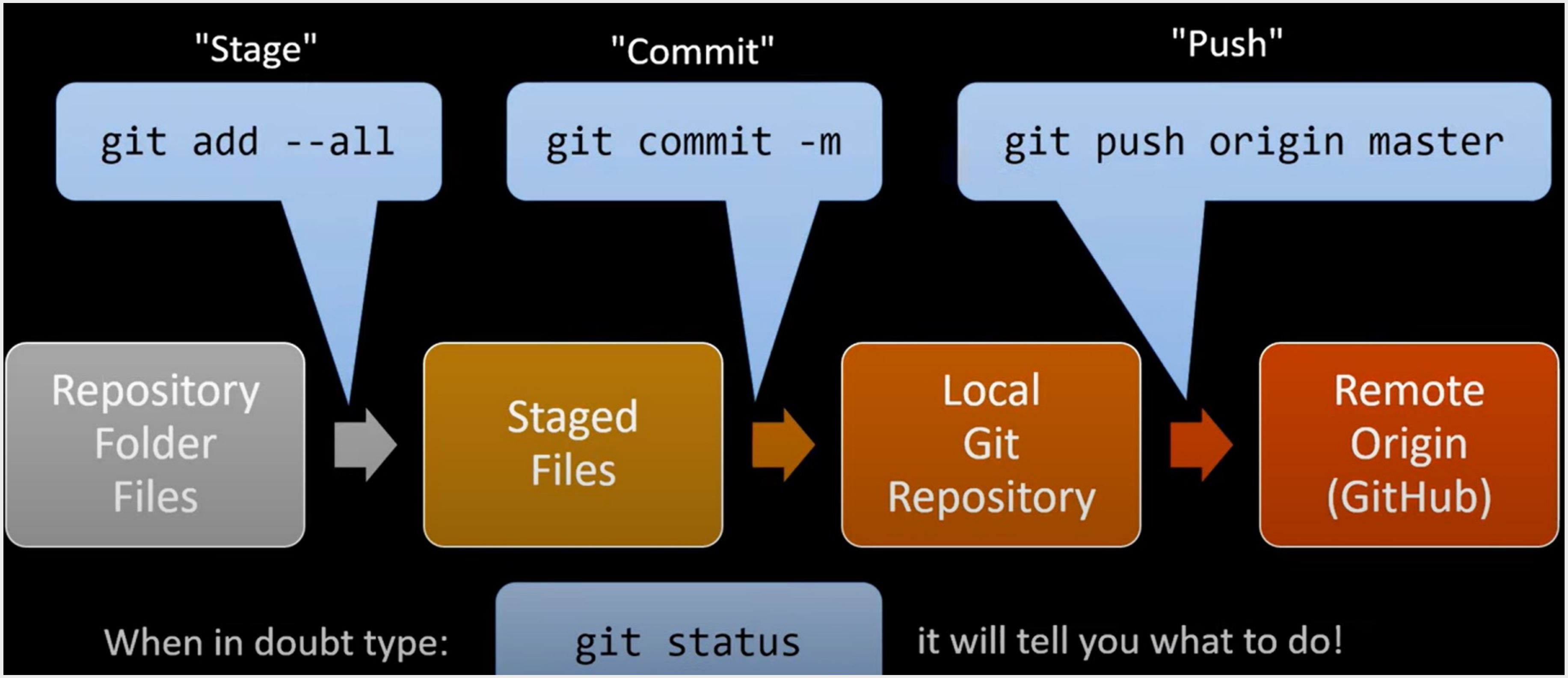
Git

- In 2005, Linus Torvalds, the developer of Linux, revolutionized source code management by developing a distributed version control system (DVCS) called Git to manage the code of the Linux kernel.

- This was to manage open-source projects.
  - It took advantage of the fact that storage costs had fallen to such an extent that most users did not have to be concerned with local storage management.

- Git maintains a clone of the repository on every user's computer

Git Basics



```
git add --all
```

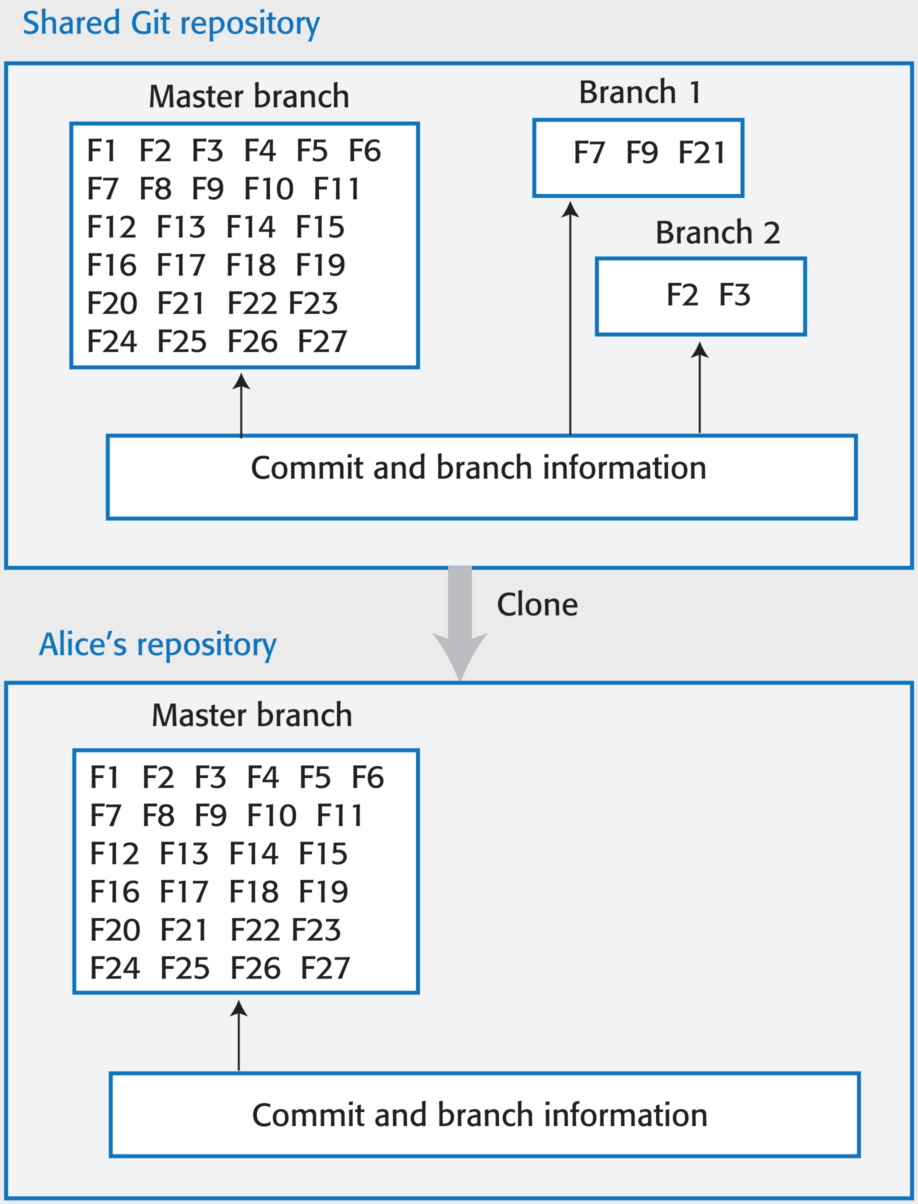
```
git commit -m
```

```
git push origin master
```

When in doubt type: `git status` it will tell you what to do!

Git tutorial:
https://www.youtube.com/watch?v=xvwBtODV0ms

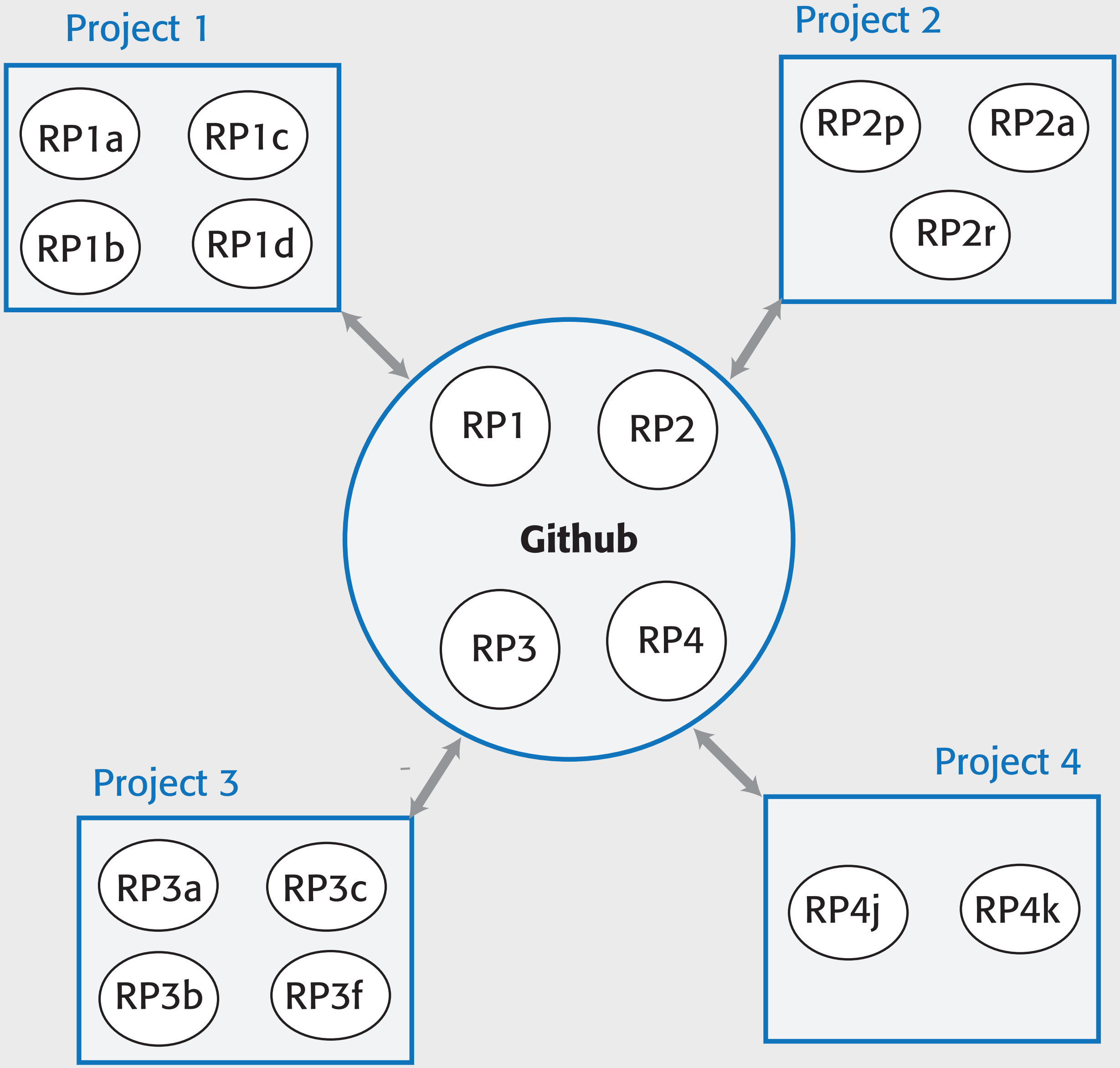
Repository cloning in Git



Benefits of distributed code management

- Resilience
  - Everyone working on a project has their own copy of the repository. If the shared repository is damaged or subjected to a cyberattack, work can continue, and the clones can be used to restore the shared repository. People can work offline if they don't have a network connection.
- Speed
  - Committing changes to the repository is a fast, local operation and does not need data to be transferred over the network.
- Flexibility (local experimentation global deployment)
  - Developers can safely experiment and try different approaches without exposing these to other project members. With a centralized system, this may only be possible by working outside the code management system.

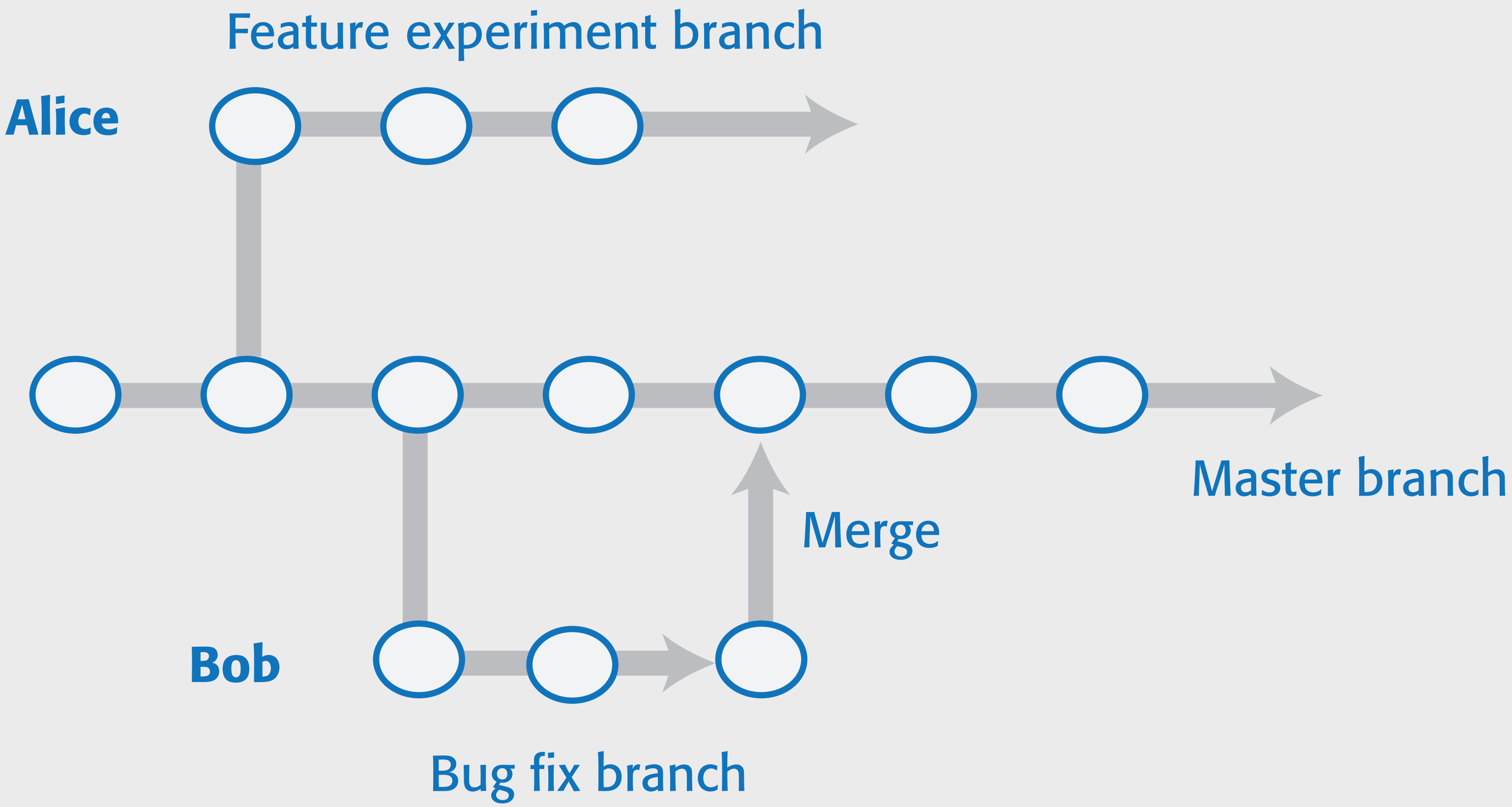
Git repositories



Branching and merging

- Branching and merging are fundamental ideas that are supported by all code management systems.
- **Branch**: is an independent, stand-alone version that is created when a developer wishes to change a file (isolated line of development).
  - 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 Alice or Bob make mistakes on the branch they are working on, they can easily revert to the master file.
- If they commit changes, while working, they can revert to earlier versions of the work they have done. When they have finished and tested their code, they can then replace the master file by merging the work they have done with the master branch

Branching and merging



Forks vs branch in Git (Github)?

DevOps+automation

- *Everything that can be, should be automated* is a fundamental principle of DevOps.

- By using DevOps with automated support, you can
  - reduce the time and costs for integration, deployment, and delivery.
  - makes integration, deployment and delivery more reliable and reproducible.

- Automation information is encoded in scripts that can be checked, reviewed, versioned and stored in the project repository.

What does DevOps automation mean?

Continuous integration

Upon committing a change to the master branch, an executable system is built

- This includes “building”, “running”, “adding artifacts”, and "integration tests”

Continuous delivery

A pre-production environment is created and the executable software version is tested by the quality assurance team.

Continuous deployment

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

Infrastructure as code

- Machine-readable scripts to configure infrastructure (network, servers, routers, etc.) for the produced software.
- The scripts also include installing and configuring software tools (compilers, libraries, DBMS).

System integration (system building)

- System integration is the process of gathering all the elements required in a working system and putting them together to create an operational system.
- Typical activities that are part of the system integration process include:
  - Installing database software and setting up the database with the appropriate schema.
  - Loading test data into the database.
  - Compiling the files that make up the product
  - Linking the compiled code with the libraries and other components used.
  - Checking that external services used are operational.
  - Deleting old configuration files and moving configuration files to the correct locations.
  - Running a set of system tests to check that the integration has been successful.
  - Examples of integration tools: Make, Ant (for Java), Maven, Yaml files, etc.

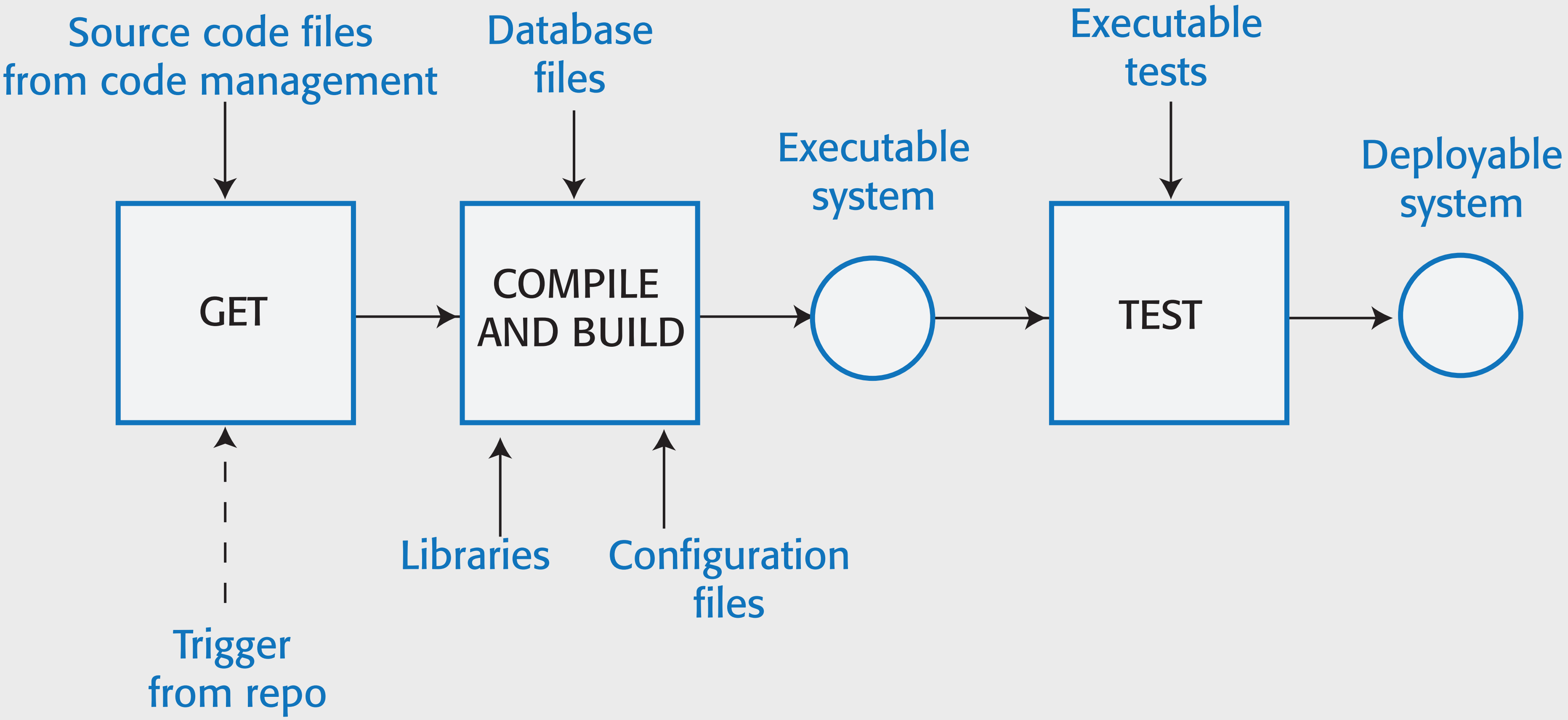
Continuous integration (CI)

- Continuous integration means that an integrated version of the system is created and tested every time a change is pushed to the repository.

- Upon completion of the push operation, the repository sends a message to an integration server to build a new version of the product

- The advantage of continuous integration (compared to infrequent integration) is faster finding and fixing bugs
  - If you make a small change and some system tests then fail, the problem almost certainly lies in the new code that is pushed to the repo!
  - You can focus on this code to find the bug causing the problem.

Continuous integration (e.g., via Azure DevOps)

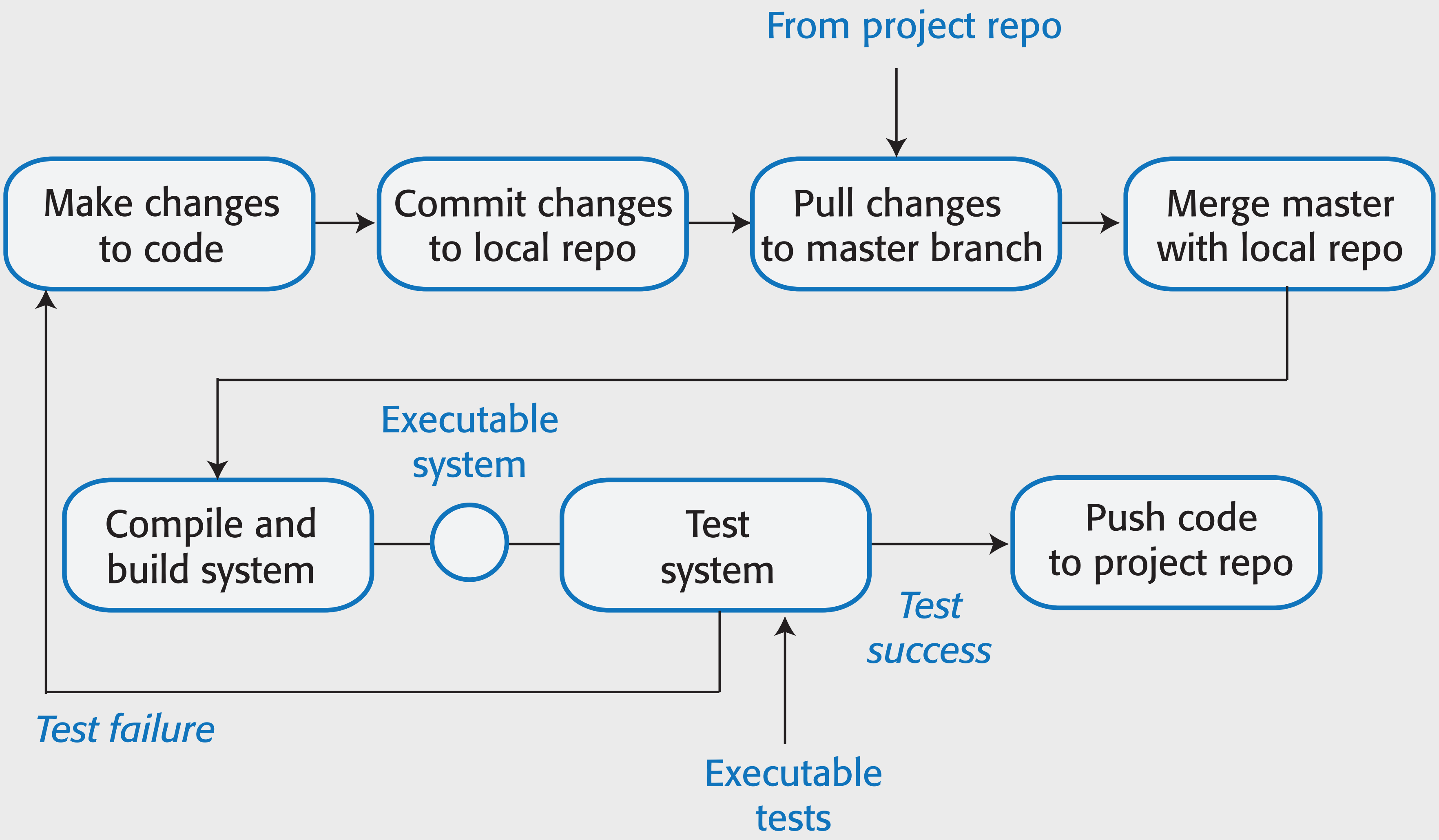


“Don’t break the build!”

- Breaking the build means pushing code to the project repository which, when integrated, causes some of the system tests to fail.
  - In a CI, developers have to make sure that they don’t ‘break the build’.
  - If this happens, you should discover and fix the problem so that normal development can continue.

- How to avoid breaking the build?
  - Practice ‘integrate twice’ approach
  - You should integrate and test on your own computer before pushing code to the project repository to trigger the integration server

First do the “local integration”



System building

- CI is only effective if the integration process is fast and developers do not have to wait for the results of their tests of the integrated system.

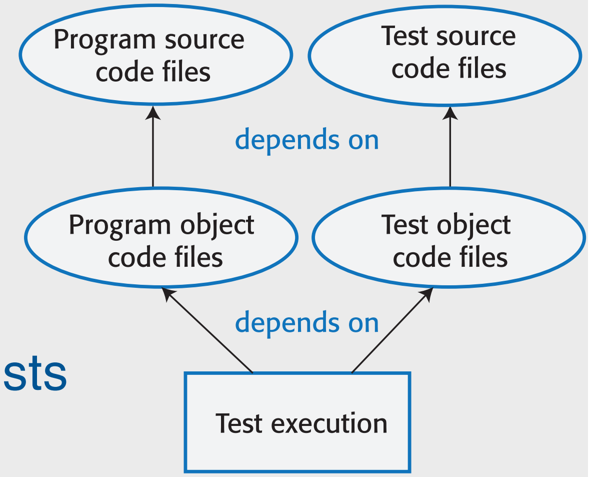
- However, some activities in the build process, such as populating a database or compiling hundreds of system files, are inherently slow.

- It is essential to have an automated build process that minimizes the time spent on these activities.

- Fast system building is achieved using a process of incremental building, where only those parts of the system that have been changed are rebuilt

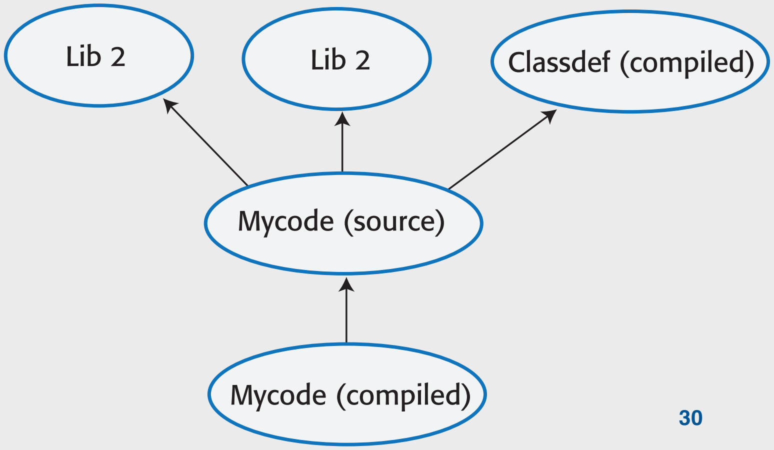
Dependencies

- Figure 1 is a dependency model that shows the dependencies for test execution.
    - The upward-pointing shows the information required to complete the task shown in the rectangle



- Running system tests depends on the existence of executable object code for the program and system tests

- Figure 2: a lower-level dependency model that shows the dependencies involved in creating the object code for a source code files, called Mycode.



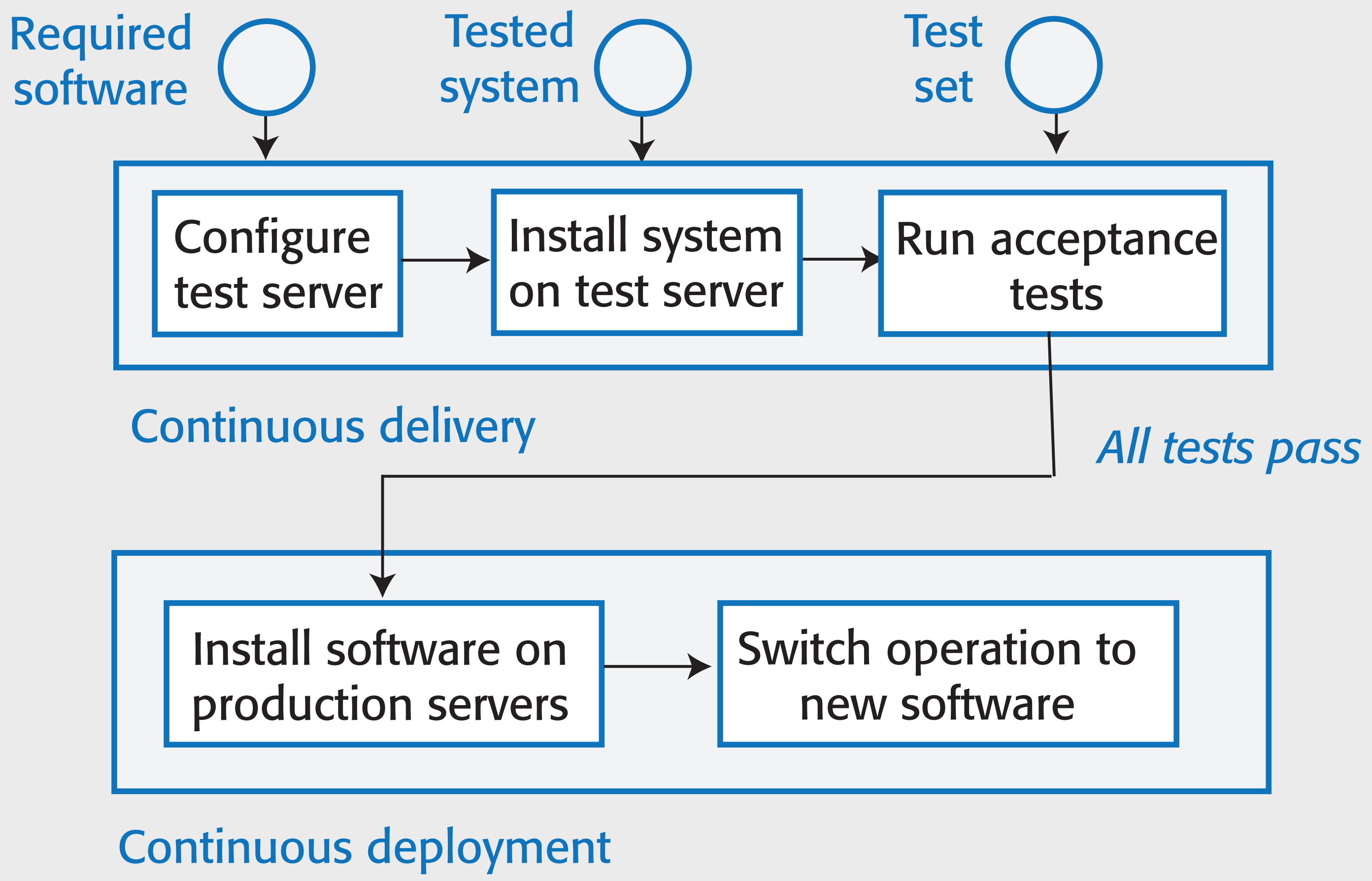
Automated build systems

- An automated build system uses the specification of dependencies to work out what needs to be done.
  - It uses the file modification timestamp to decide if a source code file has been changed.
  - The modification date of the compiled code is after the modification date of the source code.
    - The build system infers that no changes have been made to the source code and does nothing.
  - The modification date of the compiled code is before the modification date of the source code.
    - The build system recompiles the source and replaces the existing file of compiled code with an updated version.
  - The modification date of the compiled code is after the modification date of the source code. However, the modification date of Classdef is after the modification date of the source code of Mycode.
    - Therefore, Mycode has to be recompiled to incorporate these changes.

Continuous delivery vs deployment

- **Continuous integration** means creating an executable version of a software system whenever a change is made to the repository.
  - The CI tool builds the system and runs tests on your development computer or project integration server.
  - However, the real environment in which software runs will inevitably be different from your development system.
  - When your software runs in the operational environment bugs may be revealed that did not show up in the test environment.
- **Continuous delivery** means that, after making changes to a system, you ensure that the changed system is ready for delivery to customers.
  - You have to test it in a production environment to assure environmental factors do not cause failures or slow down

Continuous delivery and deployment



The deployment pipeline

- After initial integration testing, a staged test environment is created.
  - This is a replica of the actual production environment where the system will run.

- **System acceptance** tests, including functionality, load and performance tests, are done to check that the software works as expected.
  - Once all tests pass, the updated version is installed on the production servers.

- To deploy the system:
  - Momentarily stop all new requests for service and leave the older version to process the outstanding transactions.
  - Once these complete, switch to the new version and restart processing.

Benefits of continuous deployment (CD)

Reduced costs

If you use CD, you have no option but to invest in an *automated* deployment pipeline.

- Manual deployment is time-consuming and error-prone.
- Setting up an automated system is expensive and time-consuming, but you can recover these costs quickly if you make regular updates to your product.

Faster problem solving

- If a problem occurs, it only affect a small part of system and it is easy to debug
- If you bundle many changes into a single release, finding and fixing problems is more difficult.

Faster customer feedback

You can ask users for feedback and identify improvements that you need to make

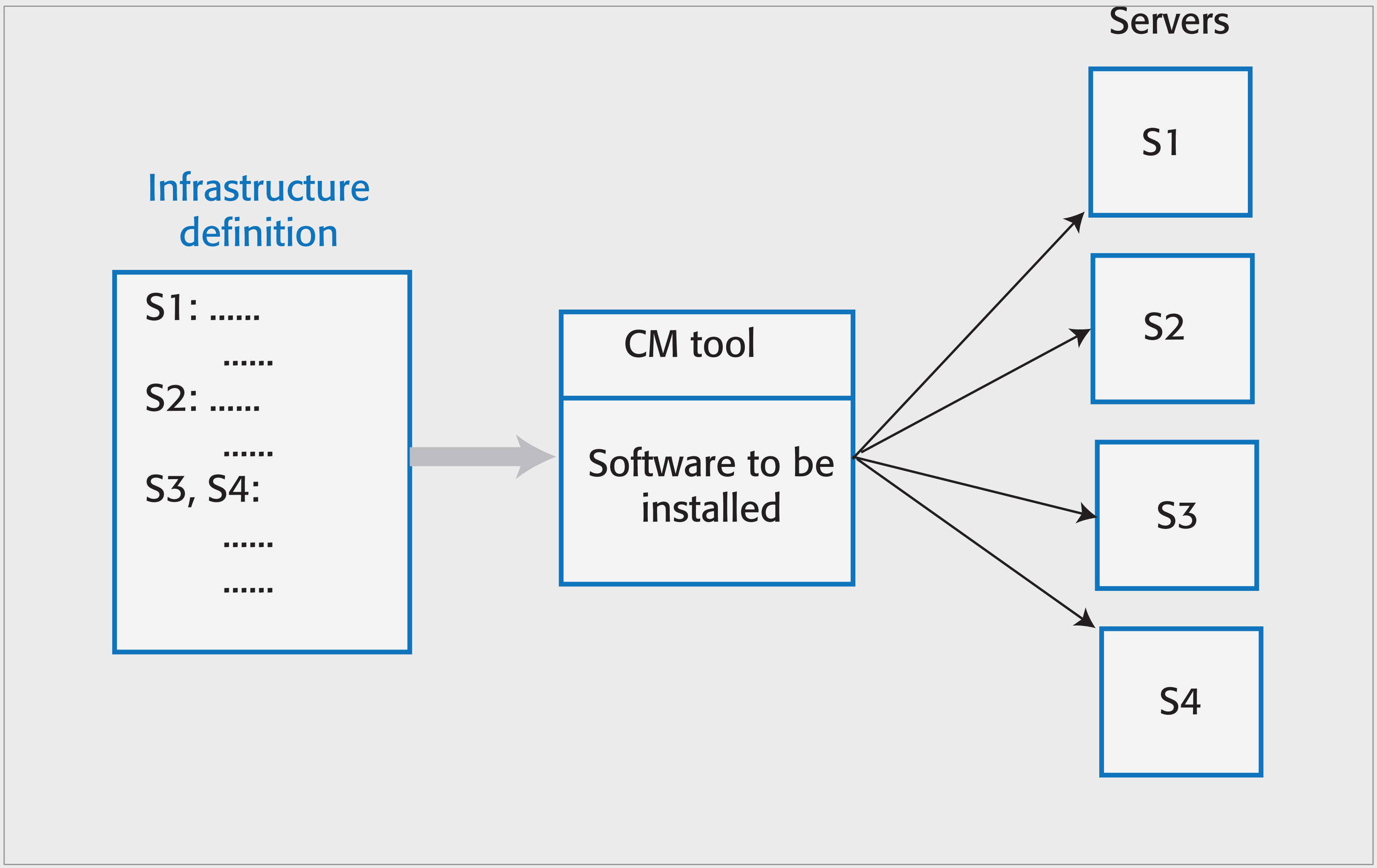
A/B testing

- This is an option if you have a large customer base and use several servers for deployment
- Deploy a new version of the software on some servers and older version running on others.
- Use load balancer to divert some customers to the new version and some to the older version.
- You can then measure and assess how new features are used to see if they do what you expect.

Infrastructure as code

- In an enterprise environment, there are different physical or virtual servers (web servers, database servers, file servers, etc.)
  - These have different configurations and run different software packages
  - It is difficult to keep track of the software installed on each machine
- Infrastructure as code proposed as a way to address this problem. Rather than manually updating the software on a company's servers, the process can be automated using a model of the infrastructure written in a machine-processable language.
- Configuration management (CM) tools (e.g., Puppet and Chef) can automatically install software and services on servers according to the infrastructure definition
- The benefits of infrastructure as code are lower costs of system management and lower risks of unexpected problems

Infrastructure as code



Characteristics of infrastructure as code

Visibility

Your infrastructure is defined as a stand-alone model that can be read, discussed, understood and reviewed by the whole DevOps team.

Reproducability

- Using a configuration management tool the installation tasks will always be run in the same sequence → the same environment is always created.
- You are not reliant on people remembering the order that they need to do things.

Reliability

System admins often make simple mistakes, especially when the same changes have to be made to several servers. Automating the process avoids these mistakes.

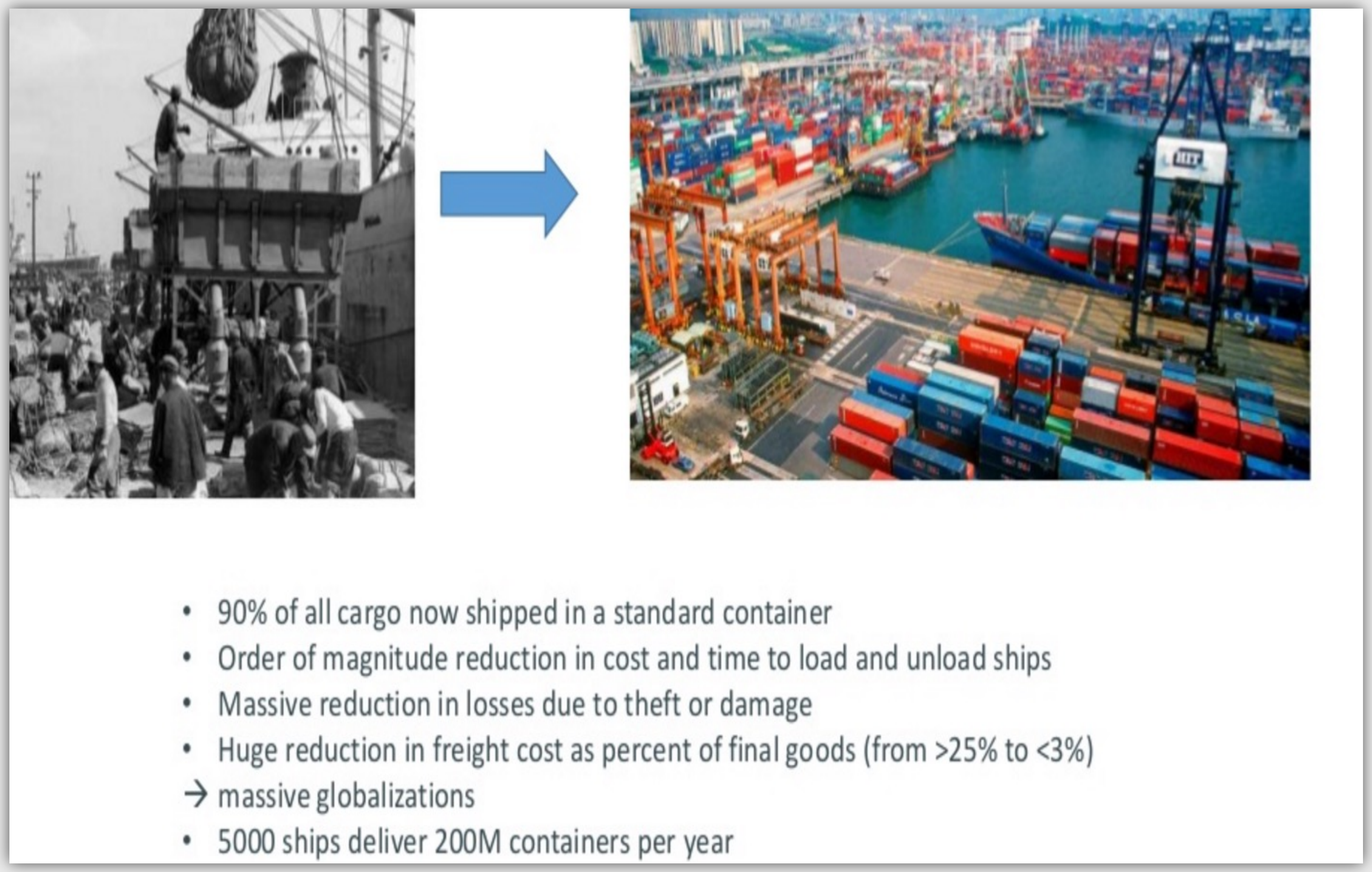
Recovery

- Like any other code, your infrastructure model can be versioned and stored in a code management system.
- If infrastructure changes cause problems, you can revert to an older version and reinstall the environment that you know works.

Containers

- A container provides a stand-alone execution environment running on top of an operating system such as Linux.
- The software installed in a Docker container is specified using a Dockerfile, which is, essentially, a definition of your software infrastructure as code.
- You build an executable container image by processing the Dockerfile.
- Using containers makes it very simple to provide identical execution environments.
  - For each type of server that you use, you define the environment that you need and build an image for execution.
  - You can run an application container as a test system or as an operational system; there is no distinction between them.
  - When you update your software, you rerun the image creation process to create a new image that includes the modified software.
  - You can then start these images alongside the existing system and divert service requests to them.

Container Nomenclature



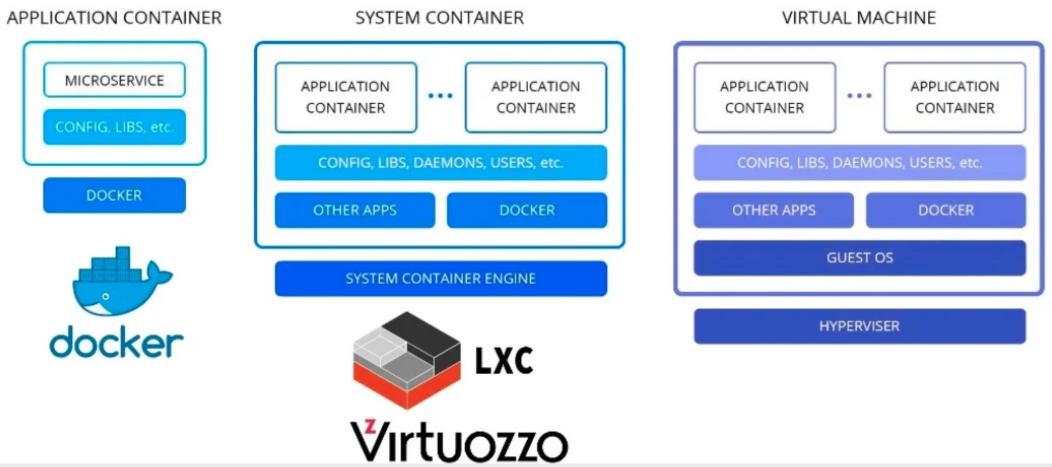
- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%)

→ massive globalizations

- 5000 ships deliver 200M containers per year

Various Container Technologies

- Container technologies:
  - Application container
    - Ex: **Docker,** Podman, Singularity.
  - System container
    - LXC and OpenVZ (developed by Virtuozzo)



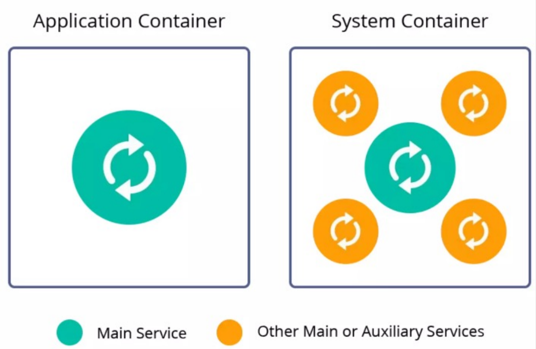
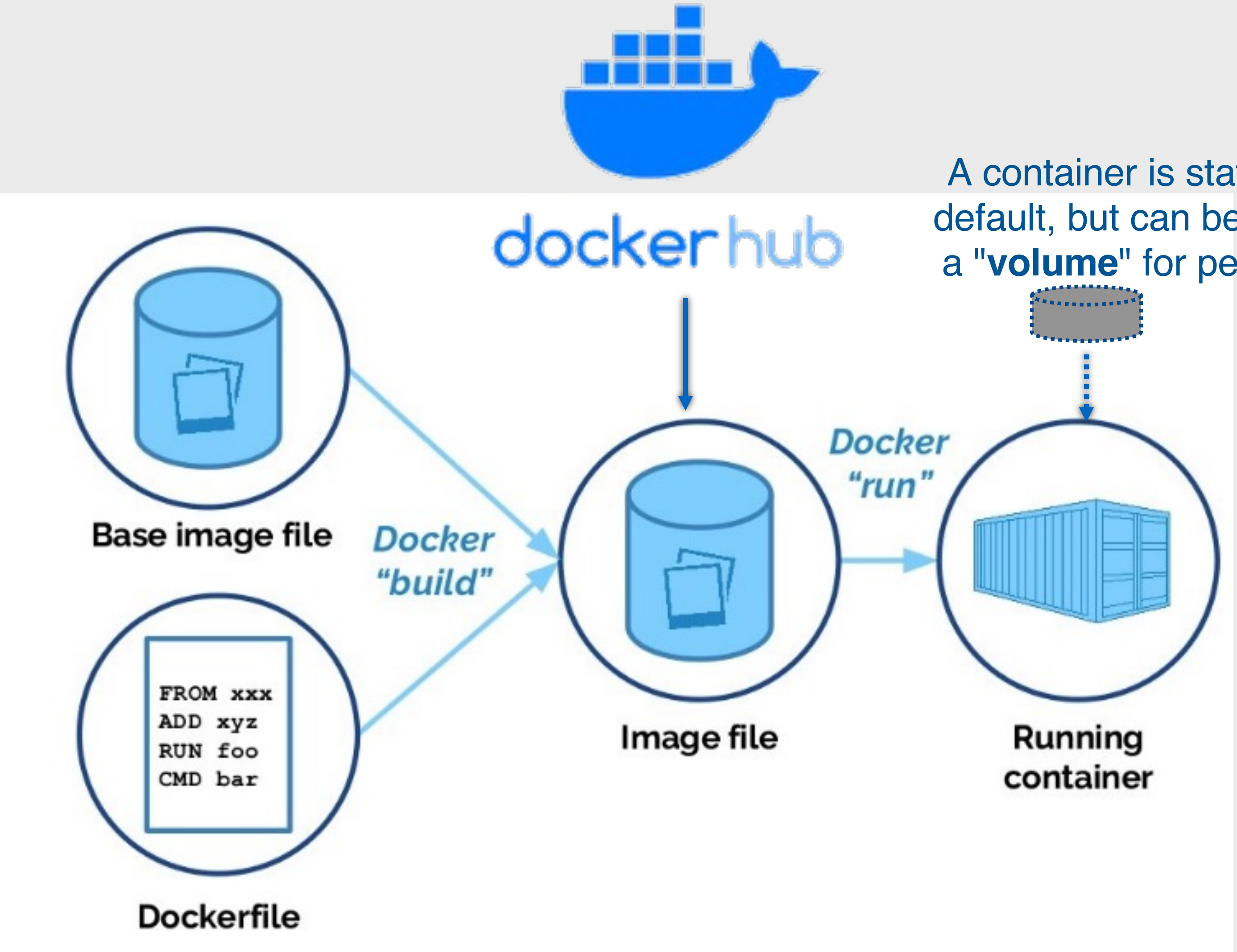


Image source: https://www.slideshare.net/jelastic/kubernetes-and-nested-containers-enhanced-3-ps-performance-price-and-provisioning

Docker File, Image, Hub, Instance, and Volume

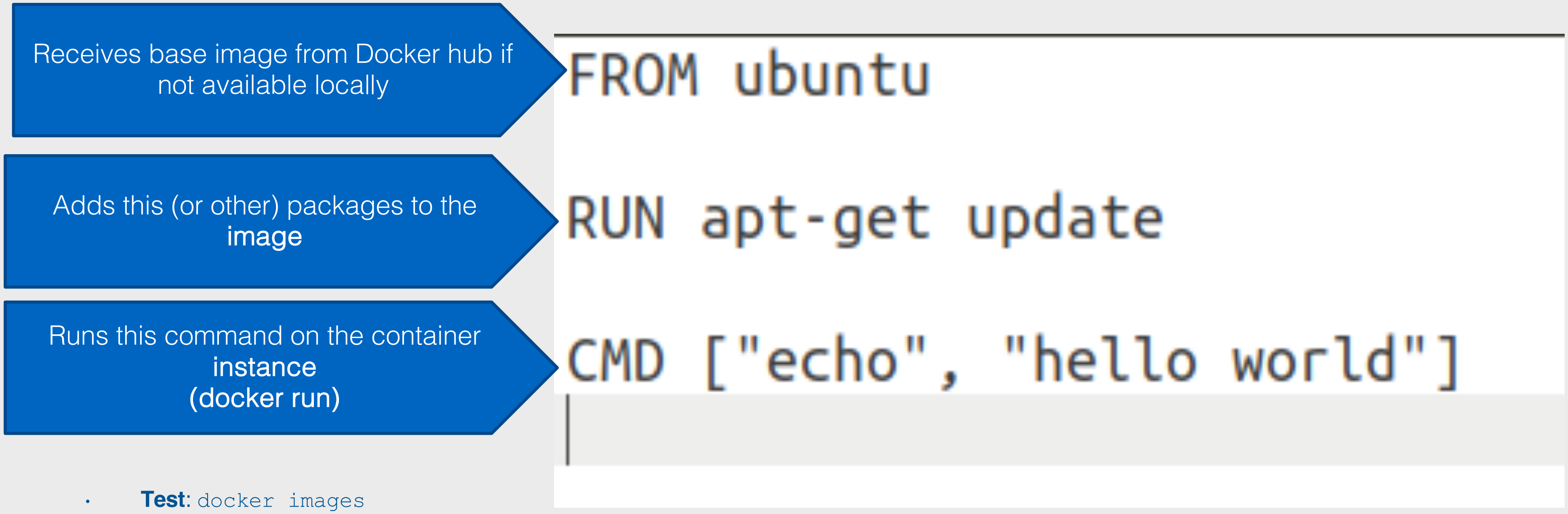


```
FROM xxx
ADD xyz
RUN foo
CMD bar
```

A container is stateless by default, but can be attached a "**volume**" for persistence

Docker File: An Example

- A script that automates docker image creation
  - It instructs Docker how to build an image
  - Below: A Hello World example on an ubuntu base image



Receives base image from Docker hub if not available locally

Adds this (or other) packages to the image

Runs this command on the container instance (docker run)

```
FROM ubuntu

RUN apt-get update

CMD ["echo", "hello world"]
```

  - **Test**: `docker images`
  - **Test**: `docker run image-name`

DevOps measurement

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

- There are four types of software development measurement:

  - **Process measurement** You collect and analyze data about your development, testing and deployment processes.

  - **Service measurement** You collect and analyze data about the software’s performance, reliability and acceptability to customers.

  - **Usage measurement** You collect and analyze data about how customers use your product.

  - **Business success measurement** You collect and analyze data about how your product contributes to the overall success of the business.

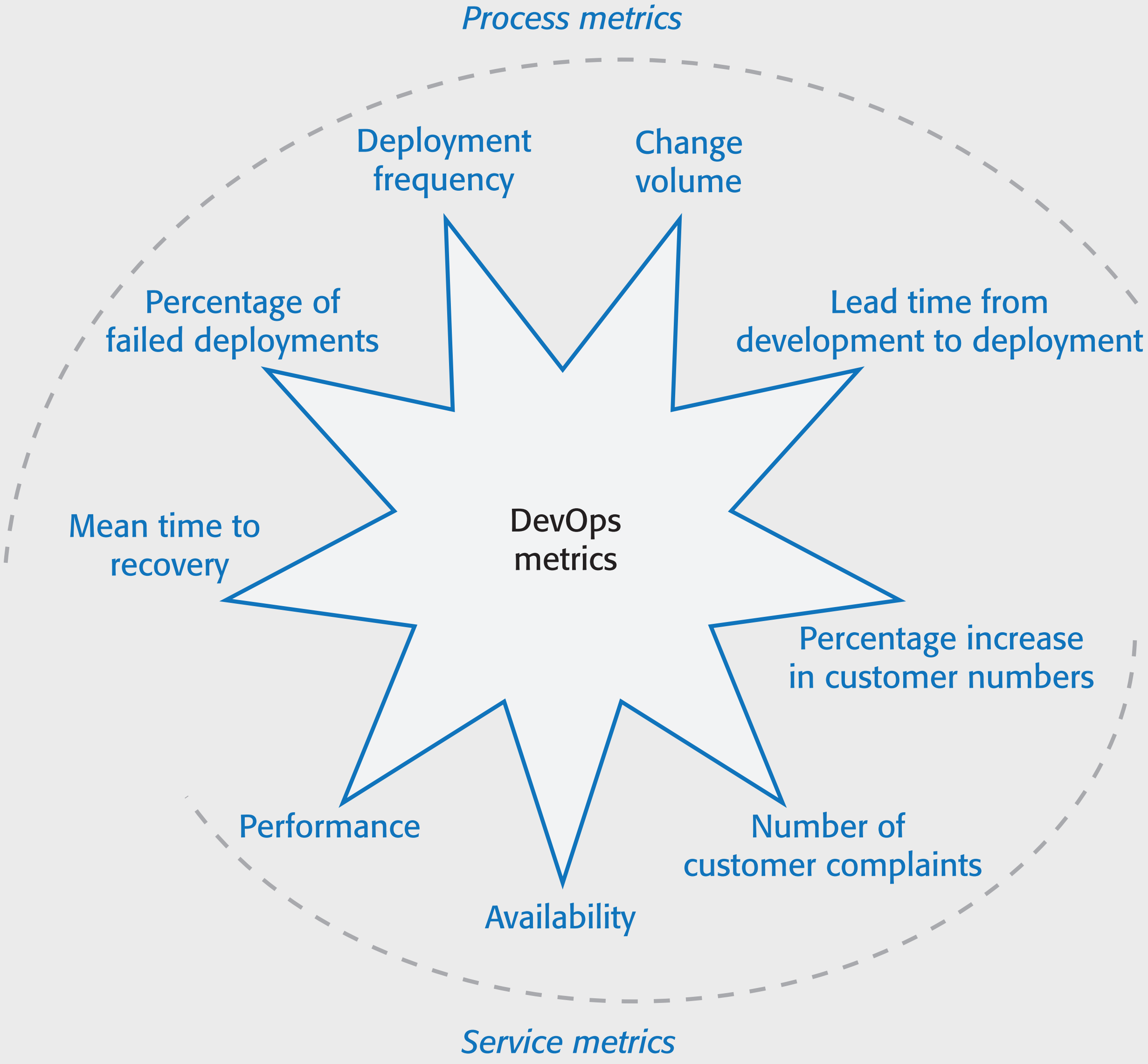
Automating measurement!

- You should instrument your software to collect data about itself

- You can also use a monitoring system, to collect data about your software's performance and availability.
  - e.g., Prometheus or Datadog

- Some process measurements can also be automated.
  - However, there are problems in process measurement because people are involved. They work in different ways, may record information differently and their performance varies over time.

Metrics used in the DevOps scorecard



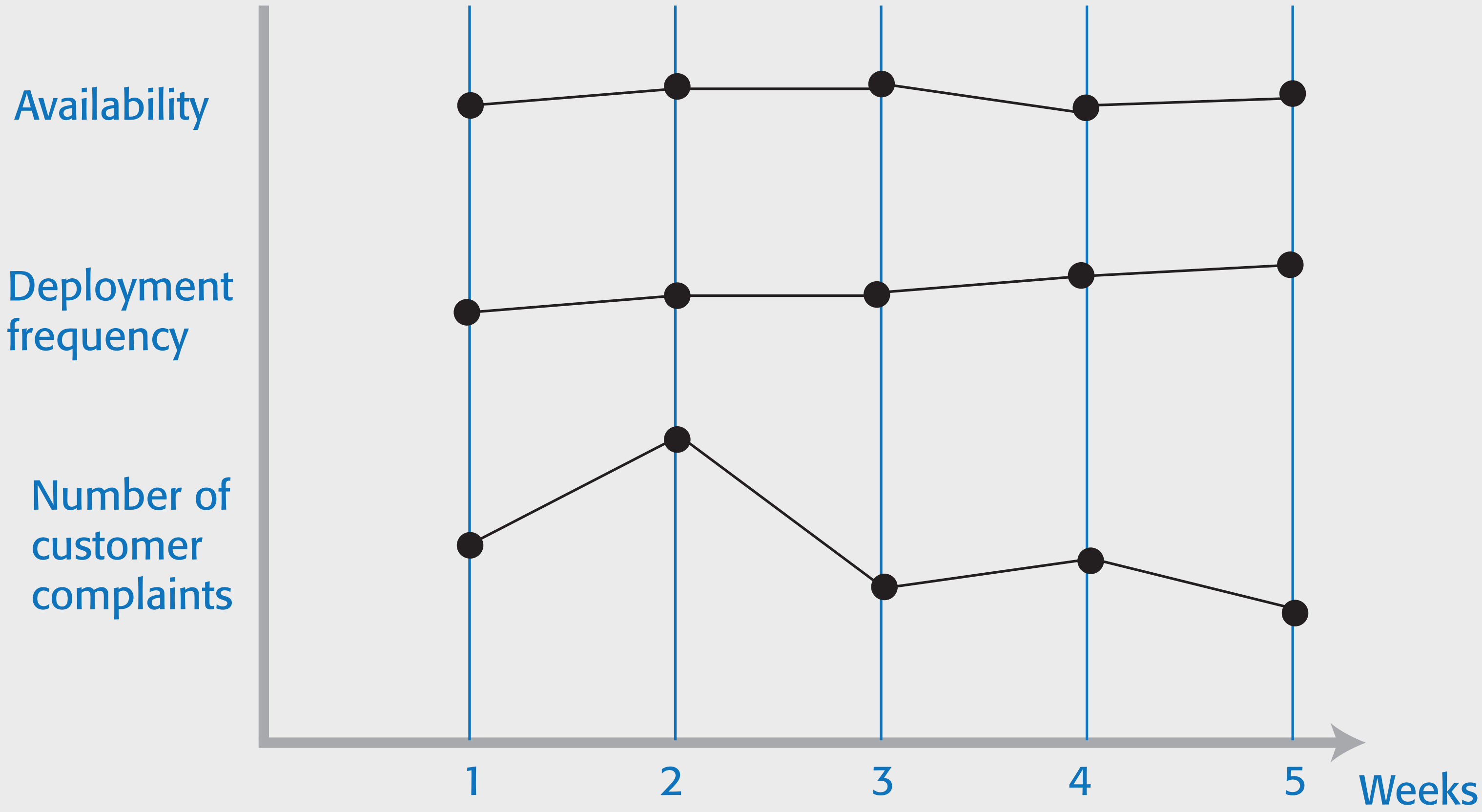
Process metrics
Deployment frequency
Change volume
Percentage of failed deployments
Lead time from development to deployment
Mean time to recovery
DevOps metrics
Percentage increase in customer numbers
Performance
Number of customer complaints
Availability
Service metrics

© Ian Sommerville 2018:

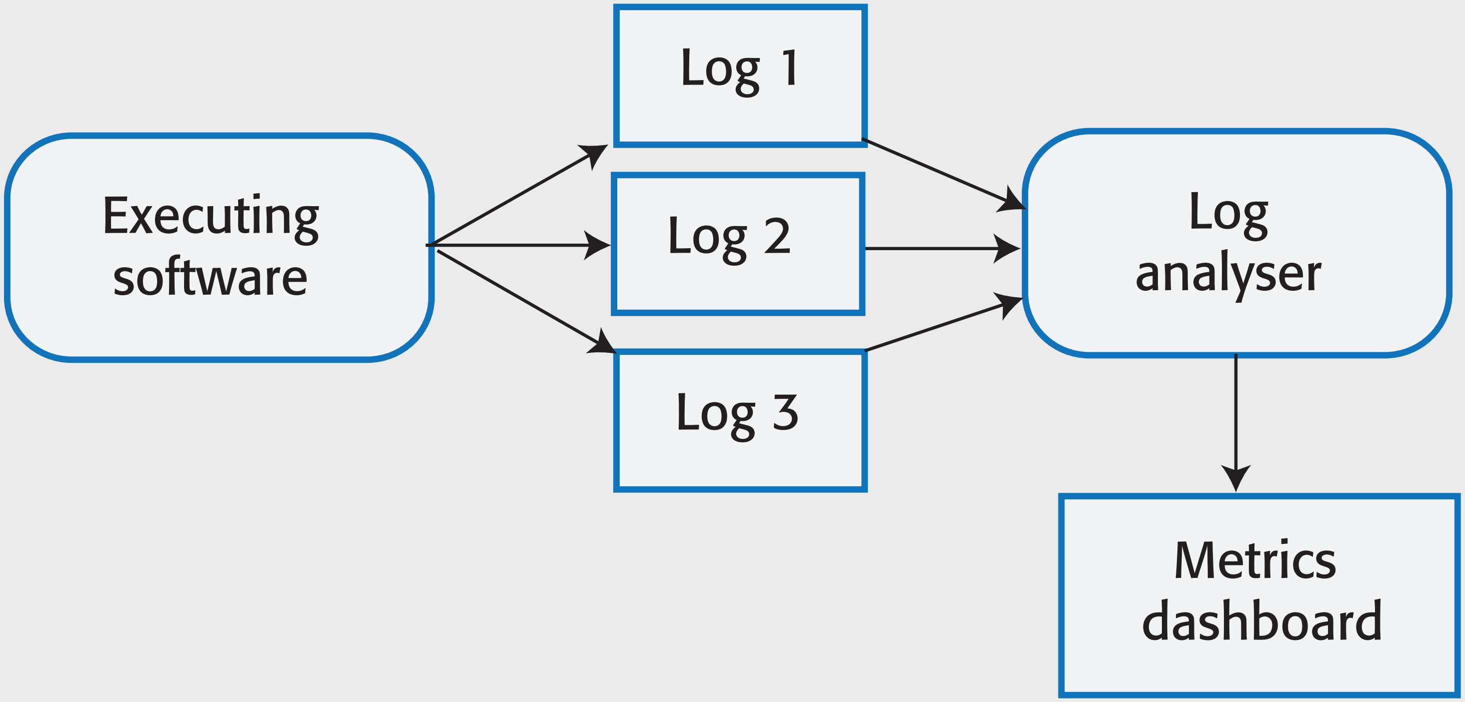
Metrics scorecard

- Payal Chakravarty from IBM suggests a practical approach to DevOps measurement based around a metrics scorecard with 9 metrics:
  - These are relevant to software that is delivered as a cloud service. They include process metrics and service metrics
  - For the process metrics,
    - You would like to see **decreases** in:
      - Number of failed deployments
      - Mean time to recovery after a service failure
      - Lead time from development to deployment.
    - You would hope to see **increases**
      - deployment frequency
      - number of lines of changed code (change volume) that are shipped.
  - For the service metrics: availability **and** performance should be stable or improving; the number of customer complaints should be *decreasing*, and the number of new customers should be increasing.

Devops metrics trends (needed in your final report):



Logging and analysis



Key points 1

- DevOps is the integration of software development and the management of that software once it has been deployed for use. The same team is responsible for development, deployment and software support.

- The benefits of DevOps are faster deployment, reduced risk, faster repair of buggy code and more productive teams.

- Source code management is essential to avoid changes made by different developers interfering with each other.

- All code management systems are based around a shared code repository with a set of features that support code transfer, version storage and retrieval, branching and merging and maintaining version information.

- Git is a distributed code management system that is the most widely used system for software product development. Each developer works with their own copy of the repository which may be merged with the shared project repository.

Key points 2

- Continuous integration means that as soon as a change is committed to a project repository, it is integrated with existing code and a new version of the system is created for testing.
- Automated system building tools reduce the time needed to compile and integrate the system by only recompiling those components and their dependents that have changed.
- Continuous deployment means that as soon as a change is made, the deployed version of the system is automatically updated. This is only possible when the software product is delivered as a cloud-based service.
- Infrastructure as code means that the infrastructure (network, installed software, etc.) on which software executes is defined as a machine-readable model. Automated tools, such as Chef and Puppet, can provision servers based on the infrastructure model.
- Measurement is a fundamental principle of DevOps. You may make both process and product measurements. Important process metrics are deployment frequency, percentage of failed deployments, and mean time to recovery from failure.