Software Engineering COMP1035

Lecture 13

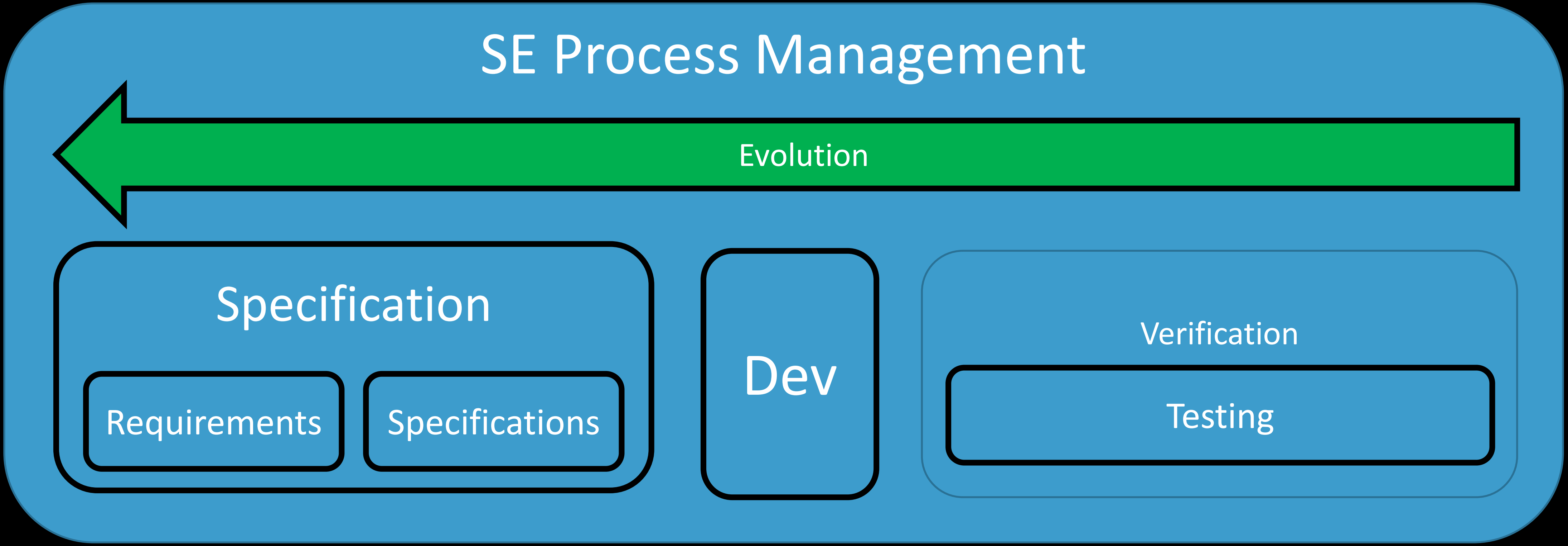
Config & Deployment

Outline

- Configuration Management
- Version Control
- System Building
- Change Management
- Release Management

Where We Are In the Process

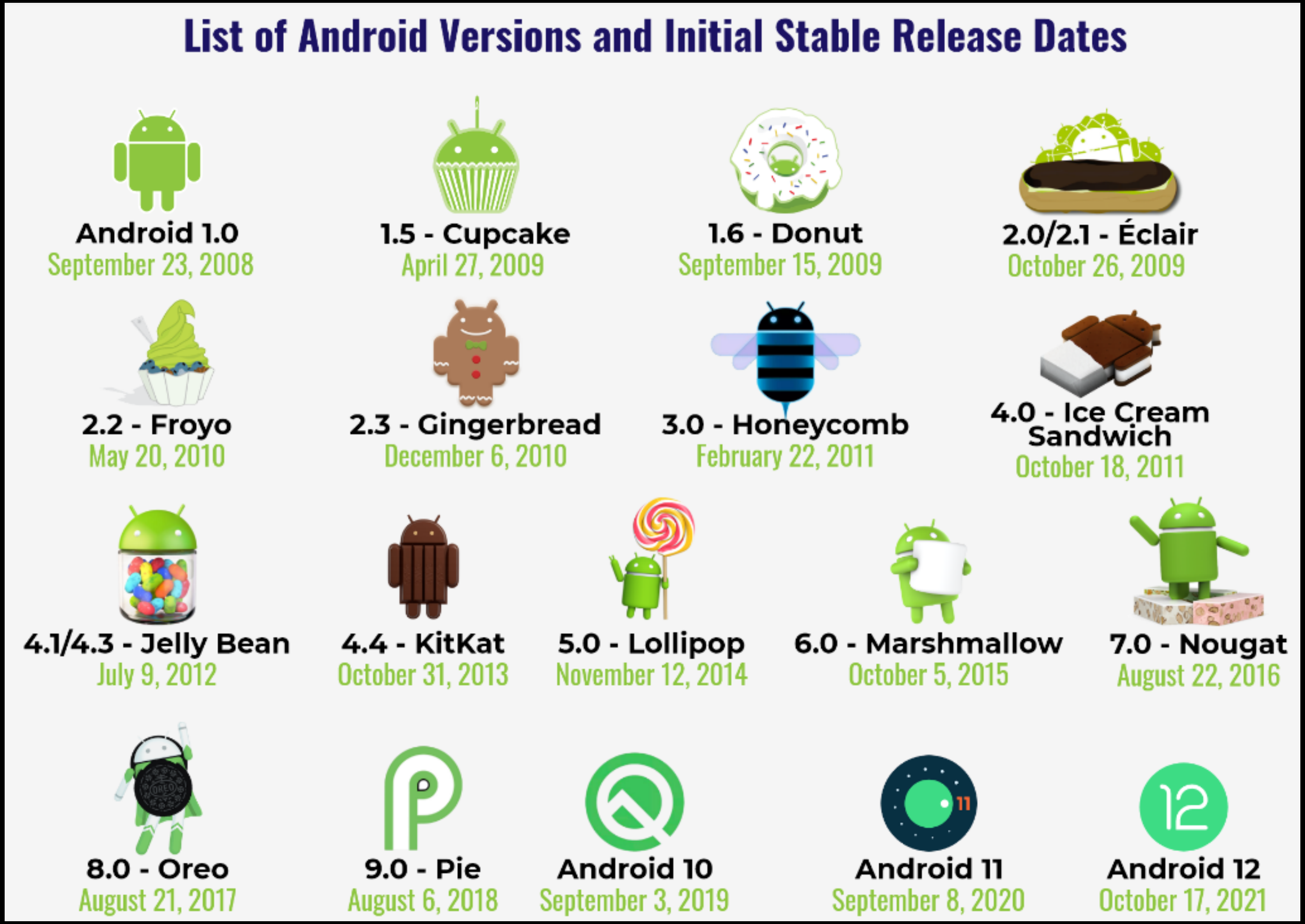
Keeping Track of SE Module



Why Evolution?

- **Why software systems are constantly changing (evolution)?**

1. Bugs are discovered and have to be fixed.
2. New versions of hardware and system platforms are released.
3. Competitors introduce new features.
4. System requirement changes.


List of Android Versions and Initial Stable Release Dates
Android 1.0 September 23, 2008
1.5 - Cupcake April 27, 2009
1.6 - Donut September 15, 2009
2.0/2.1 - Éclair October 26, 2009
2.2 - Froyo May 20, 2010
2.3 - Gingerbread December 6, 2010
3.0 - Honeycomb February 22, 2011
4.0 - Ice Cream Sandwich October 18, 2011
4.1/4.3 - Jelly Bean July 9, 2012
4.4 - KitKat October 31, 2013
5.0 - Lollipop November 12, 2014
6.0 - Marshmallow October 5, 2015
7.0 - Nougat August 22, 2016
8.0 - Oreo August 21, 2017
9.0 - Pie August 6, 2018
Android 10 September 3, 2019
Android 11 September 8, 2020
Android 12 October 17, 2021

Configuration Management

- Configuration Management (CM): track and control changes in the software.
    1. **Version Control** : keep track of the multiple versions of system components, and ensure that changes made to components by different developers do not interfere with each other.
    2. **System Building**: assemble program components, data, and libraries, then compile and link these to create an **executable system**.
    3. **Change Management**: keep track of requests for changes to delivered software from customers and developers, working out the costs and impact, and decide if and when the changes should be implemented.
    4. **Release management**: Prepare software for external release and keep track of the system versions that have been released for customer use.

Version Control - Baseline

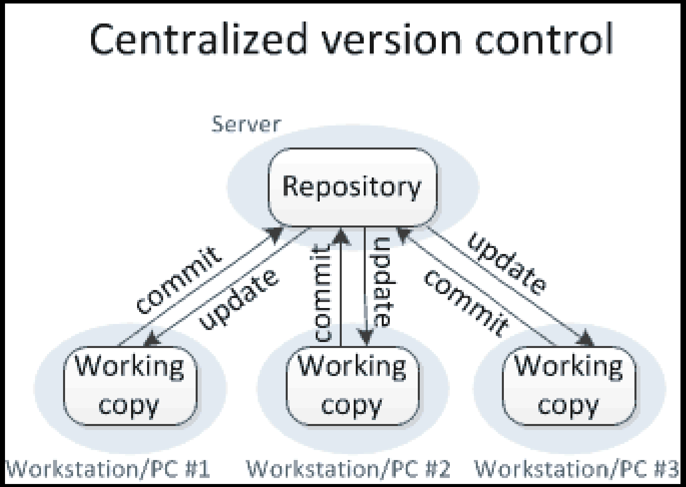
- Baseline: an agreed description of the attributes of a product, at a point in time (snapshot)
- A change is a movement from this baseline state to a next state
- As the project develops, new baselines are established, resulting in several versions of the software.

Version Control System

- Version Control System: identify, store and control access to the different versions of components.
  - **Centralized system**: a **single master repository** maintains all versions of the software components that are being developed. (Apache Subversion)
  - **Distributed system**: **multiple versions** of the component **repository** exist at the same time. (Git)
- **Project Repository**: maintains the master version of all components.
- **Private Workspace**: copy of master version. Developers can work on the files required and maintain the new versions.

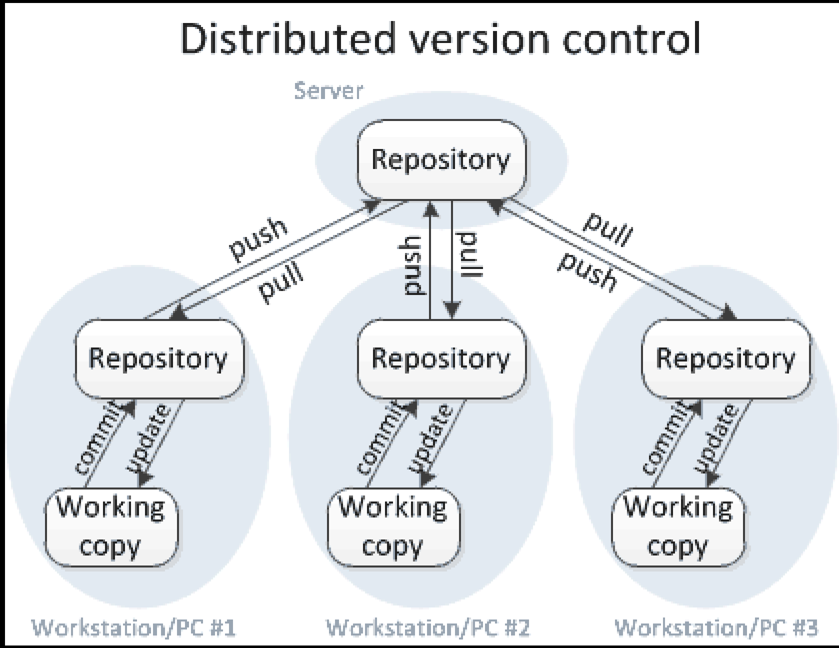
Centralized Version Control System

- A server acts as the master repository which stores every version of code.
- Every user commits directly to the main branch, and others can see the changes
- Pros:
  - Works well with binary files (graphic assets, text files), save space
  - Offers full visibility
  - Decreases the learning curve (easy to use)
- Cons:
  - A single point of failure risks data
  - Doesn’t work for large teams – hard to find stable moments to push changes



Distributed Version Control System

- Every developer clones a copy of a repository and has the **full** history of the project on their own hard drive (with all of the metadata).
- Commits are only necessary when sharing changes
- Pros:
  - Performing actions other than pushing and pulling are fast
  - Committing changesets can be done locally without anyone else seeing them
  - Allows private (off-line) work
- Cons:
  - Initial checkout of a repository is slow
  - Additional storage required (binary files, long project history)



Version Control System - Branches

- Branching: the duplication of an object under version control (such as a source code file or a directory tree).
- Each object can thereafter be modified separately and in parallel so that the objects become different.
- Rather than a linear sequence of versions that reflect changes to the component over time, there may be several independent sequences resulting from branching.

Version Control – Branches

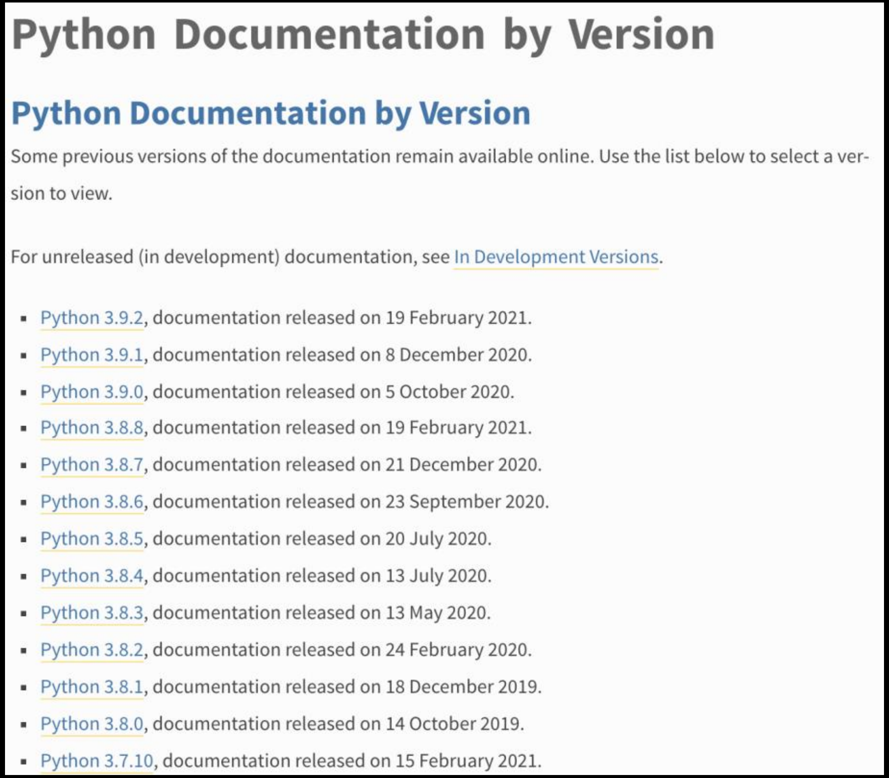
Scenario1: No Branches.

Your team only works from the main source tree.

This scenario is generally for small or medium size teams that do not require isolation for teams or features, and do not need the isolation for releases.

Secnario2: Branch for release.

Your team creates branches to support an ongoing release. This is the most common case where you need to create a branch to stabilize for a release. In this case, your team creates a branch before release time to stabilize the release and then merges changes from release branch back to the main source tree after the software is released.



Python Documentation by Version

Python Documentation by Version

Some previous versions of the documentation remain available online. Use the list below to select a version to view.

For unreleased (in development) documentation, see In Development Versions.

- Python 3.9.2, documentation released on 19 February 2021.
- Python 3.9.1, documentation released on 8 December 2020.
- Python 3.9.0, documentation released on 5 October 2020.
- Python 3.8.8, documentation released on 19 February 2021.
- Python 3.8.7, documentation released on 21 December 2020.
- Python 3.8.6, documentation released on 23 September 2020.
- Python 3.8.5, documentation released on 20 July 2020.
- Python 3.8.4, documentation released on 13 July 2020.
- Python 3.8.3, documentation released on 13 May 2020.
- Python 3.8.2, documentation released on 24 February 2020.
- Python 3.8.1, documentation released on 18 December 2019.
- Python 3.8.0, documentation released on 14 October 2019.
- Python 3.7.10, documentation released on 15 February 2021.

Version Control – Branches

*Scenario 3 – Branch for **Maintenance**.*

Your team creates a branch to maintain an old build. In this case, you create a branch for your maintenance efforts, so that you do not destabilize your current production builds. You may or may not merge changes from the maintenance branch back into the main tree.

*Scenario 4 – Branch for **Feature**.*

Your team creates branches based on features. In this case, you create a **development branch**, perform work in the development branch, and then **merge back** into your main source tree. You can create separate branches for work on specific features to be completed in parallel.

Version Control – Branches

- *Scenario 5 – Branch for **Team**.*

You branch to isolate sub-teams so they can work without being subject to breaking changes, or can **work in parallel towards unique milestones**.

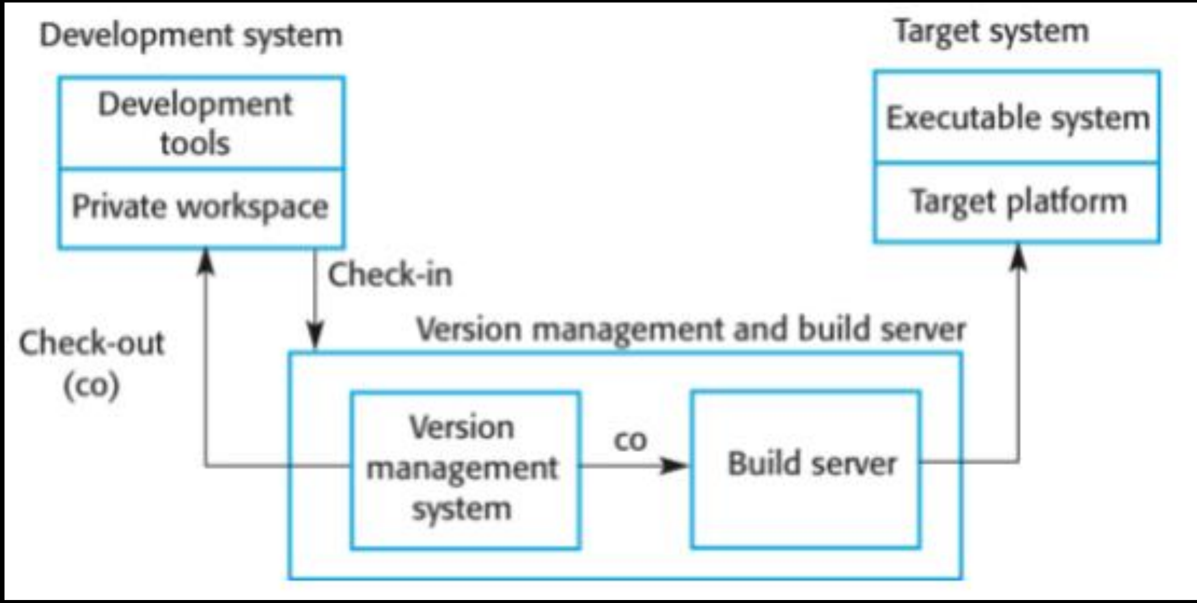
System Building

System Building

- Process of creating a complete, executable system
  - By compiling and linking the system components, external libraries, configuration files, etc.
- Needs to **communicate with VC tools** as the build process involves checking out component versions

Building Platforms

- **Development system**: includes compilers, source code editors. Developers check out code from the version management system into a private workspace before making changes to the system.
- **Build server**: used to build definitive, executable versions of the system. The system build may rely on external libraries that are not included in the version management system.
- **Target environment**: the platform on which the system executes.



Building System Functionality

- Build script generation
- Version management system integration
- Minimal re-compilation
- Executable system creation
- Test automation
- Reporting
- Documentation generation

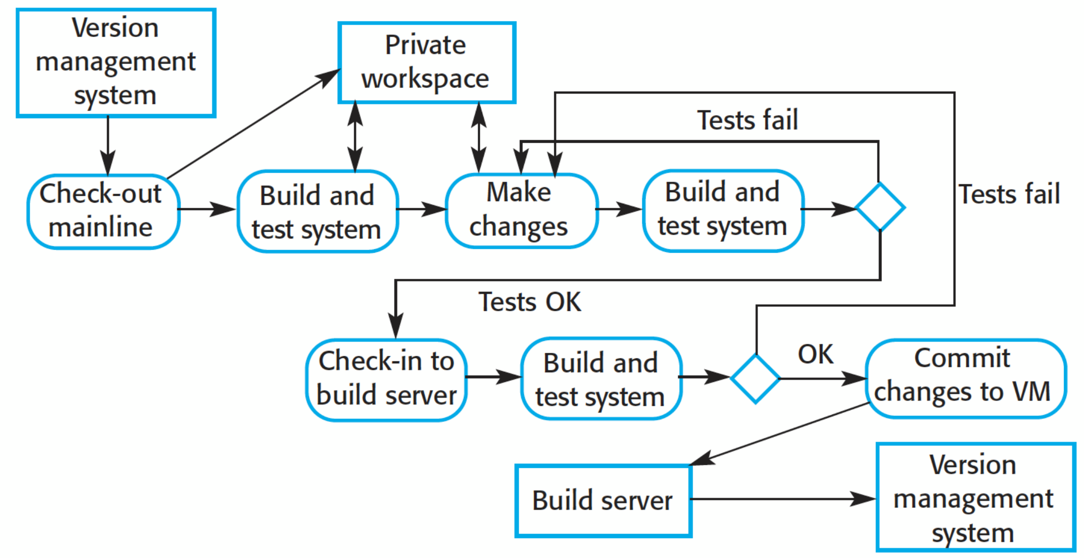
Continuous Integration

- **Practice** of **automating** the **integration of code changes** from **multiple contributors** into a **single software project**.
- Allow developers to frequently merge code changes into a central repository. (Grady Booch, 1991)

Continuous Integration: Workflow

- **Check out the mainline system** from the version management system into the developer's private workspace.
- **Build the system** and **run automated tests** to ensure that the built system passes all tests. If not, the build is broken and you should inform whoever checked in the last baseline system. They are responsible for repairing the problem.
- **Make the changes** to the system components.
- **Build the system** in the **private workspace** and **re-run system tests**. If the tests fail, continue editing.
- Once the system has passed its tests, **check it into the build system** but **do not commit it** as a new system baseline.
- **Build the system** on the **build server** and run the tests. You need to do this in case others have modified components since you checked out the system. If this is the case, check out the components that have failed and edit these so that tests pass on your private workspace.
- If the system passes its tests on the build system, then **commit the changes** you have made as **a new baseline** in the system mainline.

Figure 25.12 Continuous integration



Continuous Integration

- Pros:
  - It allows problems caused by the interactions between different developers to be discovered and repaired as soon as possible.
  - The most recent system in the mainline is the definitive working system. (Agile building process)
- Cons:
  - If the system is very large, it may take a long time to build and test, especially if integration with other application systems is involved.
  - If the development platform is different from the target platform, it may not be possible to run system tests in the developer's private workspace.

Change Management

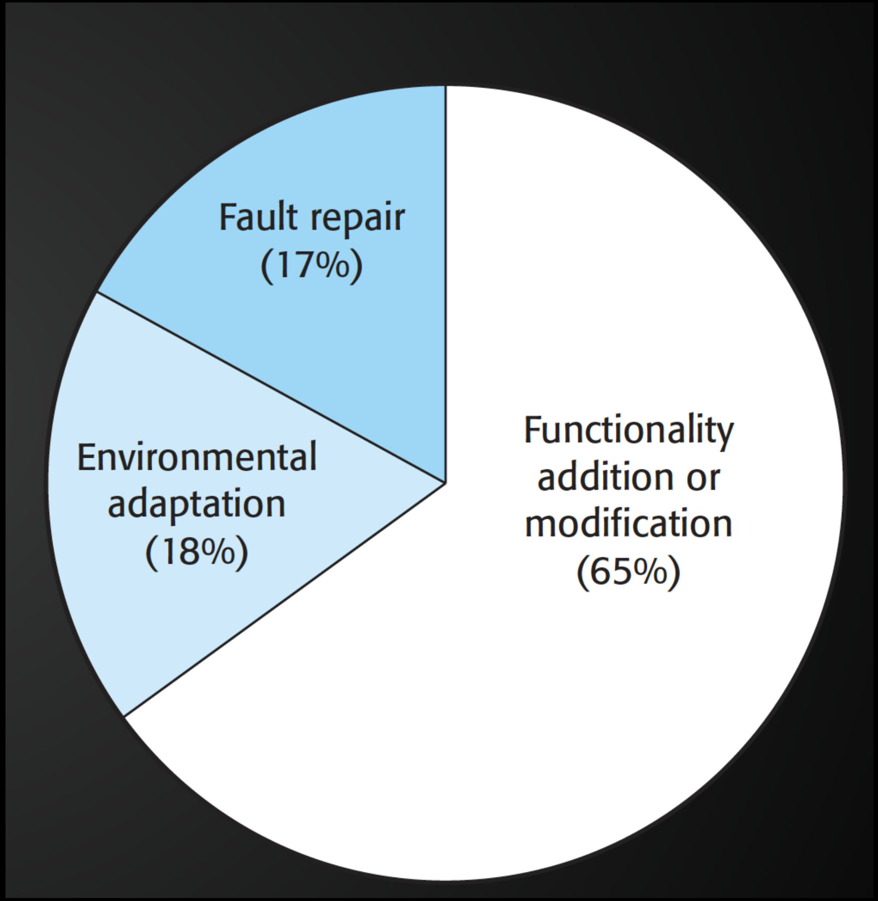
Change Management

- Ensure that system evolution is a managed process and that priority is given to the **most urgent** and **cost-effective changes**.
  - Analyze the costs and benefits of proposed changes, approve those changes that are worthwhile and tracking which components in the system have been changed.
- Factors in change analysis:
  - The consequences of not making the change
  - The benefits of the change
  - The number of users affected by the change
  - The costs of making the change
  - The product release cycle

University of Nottingham UK | CHINA | MALAYSIA

Types of Change

- **Fault Repairs** (Cheap)
  - To fix coding errors
  - Do not involve much redesign
- **Environmental Adaption** (Cheap)
  - e.g. updates for new OS
  - Do not involve much redesign
- **Functionality Addition** (Expensive)
  - To meet business changes
  - Often involves redesign



Emergency Changes

- Often problems occur that require fast or emergency changes - and updating the code takes priority over documentation

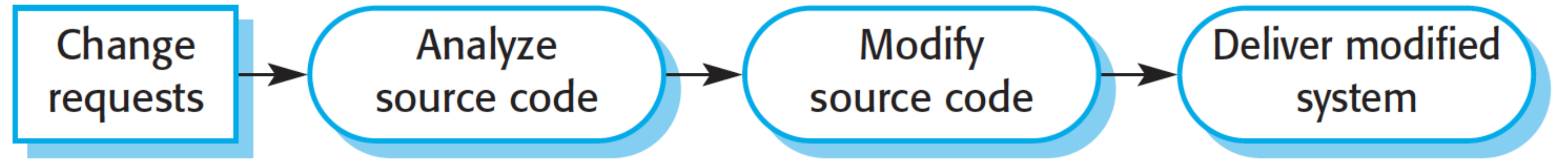


Figure 9.6 The emergency repair process

- It's best to **document emergency changes**
  - Later, proper solutions to changes can be designed
- The real danger:
  - Multiple subsequent emergency repairs occur

Release Management

Release Management

- Release:
  - A version of a software system that is distributed to customers.
- Mass market software
  - Major releases, deliver significant new functionality.
  - Minor releases, repair bugs and fix customer problems that have been reported.
- A release may include:
  - Configuration files - how the release should be configured
  - Data files
  - An installation program
  - Paper documentation

Release Planning (Factors)

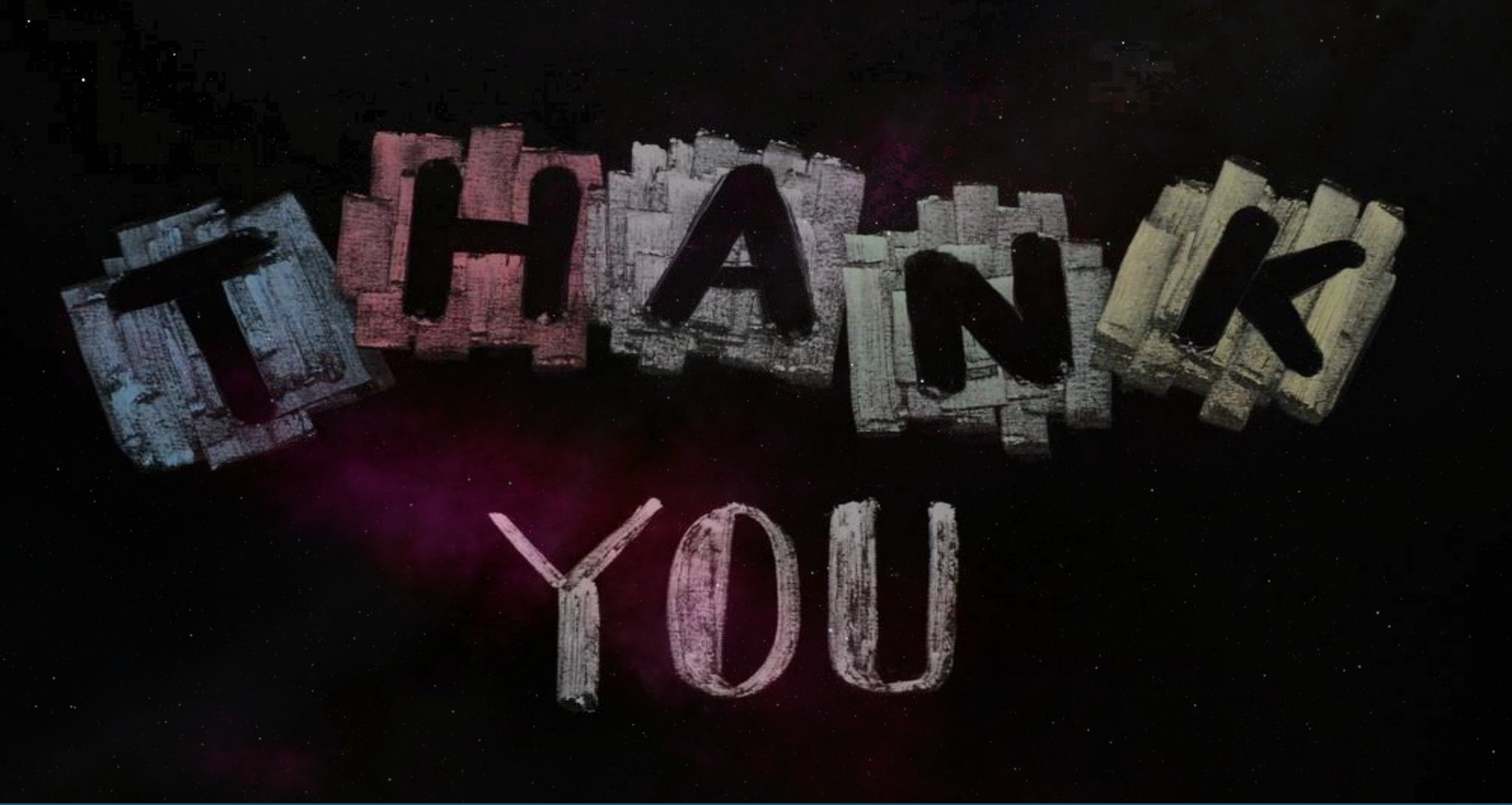
- **Competition**:
  - For mass-market software, a new system release may be necessary because a competing product has introduced new features and market share may be lost if these are not provided to existing customers.
- **Marketing requirements**:
  - The marketing department of an organization may have made a commitment for releases to be available at a particular date.
- **Platform changes**:
  - If a new version of the operating system platform is released.
- **Technical quality of the system**:
  - If serious system faults are reported.

Release Creation Process

1. The **executable code** of the programs and **all associated data files** must be identified in the version control system.
2. **Configuration descriptions** may have to be written for different hardware and operating systems.
3. **Update instructions** may have to be written for customers who need to configure their own systems.
4. Scripts for the installation program may have to be written.
5. Web pages have to be created **describing the release**, with links to system documentation.
6. When all information is available, **an executable master image** of the software must be prepared and handed over for distribution to customers or sales outlets.

Summary

- Version Control
  - VC systems
  - VC branches
- System Building
  - Building platforms, functionalities
  - Continuous Integration
- Change Management
- Release Management
  - Release planning factors
  - Release creation process


THANK
YOU