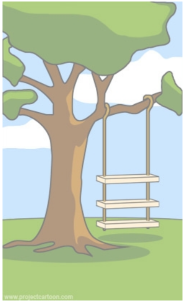
Software Engineering

13. Configuration Management | Thomas Thüm | May 10, 2022

Software Engineering
Programming Languages

ulm university universität ulm

Configuration Management

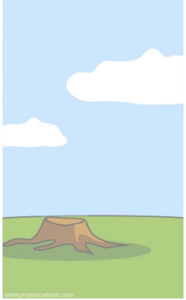


how the customer explained it

SECURITY SECURITY SECURITY SECURITY SECURITY SECURITY SECURITY
www.projectcartoon.com

how patches were applied

software evolution



how it was supported

software maintenance

www.projectcartoon.com

when it was delivered

configuration management

Lessons Learned

- Legacy software: typical properties, examples
- Software migration
- Migration strategies: wrapping, redevelopment, incremental migration, big bang migration
- Further Reading: Ludewig and Lichter, Chapter 23 (Reengineering)

Practice

- 1. Post an example of software in Moodle that reached its end of life
- 2. Interpret the reasons for an example by a colleague
- Will be discussed in next lecture
- Deadline: May 9 at 12 noon

Lecture Overview

1. Configuration Management and Version Control
2. Operations of Version Control Systems
3. Continuous Integration and Deployment

Lecture Contents

1. Configuration Management and Version Control
   - Which Products are Affected?
   - Configuration Management
   - Version Control
   - Git vs Mercurial
   - Centralized Version Control
   - Distributed Version Control
   - Lessons Learned
2. Operations of Version Control Systems
3. Continuous Integration and Deployment

Which Products are Affected?

Configuration Management [Sommerville]

Configuration Management

“**Configuration management** is concerned with the policies, processes, and tools for managing changing software systems.”

Four Activities in Configuration Management

- version control / management: keeping track of multiple versions, enabling simultaneous changes
- system building: collecting, compiling, and linking components into executable systems
- change management: tracking change requests and planning if/when realized
- release management: preparing new and managing old releases

Development Stages (Entwicklungsphasen)

- **development phase**: adding new functionality
- **system testing phase**: internal release, bug fixes, performance improvements, security fixes, but no new functionality
- **release phase**: bug reports and feature requests by users or customers
- often several versions co-exist at different stages

Version Control

Goals of Version Control

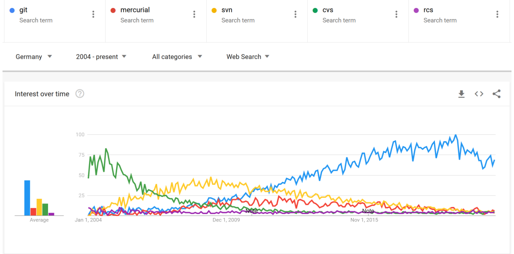
- collaborative work: synchronize files and folders with other users
- compare own version with other versions
- merge files edited by several users
- history: access old versions, log changes

Version Control Systems

- local only: SCCS (1972), RCS (1982), . . .
- centralized: CVS (1986), SVN (2000), . . .
- distributed: git (2005), mercurial (2005), . . .

invented for software, useful for most files

Alternatives? Why not use locks (cf. databases)?



git Search term
mercurial Search term
svn Search term
cvs Search term
rcs Search term

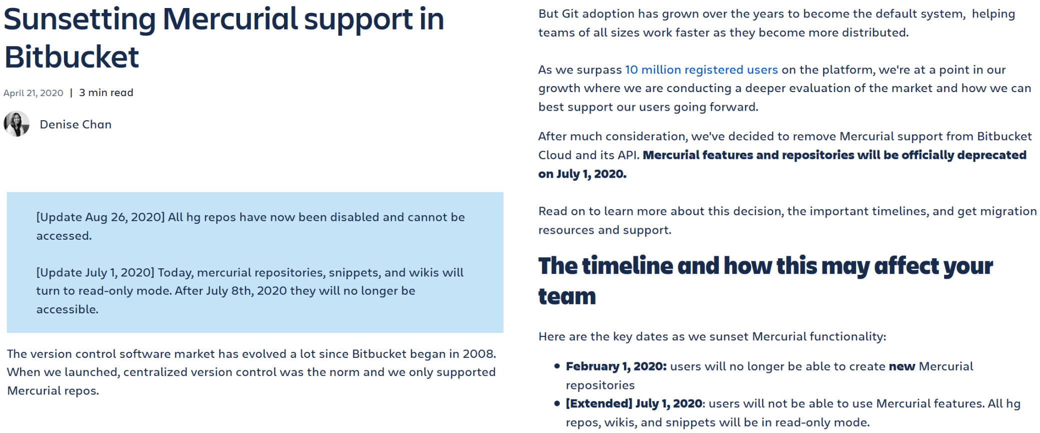
Germany ▾ 2004 - present ▾ All categories ▾ Web Search ▾

Interest over time

Personal Experience

- 2004: started working with CVS in industry
- 2007: initiated research protoype FeatureIDE with SVN
- 2009: published FeatureIDE open source (history neglected)
- 2014: migrated FeatureIDE's code to git

Git vs Mercurial



Sunsetting Mercurial support in Bitbucket

April 21, 2020 | 3 min read

Denise Chan

> [Update Aug 26, 2020] All hg repos have now been disabled and cannot be accessed.
>
> [Update July 1, 2020] Today, mercurial repositories, snippets, and wikis will turn to read-only mode. After July 8th, 2020 they will no longer be accessible.

The version control software market has evolved a lot since Bitbucket began in 2008. When we launched, centralized version control was the norm and we only supported Mercurial repos. But Git adoption has grown over the years to become the default system, helping teams of all sizes work faster as they become more distributed.

As we surpass 10 million registered users on the platform, we're at a point in our growth where we are conducting a deeper evaluation of the market and how we can best support our users going forward.

After much consideration, we've decided to remove Mercurial support from Bitbucket Cloud and its API. **Mercurial features and repositories will be officially deprecated on July 1, 2020.**

Read on to learn more about this decision, the important timelines, and get migration resources and support.

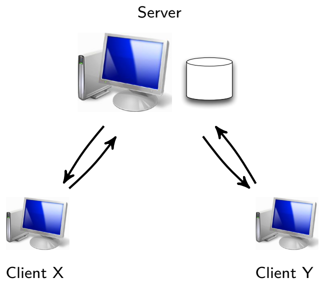
The timeline and how this may affect your team

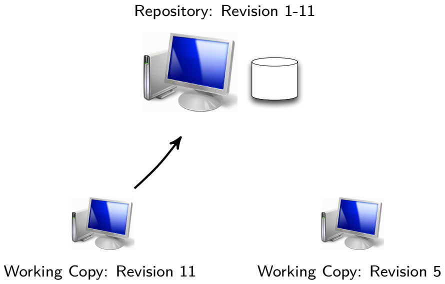
Here are the key dates as we sunset Mercurial functionality:

- **February 1, 2020:** users will no longer be able to create **new** Mercurial repositories
- **[Extended] July 1, 2020**: users will not be able to use Mercurial features. All hg repos, wikis, and snippets will be in read-only mode.

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Centralized Version Control





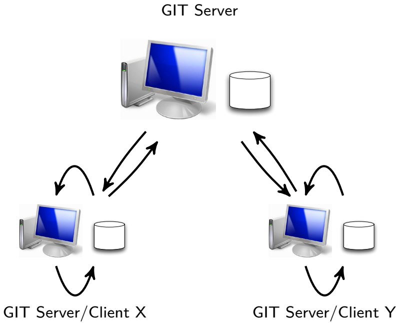
Centralized Version Control Systems

- client-server architecture
- server manages the main copy
- clients use server to synchronize files/folders

Centralized Version Control Systems

- repository on server
- working copy on each client
- new revision for every change on the server
- revisions used to undo changes, merge files

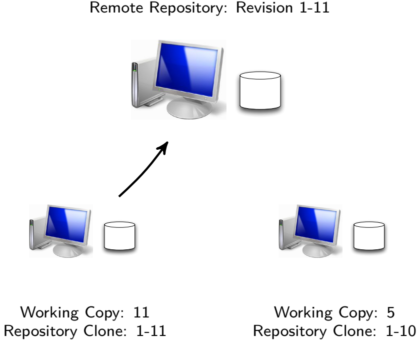
Distributed Version Control



GIT Server

GIT Server/Client X

GIT Server/Client Y



Remote Repository: Revision 1-11

Working Copy: 11
Repository Clone: 1-11

Working Copy: 5
Repository Clone: 1-10

Distributed Version Control Systems

- peer-to-peer architecture
- clients use repository clone to synchronize
- version history available on each client

Distributed Version Control Systems

- main repository on one or several servers
- clone of the repository on each client
- new revision on every change on the clients

Configuration Management and Version Control

Lessons Learned

- Configuration management: 4 activities and 3 development stages
- Version control: goals, centralized and distributed
- Further Reading: Sommerville, Chapter 25 Configuration Management

Practice

- Github live demo
- Explore another project on Github
- Share your insights in Moodle

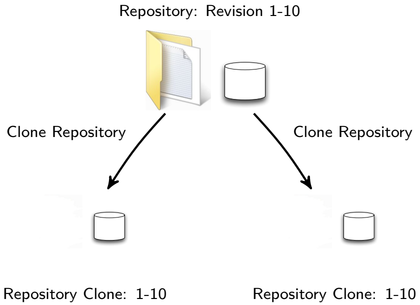
Lecture Contents

1. Configuration Management and Version Control

2. Operations of Version Control Systems
   Clone and Fetch
   Commit and Push
   How to Write Good Commit Messages?
   Example: Thomas' Calculator
   Merge and Pull
   Ignore
   Branching & Merging
   Automatic Merge
   Git Merge in Pictures
   Merge Conflicts
   Lessons Learned

3. Continuous Integration and Deployment

Clone and Fetch



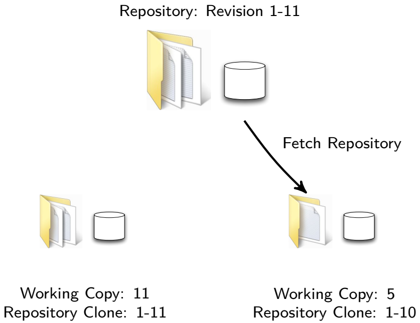
Repository: Revision 1-10

Clone Repository

Clone Repository

Repository Clone: 1-10

Repository Clone: 1-10



Repository: Revision 1-11

Fetch Repository

Working Copy: 11
Repository Clone: 1-11

Working Copy: 5
Repository Clone: 1-10

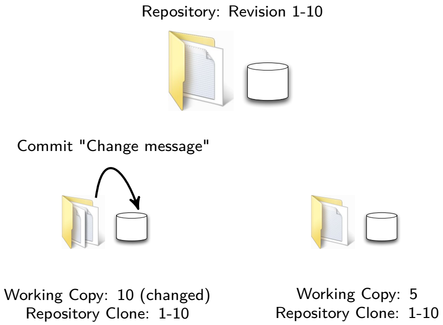
Clone

- create a local repository clone
- use local repository to create a working copy
- specify folders for the clone
- specify revision or head (latest revision)

Fetch

- download the remote repository
- note: working copy remains unchanged

Commit and Push

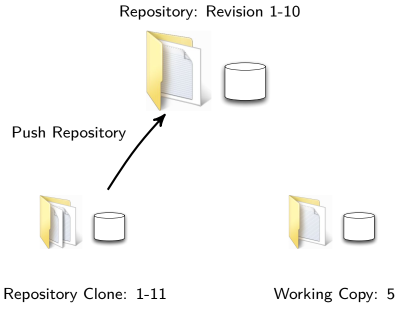


Repository: Revision 1-10

Commit "Change message"

Working Copy: 10 (changed)
Repository Clone: 1-10

Working Copy: 5
Repository Clone: 1-10



Repository: Revision 1-10

Push Repository

Repository Clone: 1-11

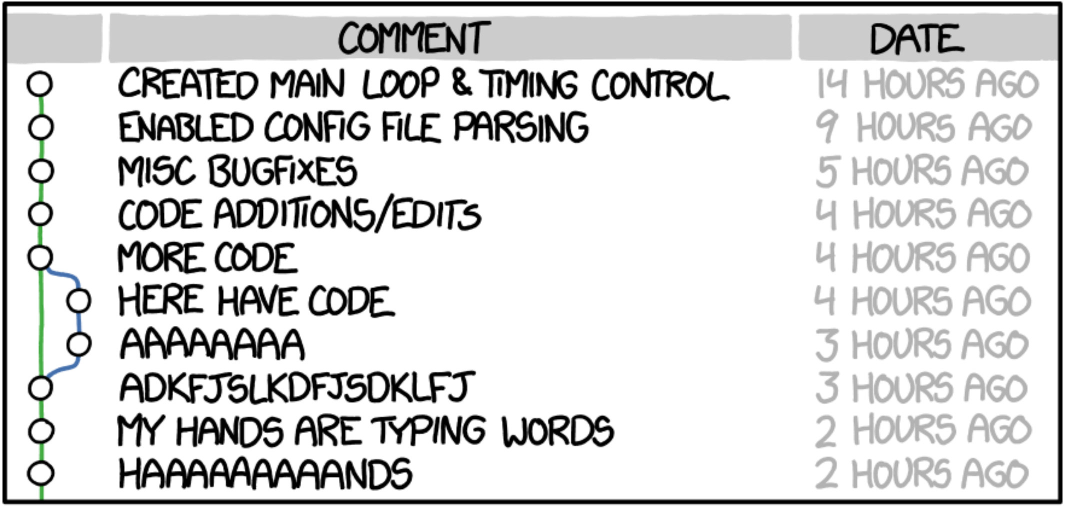
Working Copy: 5

Commit

- commit changes to local repository
- commits are atomic (all or nothing)
- specify changed folders and files
- mandatory message describing your change

Push

- push local repository to server
- condition: local repository is up to date (might require previous pull)



COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

How to Write Good Commit Messages?

Structure of Commit Messages [github.com]

Subject Line (required)

- short summary (72 chars or less)
- should complete the following sentence: “If applied, this commit will . . . ”

Message Body (optional)

- blank line followed by message body
- explain what has changed and why

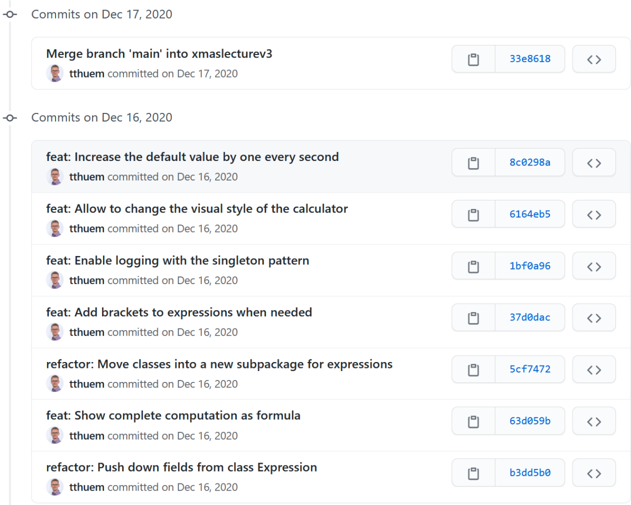
Integration with Issues [github.com, gitlab.com]

- write **#42** to refer to issue – Github/Gitlab will create links in both ways
- **close/fix/resolve/. . . #42** – issue automatically closed when commit is pushed to default branch

Conventional Commits [conventionalcommits.org]

- Machine readable subject line
- **fix:** patches a bug (**patch**)
- **feat:** introduces a new feature (**minor** change)
- **BREAKING CHANGE:** introduces a breaking API change (**major** change)
- **refactor:** applies a refactoring
- . . .
- semantic versioning: major.minor.patch (e.g., v2.6.0)

Example: Thomas' Calculator



Commits on Dec 17, 2020

Merge branch 'main' into xmaslecturev3
tthuem committed on Dec 17, 2020
33e8618

Commits on Dec 16, 2020

feat: Increase the default value by one every second
tthuem committed on Dec 16, 2020
8c0298a

feat: Allow to change the visual style of the calculator
tthuem committed on Dec 16, 2020
6164eb5

feat: Enable logging with the singleton pattern
tthuem committed on Dec 16, 2020
1bf0a96

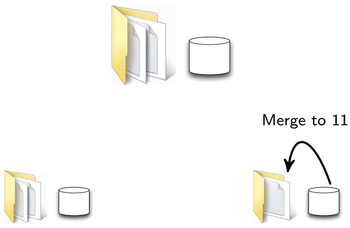
feat: Add brackets to expressions when needed
tthuem committed on Dec 16, 2020
37d0dac

refactor: Move classes into a new subpackage for expressions
tthuem committed on Dec 16, 2020
5cf7472

feat: Show complete computation as formula
tthuem committed on Dec 16, 2020
63d059b

refactor: Push down fields from class Expression
tthuem committed on Dec 16, 2020
b3dd5b0

Merge and Pull

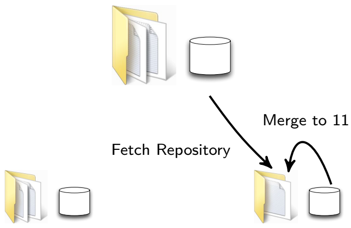


Remote Repository: Revision 1-11

Merge to 11

Working Copy: 11
Repository Clone: 1-11

Working Copy: 5
Repository Clone: 1-11



Remote Repository: Revision 1-11

Fetch Repository

Merge to 11

Working Copy: 11
Repository Clone: 1-11

Working Copy: **11**
Repository Clone: 1-**11**

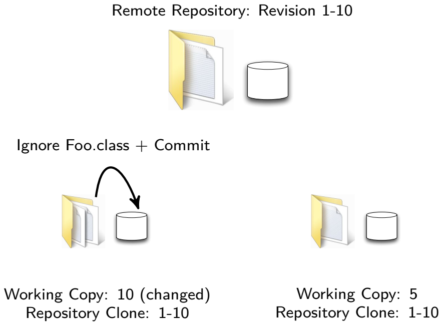
Merge

- update working copy with local repository
- integrate new commits from other branch
- fast forward / automatic merge / manual merge

Pull

- pull = fetch + merge
- download and merge in one step!

Ignore



Remote Repository: Revision 1-10

Ignore Foo.class + Commit

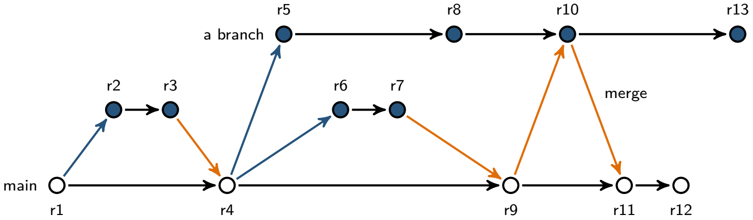
Working Copy: 10 (changed)
Repository Clone: 1-10

Working Copy: 5
Repository Clone: 1-10

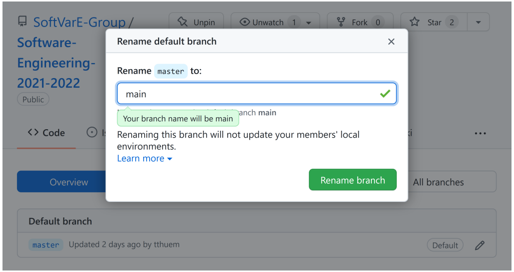
.gitignore

- resources will be ignored on commit
- user-specific files or derived/generated resources
- specify files and folders (e.g., *.log files)

Branching & Merging



- simultaneous, independent development
- option to merge in the future
- `main/` – main development
- `branch/*/` – parallel developments



Rename default branch

Rename `master` to:

main

anch main

Your branch name will be main

Renaming this branch will not update your members' local environments.

Learn more

Rename branch

Default branch

`master` Updated 2 days ago by tthuem

Default

Automatic Merge

Working Copy: Revision **11***

```
class Foo {
   void bar() {
      print("Foo");
      print("Bar");
      print("\n");
   }
}
```

Repository: Revision **11**

```
class Foo {
   void bar() {
      print("Foo");
      print("Bar");
   }
}
```

- Checkout of or update to revision 10
- Changing the working copy (**10***)
- In the meantime: New commit to repository (**11**)
- Update to head revision, automatically merged

Git Merge in Pictures

Merge Conflicts

Working Copy: Revision 11*

```
class Foo {
   void bar() {
      <<<<<<< .mine
      print("Bar\n");
      =======
      print("FooBar");
      >>>>>>> .r11
   }
}
```

Repository: Revision 11

```
class Foo {
   void bar() {
      print("FooBar");
   }
}
```

- Checkout of or update to revision 10
- Changing the working copy (10*)
- In the meantime: New commit to repository (11)
- Update to head revision results in conflict
- Automatic merge fails: user has to provide a fix

Operations of Version Control Systems

Lessons Learned

- Version control with clone, fetch, commit, push, merge, pull, ignore
- Commit messages
- Branching, merging, merge conflicts

Practice

- Quiz'n'Disquiz
- Quiz: fill out the Moodle quiz on your own
- Disquiz: compare and discuss the results with your colleagues

Lecture Contents

1. Configuration Management and Version Control

2. Operations of Version Control Systems

3. Continuous Integration and Deployment

   System Building

   Continuous Integration

   DevOps

   Lessons Learned

System Building [Sommerville]

System Building

"**System building** is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, and other information."

Building Involves Three Platforms

- **development system**: compilers and editors used on the developer's system to test prior to commit
- **build server**: server to build and distribute executable versions, triggered by commits or schedule (i.e., nightly builds)
- **target environment**: intended platform for executable system (e.g., ECU in a car)

Tooling for Building and System Integration

- build script generation: identify dependent components, automated generation or tool support for creation and editing
- version control system integration: checkout required versions of components
- minimal recompilation: determine which parts need to be recompiled
- executable system creation: compilation and linking
- test automation: run automated tests (e.g., unit tests)
- reporting: reports about success or failure of builds and tests
- documentation generation: release notes, help pages

Continuous Integration [Sommerville]

Continuous Integration

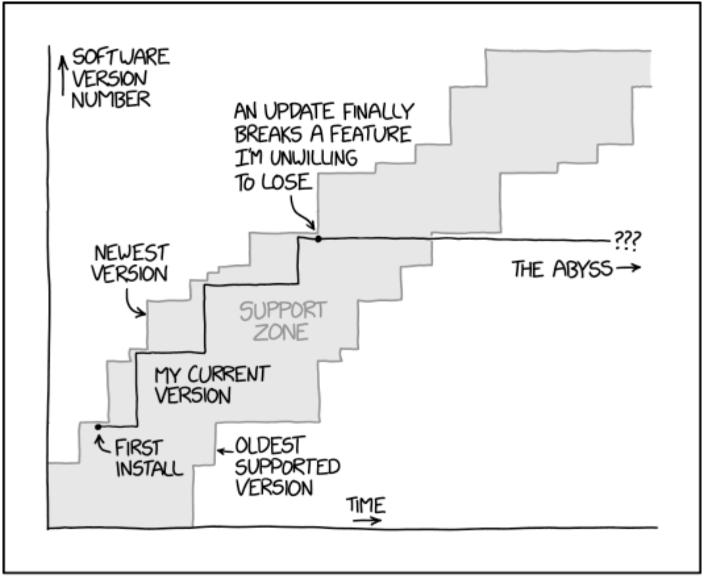
"Agile methods recommend that very frequent system builds should be carried out, with automated testing used to discover software problems. Frequent builds are part of a process of continuous integration [...]."

Continuous Integration Tools

- CruiseControl (2001–2010) — first open-source tool
- TeamCity (2006–)
- Hudson (2008–2017)
- Jenkins (2011–) — fork of Hudson
- Travis CI (2011–) — made in Germany
- GitLab (2014–)

Steps in Continuous Integration

- clone/fetch from version control
- if feasible: build and run automated tests, if it fails others are responsible
- apply changes
- build and run automated tests locally, if it fails continue editing
- if local tests pass, commit to feature branch in version control
- commit triggers build server, if it fails continue editing
- if tests pass (and code review approves changes), merge branch into main development branch



ALL SOFTWARE IS SOFTWARE AS A SERVICE.

DevOps [Krypczyk/Bochkor]

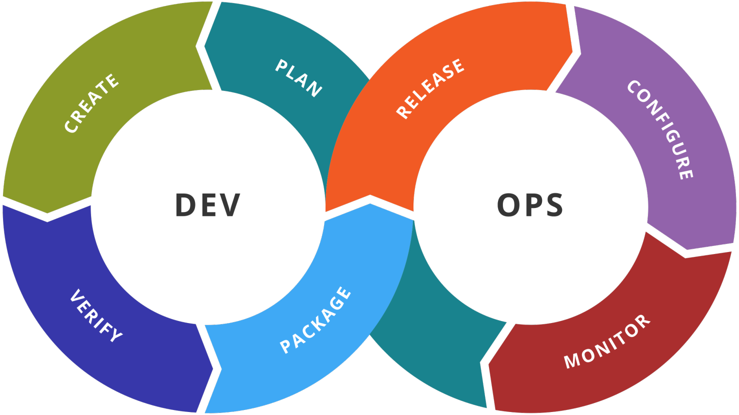
Motivation

- if software fails:
  - programmers blame administrators for misconfiguration
  - administrators blame programmers for erroneous software
- programmers want frequent updates
- administrators follow the slogan: "never change a running system"
- customers and users want a single responsibility
- shorter and shorter update cycles

DevOps

- promoted in agile development
- **Dev**: development by programmers
- **Ops**: operation (Betrieb) by administrators
- **DevOps**: teams that are responsible for both, development and operations
- goal: avoid blaming each other by shared responsibility

DevOps


CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR

Continuous Integration and Deployment

Lessons Learned

- System building: 3 platforms, requirements on tooling
- Continuous integration: tools, steps
- DevOps
- Further Reading: Sommerville, Chapter 25 Configuration Management and Krypczyk/Bochkor, Chapter 9.3 DevOps

Practice

- Form groups of 2–3 students
- 1. Discuss flavors of continuous integration
- 2. Report one flavor in Moodle