

# Software Engineering Project Workshop (SENG202)

Matthias Galster

Technical setup

July 20, 2020

# Reminders

- Time left for Phase 1

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
July 20	July 21	July 22	July 23	July 24	July 25	July 26
July 27	July 28	July 29	July 30	July 31	August 1	August 2
August 3						

- Submit your weekly individual reflection (Mondays, 5:00pm)
- Keep logging as you go
  - Check again course notes from last week
    - Set your profile as instructed – otherwise won't process your data
    - Use project (do not log "Without project"), use tags
    - Etc.

# Deliverables

- Project setup checklist
- Design document
- Reflections and logs
- Presentation



# Tools to address typical SE problems

Multi-person construction  
of multi-version software

- Build automation and configuration management

- Ant, make, **Maven**, etc. 

- Revision and version control systems


- SVN, Mercurial, **Git**, etc.



- Test frameworks and quality assurance

- **JUnit**, **Cucumber**, TestNG, jMock, Mockito, etc.



- Style checker (e.g., checkstyle), static analysis, etc. 

- Continuous integration

- **GitLab CI**, CruiseControl, Jenkins, etc.



# Technical tutorials

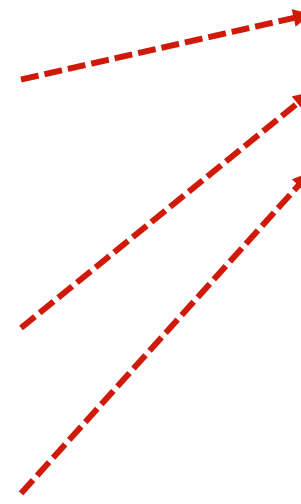
Week	Session and quiz
1	n/a
2	Maven
3	Git 1
4	Unit testing/TDD
5	JavaFX
6	Continuous integration
7	Git 2
8	Acceptance testing
9	
10	
11	
12	



Tutorial package 1

Tutorial package 2

Tutorial package 3



Week	Release
1	
2	Package 1
3	Package 2
4	Package 3
5	
6	
7	
8	
9	
10	
11	
12	

# What is a build?



- Build process
  - Compile into executable binary instructions
  - Link to any dependencies (e.g., libraries)
  - Test
  - Package into .jar
  - Generate API documentation
  - Deploy to website
  - Etc.

# Configuration management



- What resources are in the build?
  - Where are they located (absolute, relative)?
  - How are they related?
  - How are they produced?
  - What should happen when they change?
  - How do we know everything works properly?
  - How do we enforce our coding style?

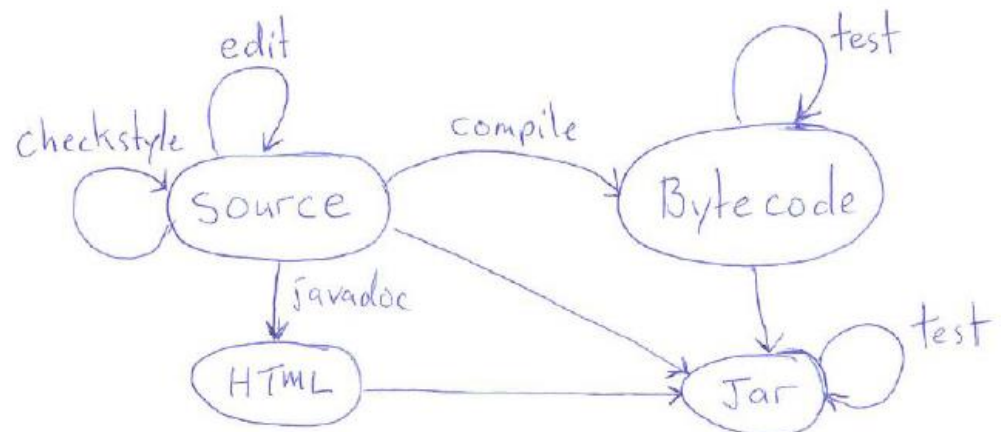
# Dependencies



- When a resource becomes out of date
  - Update other resources
  - Specify order, paths, options, etc.
  - Chain of consequences
  - **Avoid unnecessary work (builds are time-consuming)**

- Example: when a .java file changes

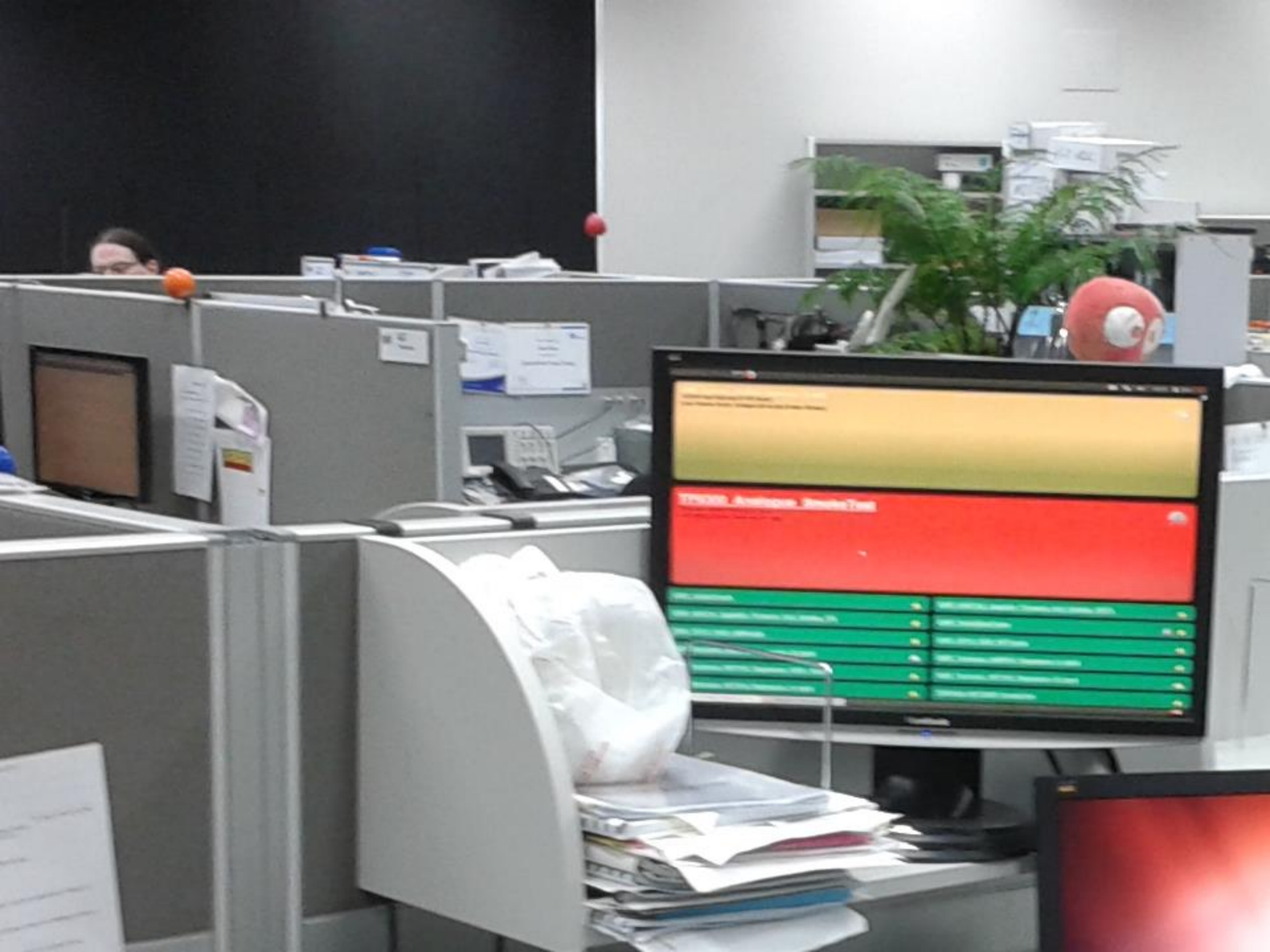
- Re-compile
- Re-run style check
- Re-run unit test
- Check in to repository
- Re-run integration tests
- Re-build application





# Builds in industry

- At Google, Java developers build 7 times a day
  - 30% fail
- On average, C++ developers in industry build 10 times a day
  - 40% fail



Yellow and Red Header	
TABLE: Antiqua, Roma, T...	
1. Antiqua, Roma, T...	2. Antiqua, Roma, T...
3. Antiqua, Roma, T...	4. Antiqua, Roma, T...
5. Antiqua, Roma, T...	6. Antiqua, Roma, T...
7. Antiqua, Roma, T...	8. Antiqua, Roma, T...
9. Antiqua, Roma, T...	10. Antiqua, Roma, T...
11. Antiqua, Roma, T...	12. Antiqua, Roma, T...
13. Antiqua, Roma, T...	14. Antiqua, Roma, T...
15. Antiqua, Roma, T...	16. Antiqua, Roma, T...
17. Antiqua, Roma, T...	18. Antiqua, Roma, T...
19. Antiqua, Roma, T...	20. Antiqua, Roma, T...
21. Antiqua, Roma, T...	22. Antiqua, Roma, T...
23. Antiqua, Roma, T...	24. Antiqua, Roma, T...
25. Antiqua, Roma, T...	26. Antiqua, Roma, T...
27. Antiqua, Roma, T...	28. Antiqua, Roma, T...
29. Antiqua, Roma, T...	30. Antiqua, Roma, T...
31. Antiqua, Roma, T...	32. Antiqua, Roma, T...
33. Antiqua, Roma, T...	34. Antiqua, Roma, T...
35. Antiqua, Roma, T...	36. Antiqua, Roma, T...
37. Antiqua, Roma, T...	38. Antiqua, Roma, T...
39. Antiqua, Roma, T...	40. Antiqua, Roma, T...
41. Antiqua, Roma, T...	42. Antiqua, Roma, T...
43. Antiqua, Roma, T...	44. Antiqua, Roma, T...
45. Antiqua, Roma, T...	46. Antiqua, Roma, T...
47. Antiqua, Roma, T...	48. Antiqua, Roma, T...
49. Antiqua, Roma, T...	50. Antiqua, Roma, T...
51. Antiqua, Roma, T...	52. Antiqua, Roma, T...
53. Antiqua, Roma, T...	54. Antiqua, Roma, T...
55. Antiqua, Roma, T...	56. Antiqua, Roma, T...
57. Antiqua, Roma, T...	58. Antiqua, Roma, T...
59. Antiqua, Roma, T...	60. Antiqua, Roma, T...
61. Antiqua, Roma, T...	62. Antiqua, Roma, T...
63. Antiqua, Roma, T...	64. Antiqua, Roma, T...
65. Antiqua, Roma, T...	66. Antiqua, Roma, T...
67. Antiqua, Roma, T...	68. Antiqua, Roma, T...
69. Antiqua, Roma, T...	70. Antiqua, Roma, T...
71. Antiqua, Roma, T...	72. Antiqua, Roma, T...
73. Antiqua, Roma, T...	74. Antiqua, Roma, T...
75. Antiqua, Roma, T...	76. Antiqua, Roma, T...
77. Antiqua, Roma, T...	78. Antiqua, Roma, T...
79. Antiqua, Roma, T...	80. Antiqua, Roma, T...
81. Antiqua, Roma, T...	82. Antiqua, Roma, T...
83. Antiqua, Roma, T...	84. Antiqua, Roma, T...
85. Antiqua, Roma, T...	86. Antiqua, Roma, T...
87. Antiqua, Roma, T...	88. Antiqua, Roma, T...
89. Antiqua, Roma, T...	90. Antiqua, Roma, T...
91. Antiqua, Roma, T...	92. Antiqua, Roma, T...
93. Antiqua, Roma, T...	94. Antiqua, Roma, T...
95. Antiqua, Roma, T...	96. Antiqua, Roma, T...
97. Antiqua, Roma, T...	98. Antiqua, Roma, T...
99. Antiqua, Roma, T...	100. Antiqua, Roma, T...

# Revision (version) control systems



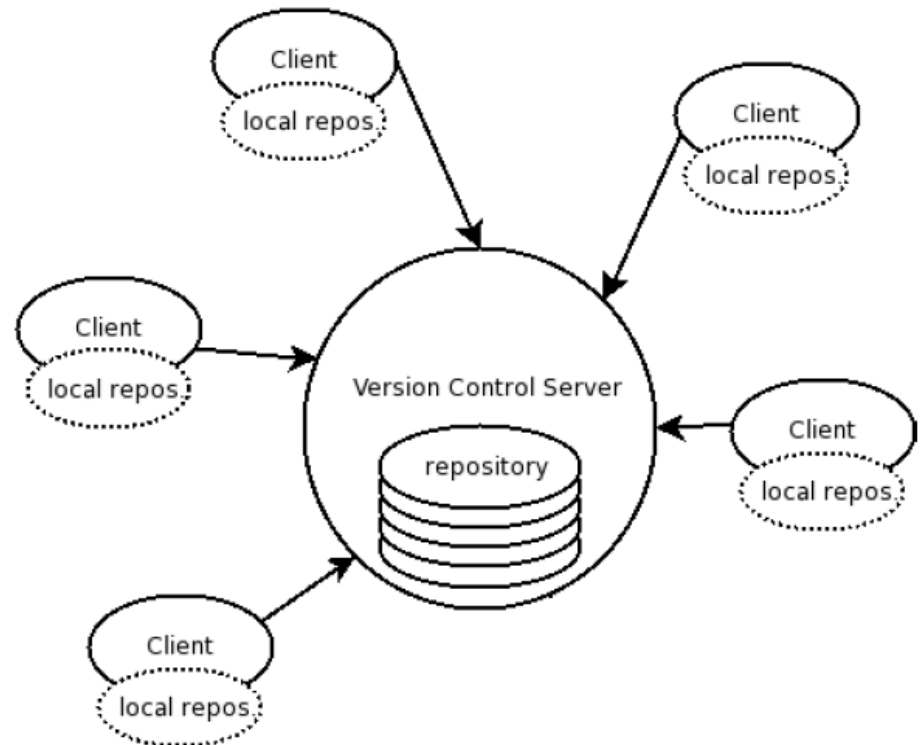
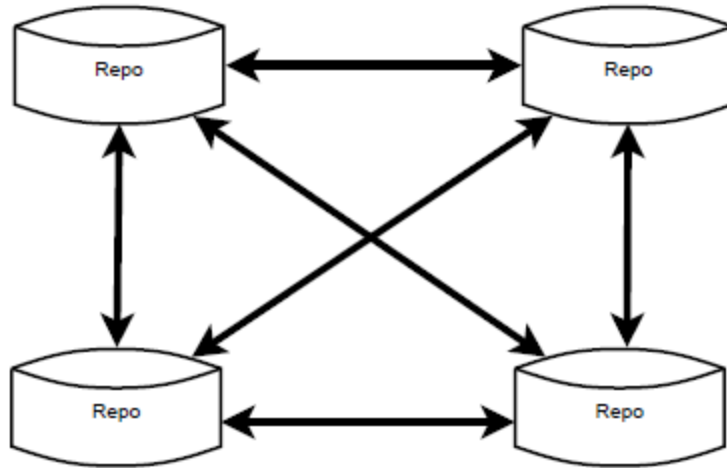
- Help manage different version of different users
  - Source code and other resources placed in “shared” DB
- Developers “check out” local copy
  - May “check in” updated copy
  - Subsequent developers will “check out” updated copy
- Access mediated by software
  - Subversion, CVS, SourceSafe, Team Foundation Server, **Git**, Mercurial, RCS, ClearCase, etc.

# Conflicts



- Blue skies flow
  - Sequence of individual changes are managed
- In reality
  - Developers may be different, or same people at different times
  - More than one developer has working copy of a file and updates it
- Revision control system must be able to support
  - Merge compatible changes (e.g., in different methods)
    - Successful merge doesn't mean system will pass tests
  - Resolve conflicting changes (e.g., in same method)
    - User intervention for conflict resolution

# Distributed / central architecture



# Differences between version control systems



- Main differences
  - Server-based or peer-to-peer (distributed versus centralized architecture)
  - Speed, functionality, learning curve
- Example: Git
  - Speed (especially on Linux systems); powerful, variety of tools (e.g., full history tree available offline); keeps track of contents, even small changes tracked as change; distributed, peer-to-peer model, everyone has own repository; suitable for many users
  - Learning curve; not optimal for single developers; limited Windows support compared to Linux; more commands, more options, more complex
- Example: SVN
  - No peer-to-peer model, central repository, client-server model; many plug-ins for IDE's
  - Slower comparative speed; all changes saved in single location
- Example: Mercurial
  - Learning curve; documentation; Windows support; fast if small team; distributed
  - Less out of the box power

# Continuous integration



- Shared code base is merged
  - Must always **build** and **pass tests**
  - Team members integrate their work frequently (daily?)
- Many small changes – **continuous** iterations, not “big bang”
  - **Automate complex build activities**
  - Developers test own changes, regression tests
  - System level integration
  - **Early warning and feedback**