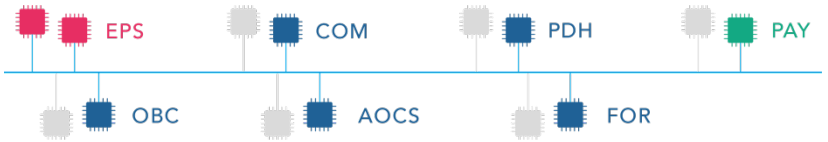
PRACTICAL EXPERIENCE IN USING CONTINUOUS INTEGRATION WITHIN THE DEVELOPMENT OF A NANOSATELLITE SOFTWARE

Karsten Gordon | Chair of Space Technology | 10th IAA Symposium, April 20-24 2015, Berlin

TUBiX20 missions

- Single-failure-tolerance and modularity
- Distributed, redundant nodes
- Centralized power- and data bus
- Reuse of hard- and software on different levels
  → common infrastructure



TechnoSat

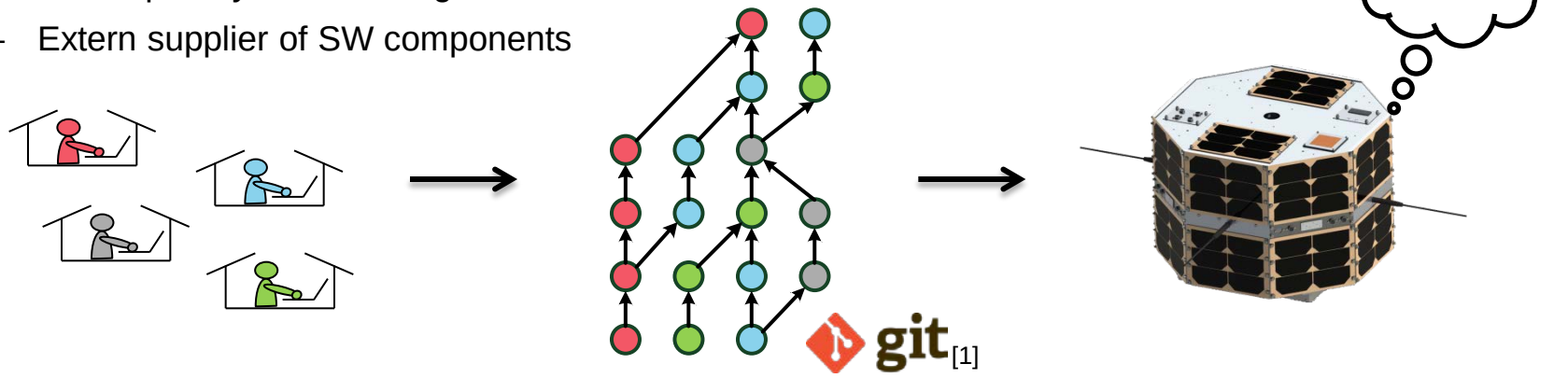
- Technology demonstration mission
- Launch in H1 2016

TUBIN

- Earth observation mission
- Launch in H1 2017

Concurrent Software Development

- Network of nodes with common infrastructure
- Developed by different engineers
- Extern supplier of SW components



Continuous Integration

- Well-structured version control management practice
- Frequent integration of new features into the mainline
- Automated analysis, build, test and deploy

→ Integration as series of small steps instead of single complex task
→ Early tests make software more robust reduces debugging time
→ Progress becomes more transparent for planning and evaluation

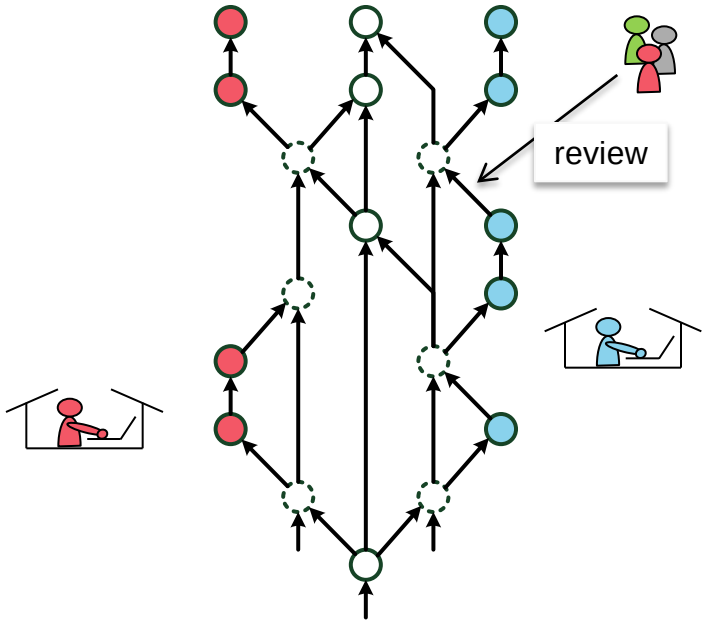
Version control system organization

Private branches

- Scratchpad for feature development
- Collision-free push/pull → push-on-logout
- Rewriting commit history → “hack now, clean up later”

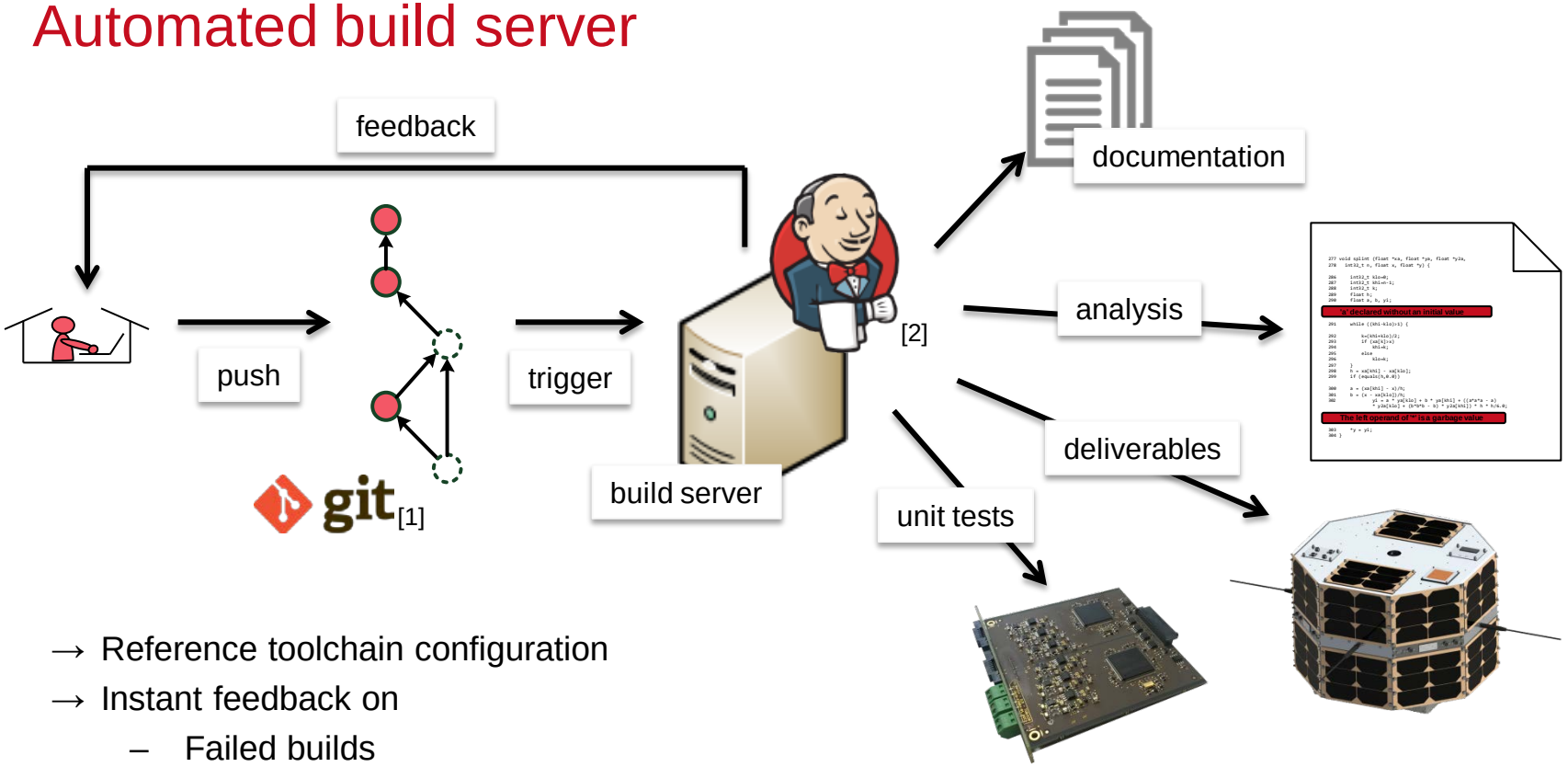
Mainline branches

- Aspect: main line of a single node
- Master: system-wide integration
- Contain only mature, reviewed code



→ Local flexibility, global quality assurance
→ Clean and short history
→ No merge conflicts
→ One central backup is sufficient

Automated build server



→ Reference toolchain configuration

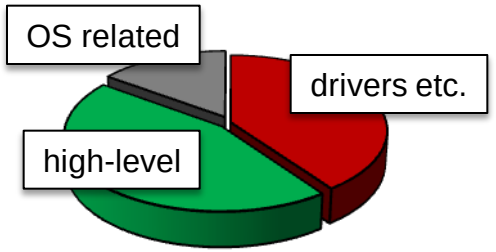
→ Instant feedback on

- Failed builds
- Code coverage [3]
- Static analysis results [4]
- Remote unit test results

Remote unit testing

Unit tests of board software in a PC environment

- Fast and convenient
- Do not cover hardware-dependent part of embedded software
- Different toolchain → different results

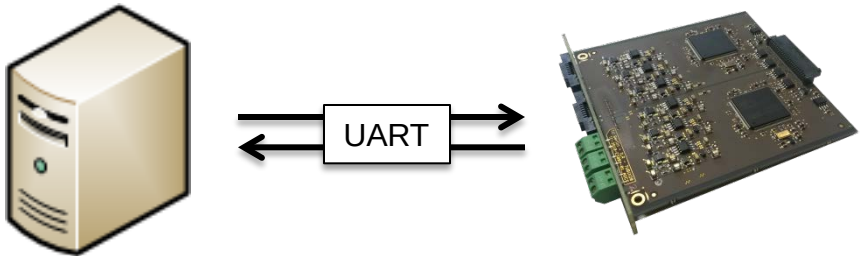


Conventional embedded testing frameworks

- Test harness and tests linked into single binary
- No automated upload and execution

ECatch test framework

- Based on Catch [5]
- Developed in TechnoSat project
- Test harness remains on host
- Facilitates test upload and result reporting



→ Automated unit testing on target hardware

Conclusions

Summary

- TechnoSat and TUBIN: TUBiX20 missions
- Bus concept: modularity and re-use for hard- and software
- Key process steps and tools of continuous integration

Experience

- Establishing a version control workflow pays off
- Automated tests and analysis support software verification to a great extent
- Systematic development approach saves time

Lessons learned

- Continuous Integration should be considered directly from the beginning
- Setup of workflow and toolchain demands time
- Third-party software components must comply with the process

Thank you!

Co-Authors

– Alexander Graf
– Merlin Barschke

TechnoSat funding

– German Aerospace Center (DLR)
– Grant No. 50 RM 1219 and 50 RM 0902

Supported by:

Federal Ministry of Economics and Technology

on the basis of a decision by the German Bundestag

References

[1] http://www.git-scm.com/

[2] https://jenkins-ci.org/

[3] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[4] http://clang-analyzer.llvm.org/

[5] https://github.com/philsquared/Catch

Practical experience: high-level modules

- Fusion of AOCS sensor data
- Determine inner state and environmental quantities

Challenges

- Numerous external and internal influence factors
- Dynamic and complex scenarios

Unit test coverage

- Orbit and environmental models
- State determination modules

Benefits

- Verified code base
- Flexible and reproducible tests for FDIR mechanisms
- Reference for hardware in the loop tests

Practical experience: hardware dependencies

- Lower layers of the UHF communication software

Challenge

- Test-coverage
- Ambiguous component documentation

Test-driven development

- Early tests refined the developer's understanding of the hardware
- Functional tests with two equal peripherals of the same microcontroller

Benefits

- ECatch remote test upload and execution → low turn-around time
- Hardware in the loop tests: improved confidence in the code base

Further steps

- Automated hardware-dependent tests connected to server