Tárgytematika / Course Description
Production Software Development

GKNM_MSTA092

Tárgyfelelős neve / Teacher's name:	dr. Bakosi József	**Félév / Semester:**	2024/25/1
Beszámolási forma / Assesment:	Folyamatos számonkérés		
Tárgy heti óraszáma / Teaching hours(week):	2/2/0	**Tárgy féléves óraszáma / Teaching hours(sem.):**	0/0/0

OKTATÁS CÉLJA / AIM OF THE COURSE

Prepare students for software development in large institutions (e.g., laboratories, industry) with hundreds or thousands of employees.

TANTÁRGY TARTALMA / DESCRIPTION

1 Key concepts, software engineering, types of software development

Concepts. The difference between computer science vs. software engineering. Which one is better? The difference between computer vs. computational science. Which one is better? The difference between research vs. production code. Which one is better? Why should you care?

2 Tools for productivity

Terminal. Terminal multiplexers. Shell utilities. File managers. Editors. Code search and navigation. Symbolic and numeric math tools. Build systems. Build automation tools. Documentation generators. Testing frameworks. Tools for testing. Code analysis tools. Version control and code history. Libraries as tools. Code review tools. Team communication tools.

3 Version control

Why use version control? Git. Github and alternatives. Basic usage. Proper commit messages. Resolving conflicts. Stash. Rewriting history.

4 Build systems

What are build systems? Why use build systems? Build correctness, performance, automation and portability. Parallel and distributed builds. Scalability. Off-the-shelf and custom build systems. CMake, GNU make, Ninja.

5 Documentation

Why document? Specifics for computational and production codes. Documenting theory, software requirements, specification, design, implementation, and interfaces. Documenting verification and validation cases and user examples. Source code control history as documentation. Documenting and archiving team collaboration. Documenting code correctness. Tracking code quality. Tools.

6 Testing, continuous integration, code quality

Why test? Types of tests. Unit tests. Regression tests. Acceptance tests. Verification and validation. Performance testing and regression. Static and dynamic code analysis. Test-driven development. Testing frameworks and libraries. Build systems integration. Fuzz testing. Continuous integration: automation of code review, code changes, builds, testing, deployment.

7 Testing, continuous integration, code quality, cont
Tools for continuous integration. Measuring test code coverage. Automating code coverage measurement. Build system integration. Instrumentation. Memory-error detection. Thread-error detection. Cache-, and branch-prediction profiling. Heap-profiling.
8 Programming styles
Procedural, object oriented, generic, functional programming styles. Languages. Using the right tool for the right job. Code correctness. Power consumption. Compile and runtime performance. Maintainability. Productivity. User and developer friendliness.
9 Software design
Why design production code? Process. Requirements. Specifications. Value. Artifacts. Design principles. Design concepts. Design considerations. Modeling languages. Design patterns. Priorities for writing code. Programmer productivity.
10 Third-party libraries
Pros and cons of using third-party libraries. Libraries for computational code. Maintenance. Upstream contributions.
11 Software development in teams, effective communication
Effective team communication. Archiving communication. Centralized vs. decentralized and synchronous vs. asynchronous comunication. The Cathedral and the Bazaar. Available tools. Software development methodologies: agile, waterfall, feature-driven, extreme programming.
12 Code review
The importance of code review. The de-facto standard open-source code development process (Github flow). Code review in distributed teams.
13 User-friendly input
The importance of user-friendliness. Balance between user-, vs. developer-friendly and foolproof vs. expert-user input. Automated user-input for verification & validation and uncertainty quantification.
14 Learning from history
Standing on the shoulders of giants. Not invented here vs. Proudly found elsewhere. Engineering project: in spec, on time, within budget. Case studies: (1) Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulatinos: Lessons learned from ASCI at Los Alamos National Laboratory, (2) The ZeroMQ Community: Large architectures, The ZeroMQ process, Designing for innovation, Burnout, Patterns for success.

SZÁMONKÉRÉSI ÉS ÉRTÉKELÉSI RENDSZERE / ASSESMENT'S METHOD

Course requirements:
* Attendance/signature: 50% mandatory
* No exam
* Grades based on degree of participation in team project:

(1) Actively participate in team communication
1: inactive, 2: does what is necessary, 3: proactive on his own task, 4: even helps others
(2) Successful use of productivity tools
1: basic terminal/tools user, 2: good terminal/tools user but needs gui for multiple tasks, 3: can do most tasks in terminal, rarely needs gui, 4: terminal guru: can do everything in terminal if necessary
(3) Successful use of version control
1: knows basic git cmds, 2: knows intermediate git cmds, 3: knows advanced git cmds, confident with merge conflicts, 4: knows advanced git cmds, e.g., complex history rewriter
(4) Contribute to project build system
1: basic build system contributor: change existing code, 2: basic new contributions to build system, 3: nontrivial new contributions to build system, 4: comfortable writing new build system code
(5) Actively participate in code review
1: successfully follows team code review procedure, 2: follows and correctly implements instructions

from peers, 3: suggests at least one review comment to others's chage-set, 4: actively contributes to other's change-sets

(6) Hook up a third-party library to our code product

1: requires substatial help hooking up lib, 2: needs little help hooking up lib, 3: hooks up lib without help, 4: even helps others hooking up other lib

(7) Participate in writing/generating code documentation

1: writes only self-documenting code, 2: correctly documents interfaces, 3: contributes to descriptive doc pages, 4: writes detailed doc page(s) complete with theory, equations, figures, basic usage, testing functionality

(8) Understand and contribute to test system setup and contribute tests

1: adds a test to existing test harness, 2: adds multiple types of tests to existing test harness, 3: contributes to testing infrastructure, 4: substantial contributions to creating testing infrastructure

* Grades:

0-16: 1
17-20: 2
21-24: 3
25-28: 4
28-32: 5

KÖTELEZŐ IRODALOM / OBLIGATORY MATERIAL

https://www.indeed.com/career-advice/finding-a-job/computer-science-vs-software-engineering
https://en.wikipedia.org/wiki/Terminal_multiplexer
https://linuxcommandlibrary.com/basic/oneliners
https://github.com/jarun/nnn
https://github.com/wting/autojump
https://github.com/junegunn/fzf
https://git.hackliberty.org/Awesome-Mirrors/awesome-cli-apps
https://git.hackliberty.org/Awesome-Mirrors/Awesome-Linux-Software
https://git.hackliberty.org/Awesome-Mirrors/awesome-math
https://git.hackliberty.org/hackliberty.org/Hack-Liberty-Resources#system-administration
https://en.wikipedia.org/wiki/Comparison_of_documentation_generators
https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
https://stackoverflow.com/a/1408464
https://github.com/will133/vim-dirdiff
https://git-scm.com/docs/gittutorial
https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History
https://egghead.io/courses/how-to-contribute-to-an-open-source-project-on-github
https://help.github.com/en/articles/fork-a-repo
https://git-scm.com/book/en/v2
https://cliutils.gitlab.io/modern-cmake
https://www.doxygen.nl
https://www.parasoft.com/blog/measuring-code-coverage
https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
https://cppcheck.sourceforge.io
https://codecov.io
https://sonarqube.org
https://nvie.com/posts/a-successful-git-branching-model
https://resources.idgenterprise.com/original/AST-0053933_seven_qualities_of_wildly_desirable_software.pdf
https://doi.org/10.1177/1094342004048534
https://abseil.io/resources/swe-book/html/ch18.html
https://en.wikipedia.org/wiki/Software_design
https://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-

disadvantages/
https://docs.github.com/en/get-started/using-github/github-flow
https://zguide.zeromq.org/docs/chapter6
https://www.3pillarglobal.com/insights/blog/the-importance-of-code-reviews
https://distantjob.com/blog/6-reasons-why-code-reviews-are-especially-important-for-remote-teams/
https://scicomp.stackexchange.com/questions/14569/test-set-for-linear-solvers
https://www.perforce.com/blog/sca/what-static-analysis

AJÁNLOTT IRODALOM / RECOMMENDED MATERIAL