UDC 004.75:004.9 **DOI: 10.20998/2411-0558.2022.02.01**

O. MNUSHKA, Sen. Lect., NTU "KhPI",
S. LEONOV, Dr. Sc. (Engineering), Prof., NTU "KhPI",
V. SAVCHENKO, Cand. Sc. (Engineering), Assoc. Prof., NTU "KhPI"

CONTINUOUS INTEGRATION FOR A DEVELOPMENT PROCESS OF THE INFORMATION TECHNOLOGY OF REMOTE MONITORING AND CONTROL

The current state of using CI/CD in commercial and open-source projects is analyzed. CI/CD pipelines are shown to be essential to the modern software development process, where they are used to configure workflows. Centralized and distributed approaches to building a software source code version control system are analyzed. For the web-oriented SCADA project, the choice and use of the version control system are justified. We provide concrete examples of using CI/CD that will allow ordinary developers, not DevOps specialists, to take the first steps to configure workflows for their projects and provide basic steps for configuring task execution using the example of the build and test phase of the source code of a real project that uses several programming languages. Figs.4. Refs. 20 titles.

Keywords: CI; CD; VCS; git; workflow; actions; building; testing; Agile.

Introduction. Web-oriented supervisory and control systems are used in various fields to acquire data and make control based on data analysis and implemented algorithms. It is a SCADA system but in the Web or a cloud. The development of such systems faces many problems caused by the complexity of processes, and a wide range of equipment – from physical quantity sensors to routers, modems, server equipment, etc. [1]

The development process consists of several big subprojects:

- embedded systems for working with field equipment;
- a collecting and intermediate data storage subsystem;
- a centralized data storage and processing system;
- a visualization system for the control process;
- security and safety systems;
- a system for providing paid services, usually based on a subscription, unlike traditional turnkey systems delivered to the customer.

In addition, a wide range of technologies used in the development process: development of embedded systems in assembler and C/C++/Python/Java; development of applied data exchange protocols; development of a database; and Web development of traditional and mobile applications.

For the systems under consideration, there is a task of ensuring the interaction of developers, monitoring the implementation of customer requirements and monitoring the compliance of the process with accepted practices; Actual changes to the program code and system architecture; documentation and elimination of bugs; issuing software patches upon detection of security problems in the technologies and tools used.

It is necessary to determine the process of developing hardware and software based on Agile, Scrum, Waterfall, or something else.

The problem statement and analysis of literature.

A. Continuous Integration, Delivery, and Development.

Continuous Integration (CI) is used to automate the building and testing of programming code and deliver artifacts based on a centralized architecture and fixed schedule.

As described in [2], CI is one of the programming practices that leads to improved release rates and predictability. It is a good choice for improving code quality, communication, and sharing experience between developers, due to the complexity and wide use of the terminology of the development process, 22 clusters of descriptive statements related to CI, which are used as the base for the descriptive model of continuous integration implementations. As shown, there is a problem of correlations between differences in practice and differences in experience, which may lead to some misunderstanding of CI effects on the process.

In [3] a collection of patterns of planning, managing, and executing CI was proposed. The patterns can be helpful for cross-platform agile-based development. As shown, there are three main categories of a task – artifacts management, source code mandatory, and build execution, which are represented as ten patterns used for planning. Each pattern means a solution for some specified tasks like “Create a platform-specific installation program for the cross-platform product.” Using the patterns leads to reduced build time, simplifies dependency management, and decreases the probability of unsuccessful builds.

In [4], the best practices of CI for rapid application development are discussed. CI is a set of engineering tools used for daily tasks like bug tracking, version control, code review, etc. CI does not cover all needs of development teams and not only infrastructure for building and running. However, CI is a practice that may be successful. Each engineer has the same vision of the process as other teammates. In addition, it is crucial to allow an engineer to use local CI runs before any commits into the main build plan that prevents inconsistency and broken builds.

Including performance benchmarks in CI execution plans make it possible to use DevOps engineers to find some specific issues in the development code. Regression benchmarks are tools for application monitoring on a regular base. As shown in [5], it is a good practice to include some performance benchmarks as early as possible in the development process. As a result, regular monitoring can help developers to find hidden issues not related to QA and improve software quality.

In [6], it is shown that including benchmarks in CI and CD (Continuous Delivery) leads to the enchantment of code quality. As a benchmarking environment, virtual cloud servers are used. Results represented as CPU and memory loads, threads, IO, and network traffic can be used for real application performance tuning.

As shown in [7], for Agile-based projects, CI and other programming practices like extremal programming help new and growth development teams and organizations improve software development and delivery processes and reduce expenses. Therefore, the Agile-based development process is a good choice for predictable results and controlling expenses.

CI has not only benefits but also some problems, such as security and additional expenses for CI. As shown in [8] additional yearly costs are about 9.7 percent recalculated to all FLOSS (Free, Libre, and Open Source Software) projects. Therefore, for economic reasons, some FLOSS projects decline regular CI. On the other hand, for commercial projects, risks related to the low quality of the code are higher than costs for CI, so CI is the excellent choice for such projects.

Security vulnerabilities are the next issue for CI, which is more vulnerable to attacks and misconfiguration than conventional software development tools [8]. Of course, here we are talking about publicly available servers like Jenkins (https://www.jenkins.io). There are four main steps: checkout code from the version control system; build preparations; build runs; notifications - all of them are the potential target of an attack. The concept of a secure build server (SBS) was proposed. SBS is back to an original and uncompromised state after each build job in this concept, so the public build server is usable for many small commands. Therefore, using the public build server may affect all jobs or teams and is not a good choice for big projects.

As shown in [9], developers face trade-offs between various aspects during a software development process. Some of them are guaranteed speed and availability of resources, information security and access to resources, ease of use, and flexible configuration. Most of these challenges are solved with CI as it automates the compilation, building, and testing of software. However, some problems need to be considered more precisely. For example, desired and actual build times for developers or using flaky test identification tools.

B. CI in projects.

In [10], implementing CI (Travis-CI) in the GitHub-based project was analyzed. On large massive of historical data was shown that CI helps to improve teams' productivity and leads to the integration of more outside contributions without an observable diminishment in code quality.

In [11] a new architecture framework named Cinders was presented. It was shown that a single architectural framework could be designed to encompass the previous CI and CD modeling techniques, representing their specific concerns as viewpoints rendered from the same underlying data model.

GNOME [12] is a big open-source project that depend on a big number of community developers. It uses a git-like approach to the building and deployment of applications. For now, the package-based distribution model of GNOME components leads to many inconsistencies for non-technical developers. OSTree CI system was presented. It consists of three independent parts related to working with the version control system, build, and deployment system. OSTree is based on Yocto provided minimal system build (core) and applications build. It leads to the improvement development process for git-like applications in contrast to the package model.

In [13], the framework that identifies software quality characteristics of the (unnamed) financial software development process that uses CI was presented. Some quality metrics are used, such as time to develop, introduced bugs, time to deliver, test quality, documentation, change management, and and cost model. The main benefit of CI is a single requirement model used for development and testing, which leads to a transparent process for developers and management teams. All requirements are documented and approved by a product owner.

In [14], a pattern-based framework for CI and QA of the EGEE Grid Middleware software development project has been presented. The main benefits are better integration of components on multiple platforms, fast bug and problem tracking, and other improvements in the process. Furthermore, the build system was balanced against the need for flexibility and rapid prototyping.

CI, as Agile technology, is widely used in various application development processes, from small open-source to big commercial or government-distributed applications. The main benefit of CI is a clearer and more predictable development process. But, there is no universal solution for all, each time, we need to consider many factors before integrating CI into the development process. There are open-source and commercial versions of CI tools. Some of them are deeply integrated with other tools like Atlassian Bamboo, while others are independent, like Jenkins or TravisCI, and used in many integration variants.

The publication aims to analyze the main principles and state of the art in the field of Continuous Integration and continuous delivery, analyzing approaches to building and executing workflows for projects that use distributed version control systems.

Continuous Integration for web-based SCADA. Consider the prototype of a continuous integration system and the main tasks solved by the functional blocks of the system (Fig. 1).

Let us define three main flows inside the development process: development, building and testing, and automated deployment.

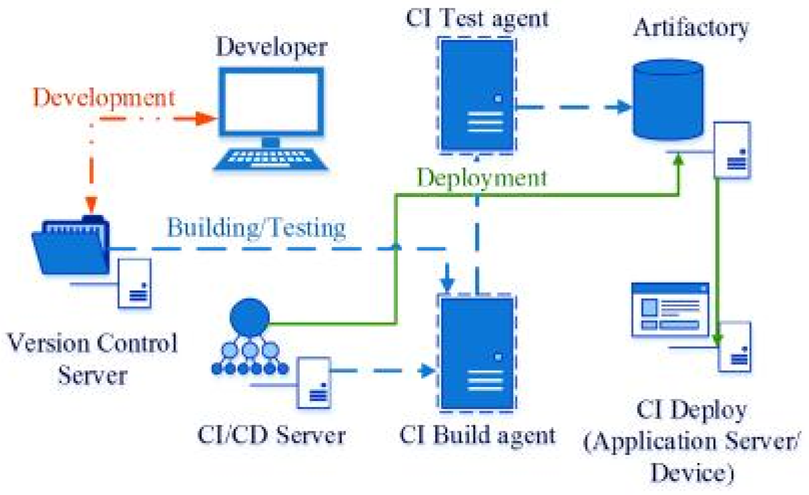


Fig. 1. CI prototype (authors)

Deployment flow means direct use of VCS server by developers and it is permanent storage for source code for all subproject teams.

Developers use local resources for building and testing it would be better to use local CI runs to debug requirements and issues. At least, it is possible to create local testing infrastructure like a remote one, run build and test plans for subproject before committing in the VCS tree.

VCS server includes review capabilities for approving changes before committing and pull request for discussing and reviewing the potential changes with collaborators.

Historically there were the two different approaches to build VCS subsystem – centralized and decentralized VCS. The centralized version control systems like Concurrent Versions System (CVS, https://savannah.nongnu.org/projects/cvs) and its successor Apache Subversion (SVN, CollabNet, and Apache Software Foundation, https://subversion.apache.org/) are based on client-server architecture and centralized remote or local data storage. Such an approach allows having a centralized database of all versions of project files. Developers can obtain the needed file version and work with it. After they are finished, they send a file back to the VCS server. In addition, any developer can check the tasks of other developers and have imagination about the project’s current state. The administrators have complete control of project files (database), so it is easy to support such a system. However, in this approach to building VCS, there is a bottleneck, and it is a VCS server. If something happens with a VCS server, all project files will be unavailable for some period, affecting some project tasks, i. e. Berkeley DB issues in the first versions of Subversion. Also, any storage issues can lead to the loss of project history, which is the main disadvantage of such systems. So, centralized VCS were de-facto standards for a long time, and some projects still use them, but their popularity decreased after distributed (decentralized) VCS were developed and introduced.

There are several distributed VCS, such as Git, Bazaar, Mercurial etc. In distributed VCS the complete project codebase is mirrored on every developer's computer, and every developer has an entire project version history. Such a VCS allows for easy branching and merging processes. Furthermore, developers may work offline and push their changes into the central repository when they return online. It is crucial, for example, in the COVID-19 pandemic restrictions to work out of the office and for other unpredictable situations like blackouts, hurricanes, snowstorms, etc. For all mentioned cases, distributed VCS allows independent working for each development team member and synchronized changes when possible.

The next advantage of distributed VCS is automatic "multiply backups," i. e. multiple complete copies of the project repository but maybe for different stages. So it is required to synchronize repositories using some remote server. Developers can work with different remote servers and teams in one project; also, distributed VCS can be used to set up different workflows in a project.

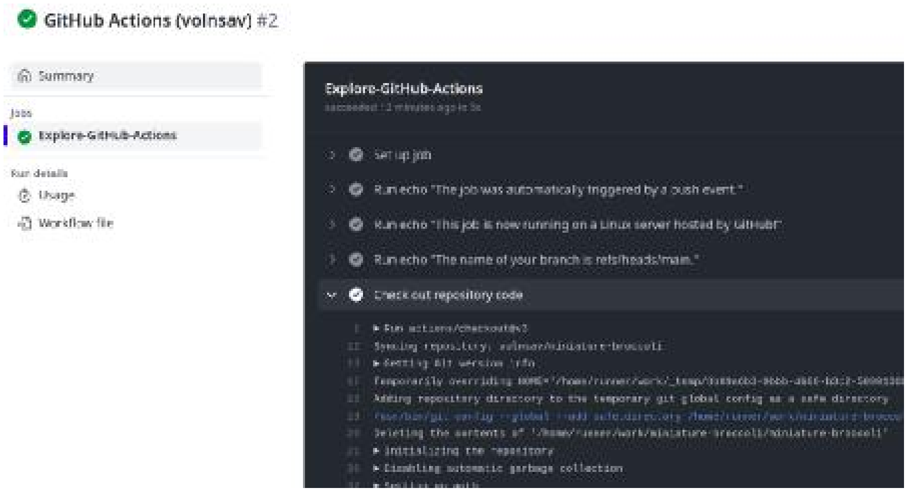
For distributed VCS (Git), we can face some disadvantages such as security issues, productivity issues for projects with many repositories for workflows using CI/CD, and many non-text files (different containers, virtual machines, multimedia), and so on.

For the project under development [16, 17], we decided to use Git VCS and keep the project in a private repository on github.com.

GitHub allows to set up of workflows based on its Actions [18]. To setup the workflow, we need to define YAML [19] configure file like the following (based on https://docs.github.com/en/actions/quickstart):

```
name: GitHub Actions Demo
run-name:  GitHub Actions (${{ github.actor }})
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - run: echo “The job was automatically triggered by a ${{ github.event_name }} event.”
      - run: echo “This job is now running on a ${{ runner.os }} server hosted by GitHub!”
      - run: echo “The name of your branch is ${{ github.ref }}."
      - name: Check out repository code
        uses: actions/checkout@v3
      - run: echo “The ${{ github.repository }} repository has been cloned to the runner.”
      - run: echo “The workflow is now ready to test your code on the runner.”
      - name: List files in the repository
        run: |
          ls ${{ github.workspace }}
      - run: echo “This job’s status is ${{ job.status }}."
```

In this YAML file, we defined when actions should run (on every push event) and job (Explore-GitHub-Actions) that runs on the ubuntu-latest server and consists of seven steps. Also, some steps use predefined actions such as "actions/checkout@v3" [18]. As a result, we have a simple workflow (Fig. 2) that is executed every time we push in our repo. To disable a workflow, we need to use GitHub Actions web interface. Also, we can have many Actions-based workflows in one project.



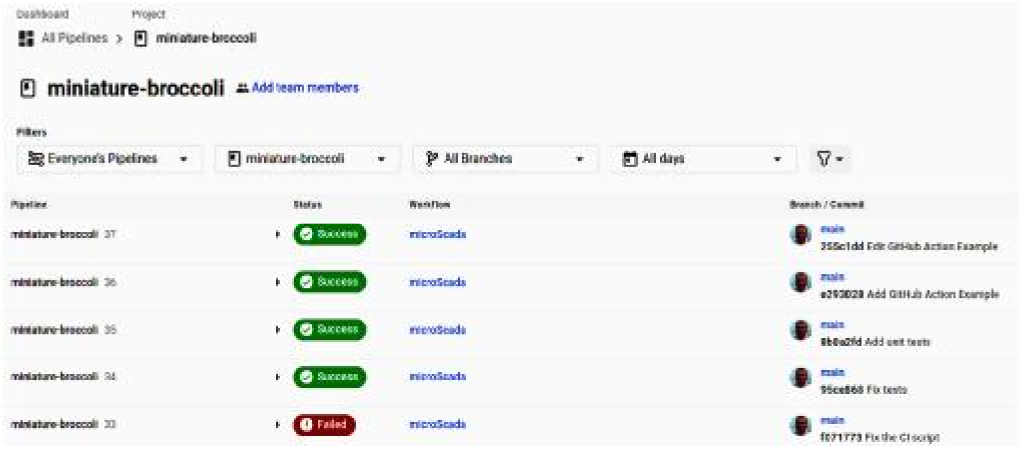
Explore-GitHub-Actions
succeeded 2 minutes ago in 5s

```
Set up job
Run echo "The job was automatically triggered by a push event."
Run echo "This job is now running on a Linux server hosted by GitHub!"
Run echo "The name of your branch is refs/heads/main."
Check out repository code
  ▶ Run actions/checkout@v3
  Syncing repository: volnsav/miniature-broccoli
  ▶ Getting Git version info
  Temporarily overriding HOME='/home/runner/work/_temp/[illegible]'
  Adding repository directory to the temporary git global config as a safe directory
  /usr/bin/git config --global --add safe.directory /home/runner/work/miniature-brocco
  Deleting the contents of '/home/runner/work/miniature-broccoli/miniature-broccoli'
  ▶ Initializing the repository
  ▶ Disabling automatic garbage collection
```

Fig. 2. GitHub Actions example (fragment)

Another approach is using CircleCI [20], which allows automated builds across multiple environments. It can be used with GitHub for free up to 6,000 build minutes per month. We must complete only two steps to use CircleCI with GitHub projects: authorizing the CircleCI application and defining a YAML configuration file. Let's define the YAML configuration file for building and testing our sub-project, which we used for testing different features and components of the information technology of remote monitoring and control [16, 17]:

```
version: 2.1
jobs:
  build:
    docker:
      - image: ubuntu:latest
    steps:
      - checkout
      - run: |
          apt -y update
          apt -y upgrade
      - run:
          apt install -y python3 python3-pip
      - run:
          apt install -y mosquitto
      - run:
          apt install -y nodejs npm
      - run:
          python3 -u -m pip install -r requirements.txt
      - run:
          npm i webpack webpack-cli html-webpack-plugin
      - run: |
          cd ~/miniature-broccoli/react-app/
          npx -y browserslist@latest --update-db
          rm -rf node_modules
          rm -f package-lock.json
          npm cache clean --force
          npm install
          npm run build
      - run: |
          python3 -u -m pip install pytest
          cd ~/miniature-broccoli
          pytest tests/Test.py
workflows:
  microScada:
    jobs:
      - build
```

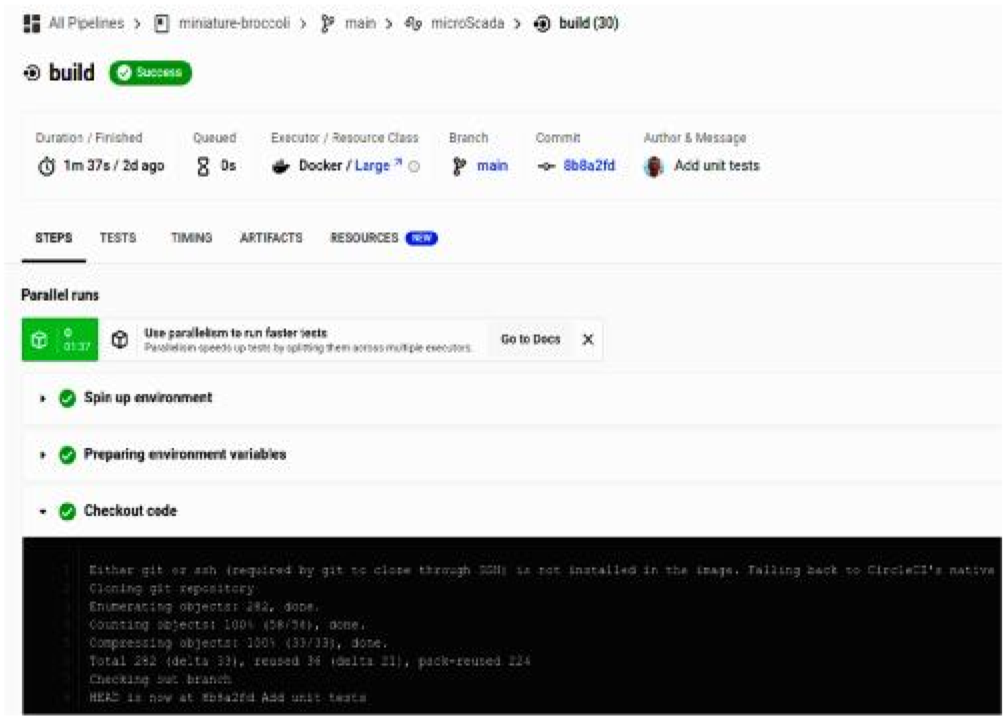

Dashboard Project
All Pipelines > miniature-broccoli

miniature-broccoli Add team members

Filters
Everyone's Pipelines | miniature-broccoli | All Branches | All days

Pipeline	Status	Workflow	Branch / Commit
miniature-broccoli 37	Success	microScada	main 255c1dd Edit GitHub Action Example
miniature-broccoli 36	Success	microScada	main e293028 Add GitHub Action Example
miniature-broccoli 35	Success	microScada	main 8b8a2fd Add unit tests
miniature-broccoli 34	Success	microScada	main 95ce868 Fix tests
miniature-broccoli 33	Failed	microScada	main f071773 Fix the CI script

Fig. 3. CircleCI dashboard (fragment)


All Pipelines > miniature-broccoli > main > microScada > build (30)

build Success

Duration / Finished	Queued	Executor / Resource Class	Branch	Commit	Author & Message
1m 37s / 2d ago	0s	Docker / Large	main	8b8a2fd	Add unit tests

STEPS
TESTS
TIMING
ARTIFACTS
RESOURCES NEW
Parallel runs
0
01:37
Use parallelism to run faster tests
Parallelism speeds up tests by splitting them across multiple executors.
Go to Docs

Spin up environment
Preparing environment variables
Checkout code

```
Either git or ssh (required by git to clone through SSH) is not installed in the image. Falling back to CircleCI's native
Cloning git repository
Enumerating objects: 282, done.
Counting objects: 100% (58/58), done.
Compressing objects: 100% (33/33), done.
Total 282 (delta 33), reused 36 (delta 21), pack-reused 224
Checking out branch
HEAD is now at 8b8a2fd Add unit tests
```

Fig. 4. CircleCI workflow execution results (fragment)

By default, CircleCI expects the YAML configuration file config.yml in the .circle folder in the project's root folder. After adding the configuration file and pushing it into the GitHub repository

Results. As the first result of the analysis and testing, we have chosen distributed model for the version control system and github.com as the host for our project under active development.

We have investigated two approaches to building and executing a workflow with GitHub Actions and CircleCI. GitHub Action is fully integrated into GitHub and can be used with public and private repositories. CircleCI is a standalone CI that can be used with GitHub projects.

Both approaches allow flexible workflow configuration with YAML configuration files, and they are independent of all other project codebases in terms of project code and can be used in parallel. In addition, they use similar but slightly different syntaxes for describing a workflow.

The main workflow is defined in the *steps* section and includes the sequence of actions (*checkout*) and commands (*run*). Any valid combination of Linux (or Unix) shell commands is allowed here; also, we can use the *make, cmake,* and other automation tools. However, in our example, we omit automation tools usage to show how we can define workflow manually.

Using predefined Actions for GitHub and Orbs for CircleCI speeds up YAML configuration with reusable and tested configuration.

The workflow execution results (Fig. 2 – 4) are represented on the web interface for both providers and show detailed data about each defined in the YAML configuration file steps.

Conclusion. CI/CD pipelines are essential to the modern software development process. Today most projects use CI/CD to set up workflows. Commercial projects usually use a DevOps team responsible for CI and CD. DevOps team tries using best practices to set up different workflows and speed up a development process.

CI/CD is useful for teams that use Agile, Waterfall, and V-model for project management. Agile projects, as the most flexible type of project management, strictly depend on CI/CD because they use short-time periods (iterations) when some new features should be developed and tested.

The results can be used to set up basic workflows for different projects based on Git VCS. We provide specific examples that will allow ordinary developers, not DevOps specialists, to take the first steps to configure workflows for their projects and give some basic steps to configure job execution on the example of the real sub-project MVP. Also, we describe only the build and testing stage because it is a prevalent task for all projects.

The prospects for further research are the optimization of work processes according to various criteria and the possibility of using CI/CD opportunities to improve the educational process of computer engineering and computer science specialists.

References:

1. Mnushka, O. and Savchenko, V. (2019). Security model of an information system based on IoT technologies, *Bulletin of the National Technical University "KhPI", A series of "Information and Modeling",* 0(28 (1353)). doi:10.20998/2411-0558.2019.28.09.
2. Ståhl, D. and Bosch, J. (2014). Modeling continuous integration practice differences in industry software development, *Journal of Systems and Software*, 87, pp.48-59. doi:10.1016/j.jss.2013.08.032.3.
3. Hsieh, C.-Y. and Chen, C.-T. (2015). Patterns for Continuous Integration Builds in Cross-Platform Agile Software Development. *Journal of Information Science and Engineering*, vol. 31, pp. 897-924.
4. Abdul, F.A. and Fhang, M.C.S. (2012). Implementing Continuous Integration towards rapid application development. *2012 International Conference on Innovation Management and Technology Research*. doi:10.1109/icimtr.2012.6236372.
5. Waller, J., Ehmke, N.C. and Hasselbring, W. (2015). Including Performance Benchmarks into Continuous Integration to Enable DevOps. *ACM SIGSOFT Software Engineering Notes*, 40(2), pp.1–4. doi:10.1145/2735399.2735416.
6. Arachchi, S.A.I.B.S. and Perera, I. (2018). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. *2018 Moratuwa Engineering Research Conference (MERCon)*, Moratuwa, pp. 156-161. doi:10.1109/MERCon.2018.8421965.
7. Boehm, B. and Turner, R. (2005). Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software*, 22 (5), pp. 30-39. doi:10.1109/ms.2005.129.
8. Gruhn, V., Hannebauer, C. and John, C. (2013). Security of public continuous integration services. *Proceedings of the 9th International Symposium on Open Collaboration - WikiSym '13*. doi:10.1145/2491055.2491070.
9. Hilton, M., Nelson, N., Tunnell, T., Marinov, D. and Dig, D. (2017). Trade-offs in continuous integration: assurance, security, and flexibility. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pp. 197-207. doi:10.1145/3106237.3106270.
10. Walters, C., Poo-Caamano, G. and German, D.M. (2013). The future of continuous integration in GNOME. *2013 1st International Workshop on Release Engineering (RELENG)*, San Francisco, CA, pp. 33-36. doi:10.1109/releng.2013.6607695.
11. Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015). Quality and productivity outcomes relating to continuous integration in GitHub. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. doi:10.1145/2786805.2786850.
12. Ståhl, D. and Bosch, J. (2018). Cinders. *Proceedings of the 2018 International Conference on Software and System Process*, vol. 83, pp. 76-93. doi:10.1145/3202710.3203165.
13. Hamdan, S. and Alramouni, S. (2015). A Quality Framework for Software Continuous Integration. *Procedia Manufacturing*, 3, pp.2019-2025. doi:10.1016/j.promfg.2015.07.249.
14. Meglio, A., Flammer, J., Harakaly, R., Zurek, M. and Ronchieri, S. (2005). A pattern-based continuous integration framework for distributed EGEE grid middleware development. *14th International Conference on Computing in High-Energy and Nuclear Physics*, pp. 579-582 [online] Available at: https://cds.cern.ch/record/865656/files/p579.pdf [Accessed 7 Dec. 2022].
15. Continuous Integration Impediments in Large-Scale Industry Projects. *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 169-178. doi:10.1109/icsa.2017.11.
16. Mnushka, O., Savchenko, V., Leonov, S. and Shaposhnikova, O. (2021). Information Technology of Remote Monitoring and Control. *2021 International Conference on Electrical,*

Computer, Communications and Mechatronics Engineering (ICECCME). doi:10.1109/iceccme52200.2021.9590889.
17. Mnushka, O., Leonov, S., Shaposhnikova, O. and Savchenko, V. (2021). Model of information technology of remote monitoring with a possibility of control of technological processes and objects. *Bulletin of the National Technical University 'KhPI' A series of 'Information and Modeling',* (1 (5)), pp.99–114. doi:10.20998/2411-0558.2021.01.08.
18. GitHub. (n.d.). GitHub Actions. [online] Available at: https://github.com/actions [Accessed 10 Nov. 2022].
19. Yaml.org. (2011). The Official YAML Web Site. [online] Available at: https://yaml.org/ [Accessed 15 Nov. 2022].
20. CircleCI. (n.d.). Continuous Integration and Delivery. [online] Available at: https://circleci.com. [Accessed 5 Nov. 2022].

Статтю представив д.т.н., проф. Національного технічного університету "Харківський політехнічний інститут" О.А. Серков.

Поступила (received) 06.11.2022

Mnushka Oksana, Senior Lecturer, M.S. (Computer. Science)
National Technical University "Kharkiv Polytechnic Institute"
2, Kyrpychova str., Kharkiv, Ukraine, 61002
Tel.:(050)2428846, e-mail: mnushka.ov@gmail.com
ORCID ID: 0000-0001-7756-9260

Leonov Serhii, Dr. Sc. (Engineering), Professor
National Technical University "Kharkiv Polytechnic Institute"
2, Kyrpychova str., Kharkiv, Ukraine, 61002
Tel.: (099) 9119113, e-mail: serleomail@gmail.com
ORCID ID: 0000-0001-8139-0458

Savchenko Volodymyr, Cand. Sc. (Engineering),
National Technical University "Kharkiv Polytechnic Institute"
2, Kyrpychova str., Kharkiv, Ukraine, 61002
Tel.:(067)5767884, e-mail: savchenko@live.com
ORCID ID: 0000-0001-6548-0891

УДК 004.75: 004.9

Безперервна інтеграція для процесу розвитку інформаційних технологій віддаленого моніторингу та управління / Мнушка О., Леонов С., Савченко В. // Вісник НТУ "ХПІ". Серія: Інформатика та моделювання. – Харків: НТУ "ХПІ". – 2022. – № 1 – 2 (7 – 8). – С. 5 – 17.

Проаналізовано поточний стан використання CI/CD у комерційних та open-source проектах. Показано, що конвеєри CI/CD є важливими для сучасного процесу розробки програмного забезпечення, де вони використовуються для налаштування робочих процесів. Проаналізовано централізований та розподілений підходи до побудови системи контролю версій вихідного коду програмного забезпечення. Для веб-орієнтованого проекту SCADA обґрунтовано вибір і використання системи контролю версій. Ми надаємо конкретні приклади використання CI/CD, які дозволять звичайним розробникам, а не фахівцям DevOps, зробити перші кроки для налаштування робочих процесів для своїх проектів і надамо базові кроки для налаштування виконання завдань на прикладі фази збірки та тестування джерела. код реального проекту, який використовує кілька мов програмування. Іл.: 4. Бібліогр.: 20 назв.

Ключові слова: CI; CD; VCS; git; workflow; actions; building; testing; Agile.

UDC 004.75: 004.9

Continuous Integration for a development process of the information technology of remote monitoring and control / Mnushka O., Leonov S., Savchenko V. // Herald of the National Technical University "KhPI". Series of "Informatics and Modeling". – Kharkov: NTU "KhPI". – 2022. – № 1 – 2 (7 – 8). – P. 5 – 17.

The current state of using CI/CD in commercial and open-source projects is analyzed. CI/CD pipelines are shown to be essential to the modern software development process, where they are used to configure workflows. Centralized and distributed approaches to building a software source code version control system are analyzed. For the web-oriented SCADA project, the choice and use of the version control system are justified. We provide concrete examples of using CI/CD that will allow ordinary developers, not DevOps specialists, to take the first steps to configure workflows for their projects and provide basic steps for configuring task execution using the example of the build and test phase of the source code of a real project that uses several programming languages. Figs.: 4. Refs. 20 titles.

Keywords: CI; CD; VCS; git; workflow; actions; building; testing; Agile.