The European Journal of Technical and Natural Sciences 2026, No 1
ISSN 2414-2352

PREMIER Publishing
ppublishing.org

Section 6. Information technology

DOI:10.29013/EJTNS-26-1-101-107

INTEGRATION OF CONTINUOUS TESTING INTO THE DEVELOPMENT LIFECYCLE OF CORPORATE WEB SYSTEMS USING THE EXAMPLE OF DRUPAL

Kuletskaia Elena [1]

[1] Independent Research, USA, Westover

Cite: *Kuletskaia, E. (2026). Integration of continuous testing into the development lifecycle of corporate web systems using the example of Drupal. European Journal of Technical and Natural Sciences 2025, No 1*. https://doi.org/10.29013/EJTNS-26-1-101-107

Abstract

The article describes how to integrate the continuous testing process into the development cycle of corporate web systems using the example of Drupal. It has been proven that moving quality control to earlier stages and regularly performing automated checks with each significant change in code, dependencies, and settings significantly reduces the risk of regressions and increases the predictability of releases. This article focuses on the features of Drupal as an enterprise platform: layered code, dependency management using Composer, and a configuration approach. Changes to settings can cause regressions in the same way as changes to the program code. In this paper, the characteristics of software quality are associated with various types of test checks. A comprehensive model for integrating these checks into the development and delivery process is proposed. The test results are recorded and serve as the basis for making decisions about which changes can be combined and implemented. In addition, the article outlines promising areas of development: strengthening security checks, using observability data, testing infrastructure, and using artificial intelligence tools to support quality control processes.
Keywords: *continuous testing, development lifecycle, enterprise web systems, Drupal, CI/CD, regression testing, Composer, configuration management, test automation, quality assurance.*

Relevance of the study

The relevance of the research is determined by current trends in the development of corporate web systems. Increasingly, these systems are being created and developed in an environment of constant change delivery and continuous integration. In such an environment, there is an increasing need not only to accelerate the release of new versions, but also to ensure consistent quality, security, and reliability of software products at all stages of the software development lifecycle (SDLC).

The traditional testing model, in which quality control is carried out mainly at the final stages of development, has ceased to meet the requirements of corporate projects.

It does not allow timely detection of defects, which reduces the predictability of releases and increases the total cost of errors. In response to these challenges, the concept of continuous testing has emerged, which involves automated quality checks at each stage of the system lifecycle (SDLC). This allows you to detect functional, technical, and business risks when making any changes to the system at an early stage.

The problem is becoming even more urgent due to the increasing complexity of software ecosystems and the increasing dependence of corporate web systems on external components, libraries, and services. In modern conditions, the security of the software supply chain is of particular importance. The integration of quality checks, dependency vulnerabilities, configuration correctness, and standards compliance directly into automated development and deployment pipelines (CI/CD pipelines) becomes a prerequisite for ensuring trust in the software product and its successful operation in the corporate environment.

Choosing the Drupal content management system as an example is also reasonable and relevant. Drupal is widely used to create corporate web platforms, government portals, large information systems, and multi-level digital ecosystems. These systems are characterized by high load, complex modular architecture, and regular updates to the core and extensions. In such circumstances, even minor problems that occur when updating components or changing configurations can lead to serious system failures and financial losses.

Thus, the study on the integration of continuous testing into the development lifecycle of corporate web systems using the example of Drupal is both scientific and practical. It meets the modern requirements of digital transformation, aims to improve the quality of software products and contributes to the creation of reproducible and manageable quality assurance models in modern corporate IT projects.

The purpose of the study

The purpose of this study is to substantiate and describe a model for integrating continuous testing into the development process of corporate web systems using the example of Drupal. This model should provide reproducible quality control, which is especially important when code, dependencies, and configurations change frequently.

Materials and research methods

The research is based on open publications and documentation on continuous integration and delivery practices, as well as various approaches to continuous testing. In addition, we used open sources that describe the architectural and operational features of Drupal, as well as the principles of automated testing within this ecosystem.

The work used methods of analytical review and generalization, comparative analysis of various approaches to quality control, modeling of the continuous process of integrating checks into the software development cycle (SDLC) and systematization of types of test checks according to software product quality criteria.

The results of the study

Continuous testing in the process of developing corporate web systems is an approach that allows you to seamlessly integrate quality control into the process of making changes. The essence of this approach is that checks are carried out regularly and automatically with each significant update of the code, dependencies or configurations. This approach allows you to quickly and objectively receive information about the changes made, identify potential regression risks, and determine whether it is safe to advance the build along the development and deployment chain. Unlike the traditional approach, in which testing is carried out only before release, the continuous model significantly reduces the likelihood of "accumulated" errors and reduces the cost of fixes due to earlier detection of defects (What is Continuous Testing?).

To better understand the role of continuous testing in the corporate development process, it is important to compare the verifiable characteristics of a software product with the types of test checks that are used in practice. In this context, the quality of software is not considered abstractly, but through its specific properties, which can be checked and controlled at various stages of the life cycle. This relationship clearly demonstrates which

aspects of quality can and should be validated through various types of testing when implementing a continuous verification system. (Table 1).

Table 1. Compliance of the quality characteristics of the software product and types of test checks

Quality Characteristics	What is usually checked	Which checks are most often used
Functional suitability	The correct performance of functions in accordance with the established requirements	Functional tests, regression tests, acceptance checks
Productivity and efficiency	Response time, bandwidth, and resource usage	Load testing, profiling, and performance tests
Reliability	Fault tolerance and fast recovery from failures	Fault tolerance tests, recovery checks, incident monitoring
Safety	No known vulnerabilities or insecure settings	Static and dynamic analysis, dependency scanning, configuration checking
Maintainability	The ability to analyze, modify, and test	Static analysis, difficulty level verification, testing, and review
Compatibility	Harmonious coexistence and interaction with other components	Integration and contract (API) tests
Ease of use	Meeting user expectations and ergonomic requirements	User testing, accessibility checks
Portability	The ability to migrate between environments and platforms	Build and deployment checks in target environments, as well as environment tests

A source: author's development

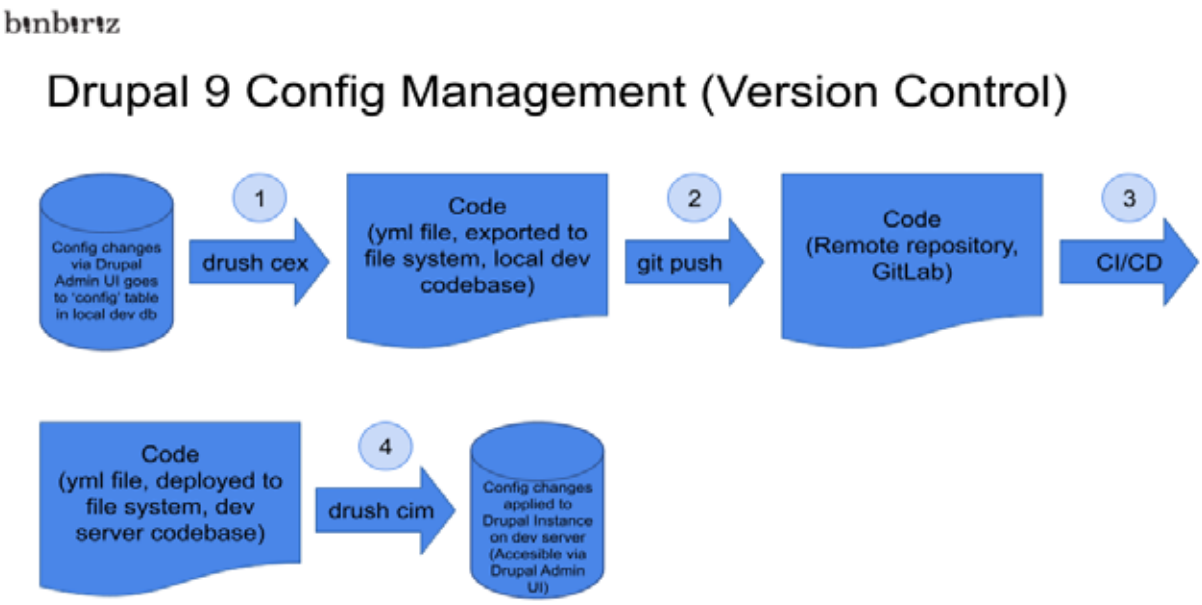
In practice, the effectiveness of continuous testing in corporate development is often assessed by the results of timely delivery and stability of changes. DORA (DevOps Research and Assessment) uses metrics that reflect the speed of change implementation and the stability of deployments. Based on these data, the research identifies performance clusters that allow you to see differences between groups of organizations in the ranges of values of key indicators (Design for a «flow diagram» to workflow configuration page).

Corporate web systems built on Drupal are not just a CMS website, but a full-fledged software platform. Business logic, integrations, and content management processes are implemented through its core, plug-ins, themes, and external libraries. For testing purposes, this means that the quality control object covers several levels: own project code (custom modules); community code; Drupal core; a set of dependencies of the PHP ecosystem, managed through Composer. During the operation of enterprise solutions, updates to the core and modules, changes in dependencies, and configuration discrepancies between environments become important sources of regression. Therefore, testing should include not only functional scenarios, but also build processes, dependency management, and configuration validation (Using Composer to Install Drupal and Manage Dependencies).

One of the key features of Drupal as an enterprise platform is advanced configuration management. Configuration objects that are stored and transferred between environments

through the configuration export and import mechanism (including the configuration synchronization interface) determine a significant part of the system's behavior. In practice, this means that changes in settings such as content types, fields, views, roles and rights, as well as module parameters, can be the same sources of regression as changes in the code. Therefore, the integration of testing should take into account the need to monitor configuration changes and their correctness when importing to test benches. Figure 1 shows an example of visualizing the configuration flow (export/import) as a typical process, which may be associated with risks of environment discrepancies and configuration migration errors.

Figure 1. Example of a Drupal configuration management workflow (export/import configuration) (Drupal Configuration Management Technical Stuff)



To organize automated testing, Drupal offers various types of PHPUnit-based tests, as well as recommended base classes for writing new ones. This is especially important for enterprise systems, as it allows you to create a multi-level verification model: from fast unit tests that test isolated logic, to functional and browser tests that test the entire system over HTTP, as well as tests with JavaScript behavior. This classification of tests provides a formal basis for selecting the minimum required set of checks for each change, as well as for separating tests by execution time and risk level.

To organize automated testing, Drupal offers various types of PHPUnit-based tests, as well as recommended base classes for writing new ones. This is especially important for enterprise systems, as it allows you to create a multi-level verification model: from fast unit tests that test isolated logic, to functional and browser tests that test the entire system over HTTP, as well as tests with JavaScript behavior. This classification of tests provides a formal basis for selecting the minimum required set of checks for each change, as well as for separating tests by execution time and risk level (Drupal core release schedule).

Finally, the specifics of corporate use of Drupal often include complex processes of working with content: stages of preparation and publication, roles and checks. Therefore, testing should cover not only pages and forms, but also business processes of approval and publication. As an example, below is a diagram illustrating the stages of the publishing process: draft → under review → published. This diagram clearly shows why regression can manifest itself not only in "page crashes", but also in violation of transitions and access rights at different stages of the content lifecycle (Figure 2).

Figure 2. An example of a publishing workflow diagram (content states and transitions) (Design for a «flow diagram» to workflow configuration page)

In the process of integrating continuous testing for Drupal, a key organizational decision is to link checks to each change that goes through the code review process. This allows you to confirm the quality of the changes even before they are combined and deployed. The SDLC (Lifecycle Management System) integration model is built around so-called "control points". These include building and installing dependencies, running automated tests, publishing test results, and deciding whether to allow changes to merge or release. An additional practical basis for this model is provided by the regulations for the release of kernel updates. Drupal patch versions are released monthly and are aimed at secure updates. Therefore, in corporate development, it is logical to accompany regular updates of the core and modules with a regular automated regression run.

Table 2 captures the "end-to-end" model in terms of SDLC: what exactly is embedded in the development flow and what results should be preserved so that the decision to promote the build is verifiable and reproducible.

Table 2. Continuous Testing integration model in SDLC for Drupal (verification flow and artifacts)

A stage in the development and delivery flow	What is controlled	What is saved as a result
Code/Configuration change → Merge request	The ability to build a project and install dependencies in accordance with the established rules (for projects using Composer)	Build log, fixed set of dependencies
Automated checks on the merge request	Running tests when changes are made and/or during the process of sending a merge request	Test reports, pipeline status, run artifacts
Release/update preparation (including kernel patch updates)	A pre-release regression check, given that patches are released regularly	Protocol of the run result, list of changes
Deployment	Using a verified build and confirming successful deployment	Build version, deployment log, result of post-checks

A source: (Release process overview)

Usually, the implementation of a continuous testing system (CI/CD) in Drupal begins with the project being created and maintained as managed using Composer, using official templates. This allows you to update and install dependencies via Composer. To automate tests, Drupal offers a PHPUnit infrastructure and documentation for running tests from the command line, including JavaScript tests. This allows you to create a pipeline in which tests will be automatically run on the CI server with each change. The Drupal ecosystem has documentation on enabling GitLab CI for projects on Drupal.org. And in the Drupal core repository, you can find the GitLab CI configuration (.gitlab-ci.yml), which confirms the practical use of this model.

As a result, the CI/CD continuous testing contour for Drupal boils down to a reproducible assembly of the Composer project, automatic test runs in the pipeline with each change, saving reports/artifacts of runs and applying these results as a formal condition for allowing the change to merge and further deploy.

The development prospects are related to the expansion of continuous testing capabilities beyond functional regression. First, special attention is paid to security. Regular checks of dependencies, configurations, and typical vulnerabilities during development, as well as testing of access rights and data processing scenarios, are becoming increasingly important. Secondly, the role of observability is increasing. Metrics, logs, and traces are used not only during operation, but also as a source of information to identify hidden defects and problems that arise after changes. Thirdly, infrastructure testing is actively developing. Infrastructure as code, reproducibility of environments, and correctness of deployments are checked. Finally, the use of AI for quality support (QA) is a promising area. Artificial intelligence helps to create test cases and data, analyze the causes of crashes, prioritize regressions, and identify unstable tests.

Conclusions

Thus, the integration of continuous testing into the development process of corporate web systems on Drupal becomes a necessary step for the controlled release of updates and reducing the risk of problems. The specifics of Drupal, including dependence on the core and modules, dependency management through Composer, as well as the important role of configurations and content creation processes, require testing not only functional scenarios, but also changes in dependencies and configurations between environments. The end-to-end testing model, based on automating checks with each change and recording the results of these checks as a formal condition for merging and deployment, allows you to make releases more predictable and increase the stability of the system during operation.

Development prospects are associated with increasing the level of security during the development process, the introduction of observability technologies to detect hidden defects, infrastructure testing, and the use of artificial intelligence tools to support quality control processes.

References

What is Continuous Testing? [Electronic resource]. – Access mode: https://qarocks.ru/continuous-testing/.

Design for a «flow diagram» to workflow configuration page [Electronic resource]. – Access mode: https://www.drupal.org/project/drupal/issues/2758621.

DORA's software delivery performance metrics [Electronic resource]. – Access mode: https://dora.dev/guides/dora-metrics/.

Drupal Configuration Management Technical Stuff [Electronic resource]. – Access mode: https://binbiriz.com/docs/drupal/drupal-configuration-management/.

Drupal core release schedule [Electronic resource]. – Access mode: https://www.drupal.org/about/core/policies/core-release-cycles/schedule.

Release process overview [Electronic resource]. – Access mode: https://www.drupal.org/about/core/policies/core-release-cycles/release-process-overview.

Using Composer to Install Drupal and Manage Dependencies [Electronic resource]. – Access mode: https://www.drupal.org/docs/develop/using-composer/manage-dependencies.

submitted 14.12.2025;
accepted for publication 28.12.2025;
published 30.01.2026

Contact: e.kuletskaia@gmail.com