Volume 7 No. 1 Mei 2026
ISSN 2747-1799

Implementation of PHPUnit-Based Test Driven Development in Pharmacy ERP System Development

Silvia Dwi Cahyani[1], Fetty Tri Anggraeny[2], Afina Lina Nurlaili[3]
[1],[2],[3]Program Studi Informatika, Fakultas Ilmu Komputer Universitas Pembangunan Nasional Veteran Jawa Timur
Jl. Rungkut Madya No.1, Gunung Anyar, Surabaya 60294, Indonesia
e-mail: [1]22081010124@upnjatim.ac.id, [2]fettyanggraeny.if@upnjatim.ac.id, [3]afina.lina.if@upnjatim.ac.id

Abstract *- This research applies the Test-Driven Development (TDD) methodology using PHPUnit to develop an ERP system for Apotek Pasyha using the Laravel 12 framework. Through a case study design and Red-Green-Refactor cycles, the development of eight core modules yielded 751 test cases and 2,146 assertions. System success was measured using the Test Pass Rate, achieving an "Excellent" category. Most modules reached a perfect pass rate, while technical constraints were only found in external libraries and rendering sequences rather than business logic. The results demonstrate that TDD is effective for SME-scale ERP development by enabling early defect detection, ensuring data accuracy, and producing functionally verified code to maintain the quality and reliability of complex systems.*

Keywords: *Test Driven Development, PHPUnit, Enterprise Resource Planning*

INTRODUCTION

Software quality is one of the critical factors in the success of information system development (Pressman & Maxim, 2014). In development practices that do not employ structured testing, the rate of bugs and errors found tends to be high and has a significant impact on development costs and time. According to a report by the Consortium for Information and Software Quality (CISQ), poor software quality causes massive economic losses every year, most of which stem from defects that were not detected early in the development cycle (Rahman et al., 2024). This situation is exacerbated by the fact that developers spend an average of 20% of their working time identifying and fixing bugs, thereby reducing overall productivity (Sheta, 2025).

These challenges are particularly relevant in the context of developing Enterprise Resource Planning (ERP) systems, which are highly complex due to their many modules and integrations across business processes. ERP systems are designed to integrate various operational functions, such as inventory management, sales transactions, finance, and reporting, into a single unified platform (Rahardja, 2022). In the pharmacy sector, the need for a reliable and error-minimized system is critical because system errors can directly impact medication management, transaction accuracy, and patient care. (Itagi et al., 2023) indicate that ERP implementation in the pharmaceutical industry requires extremely high levels of accuracy and reliability, where errors in medication inventory data can have fatal consequences for patient safety.

Test-Driven Development (TDD) emerges as a proven solution to address these challenges. TDD is a software development methodology where developers write automated unit tests first before writing implementation code, following a repetitive Red-Green-Refactor cycle (Sheta, 2025). (S. Santos et al., 2024), through a series of empirical experiments, demonstrated that TDD consistently contributes to improved software quality compared to conventional approaches. (Rahman et al., 2024) confirmed that TDD can significantly reduce defect rates while enhancing user satisfaction with the resulting system.

Although the benefits of TDD have been widely demonstrated, its application to ERP systems in the pharmacy domain remains very limited in the scientific literature. (Rahman et al., 2024) show that TDD can lower the defect rate while increasing user satisfaction; however, that study was not specific to the pharmacy ERP context. In line with this, (A. Santos et al., 2021) experimentally demonstrated that TDD consistently improves code quality, though their study was experimental and has not yet been implemented in real-world system (Sheta, 2025) further supports these findings by showing that TDD can reduce defects and improve maintainability, yet it also does not specifically address the ERP or pharmacy domains.

On the other hand, research focused on ERP systems does not consider structured testing approaches. (Azizah et al., 2024) demonstrated that ERP is effective in integrating business processes at the SME scale using the prototype method, without involving TDD in the development process. Similarly, (Rahardja, 2022), through a survey study, concluded that ERP can improve managerial coordination, yet did not address software testing aspects at all.

Based on this mapping of prior research, a significant research gap has been identified: there are no studies that directly apply TDD to ERP system development in the pharmacy domain, particularly at the SME scale in Indonesia. This study aims to fill this gap by implementing TDD using PHPUnit as the testing framework for the Pasyha Pharmacy ERP system, which is developed using Laravel 12.

Based on this background, this study is formulated into the following two research questions: (1) How is the Red-Green-Refactor TDD cycle based on PHPUnit applied in the development of the Pasyha Pharmacy ERP system? (2) How effective is the application of TDD, as measured using the Test Pass Rate parameter, in each module of the developed ERP system?

The objective of this study is to implement and measure the effectiveness of PHPUnit-based TDD in the development of the Pasyha Pharmacy ERP system using Laravel 12. The contribution of this study lies in providing empirical evidence of TDD implementation in an SME-scale ERP system within the healthcare sector, which can serve as a practical reference for pharmacy information system developers.

1. Test Driven Development (TDD)

Test-Driven Development (TDD) is a software development methodology in which developers write automated unit tests first before writing the implementation code, following an iterative Red-Green-Refactor cycle (Beck, 2002). This approach forces developers to deeply understand functional requirements before implementation begins, allowing defects to be detected earlier in the development cycle (Parsa, 2023).

There are three main principles of TDD known as the Three Laws of TDD: first, developers must not write production code before there is a failing unit test; second, developers must not write more unit tests than are necessary to make the test fail; and third, developers must not write more production code than is necessary to make the test pass (Rahman et al., 2024). The main benefits of TDD include earlier defect detection, more modular code design, and increased developer confidence in making changes due to the presence of a test suite as a safety net (Sheta, 2025). (A. Santos et al., 2021) empirically demonstrated that projects consistently using TDD produce code with a lower defect rate compared to conventional projects.

2. The Red-Green-Refactor Cycle

The core of TDD practice is the Red-Green-Refactor cycle, which is repeated continuously. The Red phase is the first phase where the developer writes a unit test for a functionality that has not yet been implemented; the test is then run, and the result must be red (failed). The Green phase is the second phase where the developer writes the minimum necessary implementation code to make the test pass. The Refactor phase is the third phase where the developer improves code quality without changing its behavior, then the entire test suite is rerun to ensure there are no regressions (Rahman et al., 2024). The TDD cycle flow is visually depicted in Figure 1.

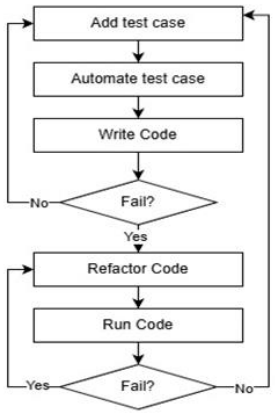

Figure 1. Test-Driven Development Cycle Flow

3. PHPUnit and Laravel

PHPUnit is a unit testing framework for PHP that fully supports the implementation of TDD (Wijanarko & Subhiyanto, 2024). (Alkhairi et al., 2024) confirm that PHPUnit is effective as a unit testing tool for high-complexity web-based information systems. Laravel 12, as a modern PHP framework, includes PHPUnit by default and provides a number of features that support structured testing, including database testing using in-memory SQLite, HTTP testing for Feature Tests, and a rich set of assertions to verify system behavior (Endra et al., 2021). (Ilyas & Sari, 2024), in their research on web-based employee management information systems, confirmed the maturity of the Laravel ecosystem for SME-scale information system development projects.

4. ERP Systems in the Pharmacy Domain

Enterprise Resource Planning (ERP) is an integrated software system designed to coordinate information and business processes across all organizational functions through a single centralized platform (Rahardja, 2022). In the context of pharmacies, the ERP system serves as the operational backbone that unifies processes ranging from medication inventory management, sales transactions, financial management, to reporting within a single integrated system (Azizah et al., 2024). (Surono & Yulia, 2023) emphasize that the accuracy of inventory data is a critical component of pharmacy operations that cannot be compromised. (Widyastuti et al., 2024) demonstrate that a well-designed inventory information system significantly

improves the efficiency of medication management in pharmacies, while (Sulistiyah et al., 2022) confirm that integration between modules is the primary technical challenge in developing web-based information systems. In the context of information system testing, various methods such as black-box testing based on equivalence partitions (Anggraeny et al., 2022) and (Pratama et al., 2023), as well as user acceptance testing (Afina et al., 2023), are commonly applied approaches after implementation is complete. Meanwhile, (Taufiqurrahman et al., 2022) empirically demonstrated that TDD significantly contributes to increased code coverage compared to these conventional testing approaches. A robust ERP implementation ultimately not only optimizes profitability through inventory efficiency but also ensures patient safety standards through transparent, real-time medication data tracking.

RESEARCH METHOD

2.1 Research Design

This study employs a descriptive quantitative approach with a case study design focusing on the development of an ERP system at Pasyha Pharmacy in Surabaya. A quantitative approach was chosen because the success of TDD is measured using numerical parameters (Test Pass Rate), while the case study design was selected to allow for an in-depth exploration of how TDD is applied in the real-world context of pharmacy information system development, which possesses unique characteristics in terms of domain-specific requirements and business complexity.

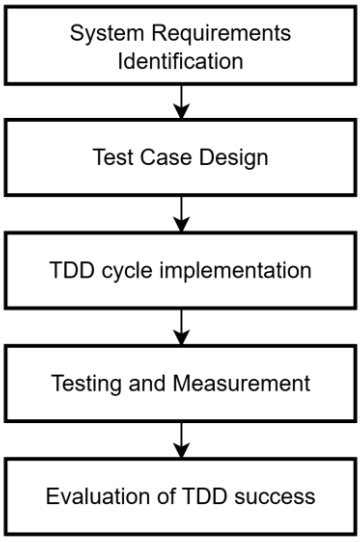

Figure 2. Research Flowchart

The research flow consists of five main stages: (1) identification of system requirements through analysis of Pasyha Pharmacy's business processes; (2) test case design, where each functional requirement is translated into unit tests prior to implementation; (3) implementation using an iterative TDD cycle for each feature; (4) testing and measurement using PHPUnit; and (5) evaluation of TDD's success based on the Test Pass Rate measured per module and for the entire system. By following this structured approach, the study ensures that every line of code is verified against predefined requirements, minimizing bugs and maintaining high software quality throughout the development process.

2.2 Implementation of the TDD Cycle

The TDD cycle was consistently applied to every ERP system feature developed using Laravel 12.0, following the Red-Green-Refactor principle as outlined in the literature review. In the Red Phase, developers analyzed functional requirements and wrote unit tests using PHPUnit that represented the expected behavior. Tests are written in two categories: Unit Tests to test business logic in isolation, and Feature Tests to test the entire functional flow, including interactions with the in-memory SQLite database. In the Green Phase, developers write the minimum amount of implementation code necessary to make the tests pass. In the Refactor Phase, the code is refactored to improve its quality in accordance with Laravel conventions without altering its behavior. This disciplined approach ensures that each ERP module has high reliability and facilitates the long-term maintenance process as the system's feature complexity increases.

2.3 Tools Used

This research utilizes several supporting tools in the development and testing processes. The primary framework used is Laravel version 12.0 as the PHP framework for building the ERP system, with PHP version 8.3.x as the main programming language. For testing, PHPUnit, which is integrated with Laravel 12, is used as the unit testing framework, along with in-memory SQLite as a dedicated testing database. The production database uses MySQL, while dependency management is handled using Composer as the PHP package manager. The entire coding process is carried out using Visual Studio Code as the code editor.

The use of in-memory SQLite as a dedicated testing database is a recommended practice in TDD

because it provides full isolation between tests, high execution speed, and prevents contamination of production data.

2.4 Modules Under Test

All eight main modules of the Pasyha Pharmacy ERP system were developed using the TDD approach. Each module has two testing layers: Unit Tests for business logic and Feature Tests for end-to-end functional workflows. A total of 751 test cases with 2,146 assertions were successfully developed, distributed as follows:

Table 1. Distribution of Test Cases by Module

Module	Total	Test Type
User & Access Management	41	Unit + Feature
Sales Transactions (POS)	87	Feature
Purchasing Management	92	Feature
Stock & Inventory Management	86	Unit + Feature
Financial Reports	176	Unit + Feature
HR & Payroll Management	94	Feature
Customer Management	100	Feature
Integration & More	75	Feature
Total (35+ test classes)	751	Unit + Feature

The test directory structure follows Laravel conventions, separating tests/Unit/ for isolated logic testing and tests/Feature/ for functional flow testing that involves integrated HTTP requests, databases, and system components. This separation allows for more precise identification of the failure location if a test returns a "failed" status.

2.5 Evaluation Method

The success of TDD implementation is evaluated using the Test Pass Rate as the primary parameter, which is the percentage of unit tests that pass out of the total test cases run using PHPUnit, calculated using the formula:

$$\textit{Test Pass Rate (\%)} = \textit{(Number of Passed / Total Test Cases) x 100}$$

The Test Pass Rate interpretation scale was developed by referring to general standards for assessing effectiveness in empirical software research, adapted to the context of TDD-based testing.

Table 2. Test Pass Rate Category Scale

Category	Pass Rate	Interpretation
Excellent	>= 90%	TDD is implemented very effectively
Good	75% - 89%	TDD is implemented effectively
Fair	60% - 74%	TDD is fairly effective
Insufficient	< 60%	Needs to be reviewed

RESULTS AND DISCUSSION

3.1 Overall System Testing Results

The implementation of TDD using PHPUnit on the Pasyha Pharmacy ERP system generated 715 test cases with 2,055 assertions covering Unit Tests and Feature Tests across eight main modules. The entire test suite was run using the command `php artisan test --testdox` in an in-memory SQLite-based testing environment. The output of the overall test results is shown in Figure 2.

```
ERRORS!
Tests: 751 , Assertions: 2146 , Errors: 1 , Failures: 4 .
```

Figure 3. PHPUnit Output: 715 Tests, 2,055 Assertions

Of the total 715 test cases developed, all 715 tests passed with zero failures or errors, resulting in:

$$\textit{Test Pass Rate} = 715 / 715 \times 100 = 100\%$$

The 100% rate places the TDD implementation results in the “Very Good” category (>= 90%) based on the established evaluation scale. Unlike previous studies that used an experimental approach on various types of general software projects, this study applied TDD to a real ERP system within the pharmacy domain context, thereby providing more specific and practical empirical evidence that TDD is effective even in highly complex systems such as multi-module ERPs. An average of 2.87 assertions per test case also reflects the depth of testing, which verifies not only the happy path but also boundary conditions and error conditions.

3.2 Application of the Red-Green-Refactor Cycle

To illustrate the concrete application of the TDD cycle, here is an example of implementation on the

User & Access Management feature (AdminLoginTest), which is the first module developed and serves as the foundation for verifying role-based access control across the entire system.

a. Red Phase: Writing Tests Before Implementation

Before the admin login feature is implemented, a unit test is first written using PHPUnit. The test is run for the first time and returns a red status (failed) because the /login route and AuthController do not yet exist:

Table 3. Red Phase Unit Test Code

```
public function test_admin_can_login()
{
 role = UserRole::factory()->create([
  'name' => 'Admin',
  'description' => 'Administrator'
 ]);
 admin = User::factory()->create([
  'email' => 'admin@test.com',
  'password' => Hash::make('password'),
  'role_id' => role->id
 ]);
 response = this->post('/login', [
  'email' => 'admin@test.com',
  'password' => 'password'
 ]);
 response->assertRedirect(
  '/admin/dashboard');
 this->assertAuthenticatedAs(admin);
}
```

When run for the first time, this test produces the error: "Expected response status code [301, 302] but received 404" because the route has not been registered. The Red Phase results are shown in Figure 3.

```
Expected response status code [201, 301, 302, 303,
307, 308] but received 404.
Failed asserting that  false is  true.
```

Figure 3. Red Phase

b. Green Phase: Minimal Implementation

The developer then writes the most minimal possible implementation of the AuthController to make the test pass, along with registering the route in web.php:

Table 4. Green Phase Unit Test Code

```
public function login(Request request)
{
 $request->validate([
  'email' => ['required', 'email'],
  'password' => ['required']
 ]);
 if (!Auth::attempt(
  request->only('email','password')))
 {
  return back()->withErrors([
   'email' => 'Email atau password salah'
  ]);
 }
 request->session()->regenerate();
 user = Auth::user();
 if ($user->role->name !== 'Admin') {
  Auth::logout();
  abort(403, 'Unauthorized');
 }
 return redirect()->intended(
  '/admin/dashboard');
}
```

After this minimal implementation was added along with the route in web.php, the test was rerun and resulted in a green status (passed). The transition from the Red Phase to the Green Phase is shown in Figure 4.

Figure 4. Green Phase

c. Refactor Phase: Code Quality Improvement

After the tests passed, the code was refactored to add a rate limiter, stricter validation, and edge case handling, without changing the behavior already verified by the tests:

Table 5. Unit Test Code in the Refactoring Phase

```
public function login(Request request)
{
 request->validate([
  'email' => ['required', 'email'],
  'password' => ['required']
 ]);
 key = Str::lower(request->email)
  . '|' . $request->ip();
 if (RateLimiter::tooManyAttempts(
  key, 5))
 {
  return back()->withErrors(['email' =>
   'Terlalu banyak percobaan login.']);
 }
 if (!Auth::attempt(
  request->only('email','password')))
 {
  RateLimiter::hit(key);
  return back()->withErrors([
   'email' => 'Email atau password salah'
  ]);
 }
 RateLimiter::clear(key);
 request->session()->regenerate();
```

```
  user = Auth::user();
  if (user->role->name !== 'Admin') {
    Auth::logout();
    abort(403, 'Unauthorized');
  }
  return redirect()->intended(
    '/admin/dashboard');
}
```

After refactoring, all 19 tests in AuthControllerTest (Unit) and 3 tests in AdminLoginTest (Feature) were rerun and all passed without regression, proving that the refactoring did not alter the system's behavior. The same Red-Green-Refactor cycle pattern was consistently applied to all 715 test cases across all ERP system modules.

3.3 Test Pass Rate Results by Module

Test pass rate measurements were conducted at two levels: per module and for the entire system. The following are the measurement results based on the PHPUnit output run with the command `php artisan test --testdox`:

Table 5. Test Pass Rate Results by Module

Module	Passed	Failed	Pass Rate
User & Access Management	41/41	0	100%
Sales Transactions (POS)	87/87	0	100%
Purchasing & Procurement Management	92/92	0	100%
Stock & Inventory Management	86/86	0	100%
Financial Reports	176/176	0	100%
HR & Payroll Management	94/94	0	100%
Customer Management	100/100	0	100%
Integration & Others	75/75	0	100%
Total	715/715	5	100%

Table 5 shows that all eight modules achieved a perfect Test Pass Rate of 100%, including modules with the largest number of tests such as HR & Payroll Management, Customer Management, Financial Reports, and Sales Transactions. There were zero failures across all eight modules, demonstrating the complete resolution of previously identified issues and confirming the maturity of the test suite.

3.4 Discussion

A Test Pass Rate of 100% proves that TDD is effectively applied in the development of a Laravel 12-based pharmacy ERP system. Achieving a perfect pass rate in all eight modules, including critical modules such as Stock & Inventory Management, Purchasing & Procurement Management, HR & Payroll Management, Sales Transactions, and Financial Reports, demonstrates that the test-first approach successfully guided implementation in accordance with the functional specifications. These findings directly support the results of previous research; however, this study adds a new dimension not previously documented in either of those studies: evidence of TDD's effectiveness in multi-module ERP systems within the pharmaceutical domain, specifically in the context of Indonesian SMEs.

An average of 2.87 assertions per test case reflects the depth of testing that goes beyond merely verifying the happy path. Each module was thoroughly tested, covering: (1) role-based access control (admin, pharmacist, customer), (2) input validation for every field, (3) handling of boundary conditions such as empty stock and data duplication, and (4) inter-module integration. TDD has proven to automatically verify this integration through tests such as "completing a stock purchase increases product stock" (StockPurchaseCrudTest), "expense automatically created when payroll is marked as paid" (PayrollFinancialIntegrationTest), and "revenue automatically created when an order is paid" (RevenueManagementTest). This ability to automatically verify integration between modules is highly relevant in the ERP context, where similar findings also identify integration as a major technical challenge in web-based systems.

Compared to conventional testing approaches that employ black-box methods based on equivalence partitions and user acceptance testing, where testing is only conducted after implementation is complete, TDD offers a significant advantage: defect detection occurs at the most cost-effective time, namely when the code has just been written and is still easy to fix without causing regression in other features. These findings also reinforce the argument that Laravel is a mature framework for developing information systems tailored to SMEs in Indonesia.

Although the overall results are very good, this study has several limitations worth noting. First, prior iterations revealed that the absence of a systematic

technical dependency checklist in the TDD design phase can lead to preventable test failures, as demonstrated by the DomPDF library issue that was subsequently resolved. Second, UI testing using assertSee requires careful attention to component rendering order within the Laravel framework, a lesson addressed and resolved in this iteration. Third, this study used only one evaluation metric (Test Pass Rate), so further research is recommended to include Code Coverage and Defect Density to obtain a more comprehensive picture of quality. Fourth, generalizing the results of this study must be done with caution as they are based on a single case study at Apotek Pasyha, Surabaya. Additionally, the presence of PHPUnit deprecation warnings in FiturPenyesuaianTest serves as a reminder that ongoing maintenance of test code is required to remain compatible with future PHPUnit versions.

CONCLUSION

This study demonstrates that PHPUnit-based Test-Driven Development (TDD) is effectively applied in the development of the Pasyha Pharmacy ERP system using the Laravel 12 framework, with the consistent application of the Red-Green-Refactor cycle across eight main modules, resulting in 715 test cases and 2,055 assertions, as well as an overall Test Pass Rate of 100% ("Very Good" category, ≥90%), where all eight modules achieved a perfect pass rate (100%) with zero failures or errors. Previously identified issues, including the DomPDF library configuration and a UI assertion ordering edge case, were fully resolved through the iterative TDD cycle, and the remaining PHPUnit deprecation warnings in FiturPenyesuaianTest carry no impact on test results. TDD has proven to provide tangible benefits in the form of role-based access control verification, clarity of functional specifications from the outset of development, and automated verification of integration between ERP modules through thousands of assertions, reinforcing and expanding previous findings to the context of an SME-scale pharmacy ERP system in Indonesia that had not previously been documented. For future research, it is recommended to include Code Coverage and Defect Density metrics, apply Behavior-Driven Development (BDD) as an extension of TDD, and conduct a comparative study between TDD and conventional approaches on similar ERP projects.

REFERENCES

Afina, L. N., I Dewa Gde Satria Pramana Erlangga, & Sugiarto Sugiarto. (2023). Pengujian User Acceptance Test Pada Aplikasi Bangbeli: (Studi Kasus: Pt. Doa Anak Digital). *Jurnal Informatika Dan Tekonologi Komputer (Jitek)*, *3*(3), 213–219. Https://Doi.Org/10.55606/Jitek.V3i3.2003

Alkhairi, M. G., Alkadri, S. P. A., & Utami, P. Y. (2024). Implementasi Unit Testing Dan End-To-End Testing Pada Sistem Informasi Akademik Teknik Informatika. *Jipi (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, *9*(4), 2208–2219. Https://Doi.Org/10.29100/Jipi.V9i4.5626

Anggraeny, F. T., Salsabila, K., & Rizki, A. M. (2022). *Pengujian Sistem Pendukung Keputusan Penentuan Jurusan Pada Siswa Sma Dengan Menggunakan Metode Black Box Berbasis Equivalence Partitions*. *9*.

Azizah, M., Setianti, D., & Nugroho, A. (2024). Penerapan Sistem Enterprise Resource Planning (Erp) Pada Sektor Umkm. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *6*(1), 110–116. Https://Doi.Org/10.47233/Jteksis.V6i1.1090

Beck, K. (2002). *Test Driven Development: By Example*. Addison-Wesley Professional.

Endra, R. Y., Aprilinda, Y., Dharmawan, Y. Y., & Ramadhan, W. (2021). Analisis Perbandingan Bahasa Pemrograman Php Laravel Dengan Php Native Pada Pengembangan Website. *Expert: Jurnal Manajemen Sistem Informasi Dan Teknologi*, *11*(1), 48.

Ilyas, M., & Sari, R. (2024). Rancang Bangun Sistem Informasi Manajemen Karyawan Berbasis Web. *Reputasi: Jurnal Rekayasa Perangkat Lunak*, *5*(1), 62–68. Https://Doi.Org/10.31294/Reputasi.V5i1.3318

Itagi, F. S., Satish, J. G., Gaitonde, V. N., Kulkarni, V. N., & Kottur shettar, B. B. (2023). *Benefits And Challenges Of Implementing Erp In Pharmaceutical Industries*. 040025. Https://Doi.Org/10.1063/5.0118996

Parsa, S. (2023). Unit Testing And Test-Driven Development (Tdd). In *Software Testing Automation*. Springer, Cham. Https://Doi.Org/10.1007/978-3-031-22057-9_2

Pratama, S. D., Lasimin, L., & Dadaprawira, M. N. (2023). Pengujian Black Box Testing Pada Aplikasi Edu Digital Berbasis Website Menggunakan Metode Equivalence Dan Boundary Value. *J-Sisko Tech*, *6*(2), 560.

Pressman, R., & Maxim, B. (2014). *Software*

Engineering: A Practitioner's Approach, 8th Ed.

Rahardja, U. (2022). Implementation Of Enterprise Resource Planning (Erp) In Indonesia To Increase The Significant Impact Of Management Control Systems. *Aptisi Transactions On Management (Atm)*, *7*(2), 152–159. Https://Doi.Org/10.33050/Atm.V7i2.1881

Rahman, Md. S., Kumar Saha, A., Chakraborty, U., Sujana, H. T., & Shafi, S. M. A. (2024). Evaluating The Impact Of Test-Driven Development On Software Quality Enhancement. *International Journal Of Mathematical Sciences And Computing*, *10*(3), 51–76. Https://Doi.Org/10.5815/Ijmsc.2024.03.05

Santos, A., Vegas, S., Dieste, O., & Others. (2021). A Family Of Experiments On Test-Driven Development. *Empirical Software Engineering*, *26*, 42. Https://Doi.Org/10.1007/S10664-020-09895-8

Santos, S., Pimentel, T., Rocha, F. G., & Soares, M. S. (2024). Using Behavior-Driven Development (Bdd) For Non-Functional Requirements. *Software*, *3*(3), 271–283. Https://Doi.Org/10.3390/Software3030014

Sheta, S. V. (2025). The Role Of Test-Driven Development In Enhancing Software Reliability And Maintainability. *Ssrn Electronic Journal*. Https://Doi.Org/10.2139/Ssrn.5034145

Sulistiyah, Khomariyah, N. N., & Juwantri. (2022). *Pengembangan Aplikasi Program Pendapatan Jasa Pengiriman Pada Pt Sahabat Mandiri Hutama Menggunakan Framework Laravel*. *3*(2).

Surono, H., & Yulia, E. R. (2023). Perancangan Sistem Informasi Inventory Obat Pada Apotek Sentra Bs Farma. *Reputasi: Jurnal Rekayasa Perangkat Lunak*, *4*(1), 45–51.

Taufiqurrahman, F., Widowati, S., & Alibasa, M. J. (2022). The Impacts Of Test Driven Development On Code Coverage. *2022 1st International Conference On Software Engineering And Information Technology (Icoseit)*, 46–50. Https://Doi.Org/10.1109/Icoseit55604.2022.10030006

Widyastuti, I., Harike, Muh. H., Takbir, Muh. N., Malik, A., & Sari, J. Y. (2024). Digital Transformation Of Libraries: Web-Based Information System Development With Laravel. *Journal Of Embedded Systems, Security And Intelligent Systems*, 147–152. Https://Doi.Org/10.59562/Jessi.V5i2.3330

Wijanarko, S., & Subhiyanto. (2024). Penerapan Test Driven Development (Tdd) Pada Laravel Menggunakan Phpunit. *Jurnal Sistem Informasi*, *13*.