Time reduction of integration testing using concurrency

Kenan Sejmenović

Fakultet Informacijskih Tehnologija,
Univerzitet "Džemal Bijedić", Mostar,
Bosnia and Herzegovina
kenan.sejmenovic@edu.fit.ba

***Abstract*— Developing applications of enterprise size and enterprise qualities means robustness and complexities which come with their own set of challenges and priorities. Time constraint is always a parameter which leverage causes the most challenges and is most influential on the project outcome, thus every reduction of time in the development of enterprise applications decreases total project cost. Unit tests use mocking and usually test a single function (or unit of work), so as a result - the execution time is fast (e.g. 5000 tests in < 5 seconds). Integration tests imply a connection to the database, which is (often) populated by testing data. To preserve the integrity of data, every integration test makes and populates a database, which linearly raises the time needed for test execution. This paper showcases a method of drastically reducing the test execution time using concurrency.**

***Keywords*— Time reduction, integration testing, database, concurrency, parallelism, execution time.**

I. INTRODUCTION

Integration testing plays a crucial role in software development, ensuring that individual components work together seamlessly. However, as projects scale in complexity, the time required for comprehensive integration testing can become a significant bottleneck in the development lifecycle. This challenge is particularly pronounced in projects leveraging enterprise technologies, where each test may incur substantial setup and teardown overhead.

In response to these challenges, this document explores a novel approach to reduce integration testing time using concurrency [1] techniques within the xUnit testing framework. Concurrency, the ability to execute multiple tasks simultaneously, is a powerful concept that can be leveraged to parallelize test execution and optimize resource utilization. By employing concurrency at various levels of integration testing, from database setup to test execution, significant time savings can be achieved without compromising test integrity.

One of the key strategies employed in this approach is the creation of a new database instance for each test. Traditionally, integration tests share a single database instance, leading to contention and potential bottlenecks. However, by utilizing concurrency to create isolated database environments per test, contention can be eliminated, and greater parallelism can be achieved, thereby reducing overall test execution time.

Furthermore, the use of xUnit's concurrency features allows us to maximize parallelism by executing tests in separate classes concurrently. This ensures that the resources of the testing environment are fully utilized, harnessing the power of modern multicore processors to expedite test runs. By organizing tests into separate classes, each with its unique database instance, a higher degree of concurrency can be achieved, consequently, faster test execution.

The internal architecture of xUnit, a popular testing framework for unit testing in software development, is meticulously designed to ensure robustness and flexibility. At its core, xUnit adheres to a modular structure that is built around several key components: Test Runner, Test Framework, and Test Execution. The Test Runner is responsible for discovering and executing tests, often leveraging reflection to identify test methods and attributes dynamically. The Test Framework component encapsulates the various test lifecycle events, such as setup, teardown, and execution. This separation of concerns allows xUnit to maintain a clear distinction between test discovery and execution, facilitating easier maintenance and extension. The framework also employs a set of predefined attributes and conventions to mark test methods, setup/teardown methods, and test fixtures, ensuring that tests are written in a consistent and predictable manner.

Concurrency is another area where xUnit demonstrates sophisticated engineering. To enhance performance and reduce the overall test execution time, xUnit supports parallel test execution. This is particularly beneficial for large test suites where sequential execution would be time-prohibitive. The concurrency model in xUnit is managed through fine-grained control mechanisms that allow developers to specify the level of parallelism at various granularities, such as class-level or method-level. By leveraging asynchronous programming paradigms and thread-safe constructs, xUnit ensures that tests can run concurrently without interfering with each other. This is achieved through careful management of shared resources and synchronization points, ensuring that tests which require isolation or shared state are executed in a manner that prevents race conditions and other concurrency-related issues. Consequently, xUnit

provides a reliable and efficient testing environment that scales well with the complexity and size of modern software projects.

II. SOLUTION

Integration testing is a crucial aspect of software development, ensuring that different parts of a system work together seamlessly. However, it often comes with a significant time cost, especially when dealing with databases and external dependencies. To address this challenge, the author implemented a solution leveraging concurrency in the xUnit [2] testing framework within a .NET environment. This solution aimed to reduce the time required for integration testing while maintaining the reliability and effectiveness of the tests.

The first key aspect of the solution was the utilization of concurrent database creation for each integration test. By creating a new database instance for every test, the author ensured isolation and independence, preventing any interference or contamination between test cases. This approach not only enhanced the reliability of the tests but also significantly reduced the setup and teardown time typically associated with database operations in integration testing.

Furthermore, the author incorporated concurrency at the test execution level by organizing each integration test into its own class. This design choice allowed the author to maximize concurrency within the xUnit framework, enabling multiple tests to run concurrently without affecting each other. Leveraging the inherent parallel execution capabilities of xUnit, the author achieved a substantial reduction in the overall test execution time, thereby improving the efficiency of the testing process.

To facilitate the management and customization of concurrent database creation, the author implemented an abstract parent class that all integration test classes inherited from. This parent class contained overridden methods responsible for creating unique databases based on the provided class names. This design not only streamlined the database creation process but also enhanced the scalability and maintainability of the testing infrastructure.

In practical usage, developers can seamlessly integrate the concurrency-based approach into their integration testing workflows within a .NET environment. The process begins by structuring integration tests into individual classes, each representing a distinct test case or scenario. Developers then inherit from an abstract parent class that handles the creation of unique databases based on the class names, ensuring isolation and independence between tests. This method enhances test reliability by preventing interference and data contamination among different test scenarios. Additionally, this approach allows for more efficient parallel test execution, significantly reducing the overall testing time.

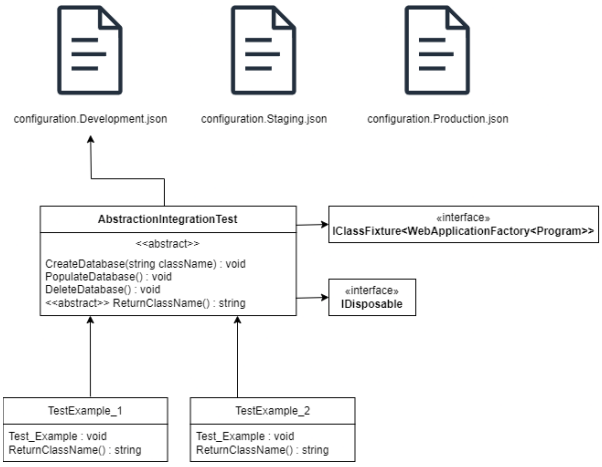

Fig. 1 Diagram of dependencies between configurations, abstract and subclasses of the testing model

To execute the integration tests, developers simply leverage the concurrency features provided by the xUnit testing framework. By default, xUnit supports parallel execution of test classes, allowing multiple tests to run concurrently without conflicts. This concurrency at the test execution level significantly reduces the overall test execution time, enabling developers to obtain rapid feedback on the system's behavior and functionality.

Despite the benefits of concurrency in integration testing, there are potential bottlenecks that developers should be mindful of during implementation. One such bottleneck is the management of shared resources, such as database connections or external services, under high concurrency scenarios. Without proper synchronization mechanisms or resource pooling strategies, excessive concurrent access to shared resources can lead to contention, increased latency, and potential data inconsistencies or errors in test results. Therefore, developers need to carefully design and optimize their concurrency strategies to mitigate these bottlenecks and ensure the reliability and accuracy of integration tests.

III. RESULTS

The implementation of concurrency in integration testing significantly reduced the overall execution time of the project's test suite. By leveraging xUnit's concurrency features, specifically the creation of distinct databases for each test and concurrent test execution, substantial improvements in testing time were observed. Prior to concurrency optimization, the integration tests, which connect to an MSSQL database, required approximately 8.3 minutes to complete. However, with the concurrency measures in place, the execution time was reduced to an impressive 44.2 seconds, representing a time reduction of over 90%.

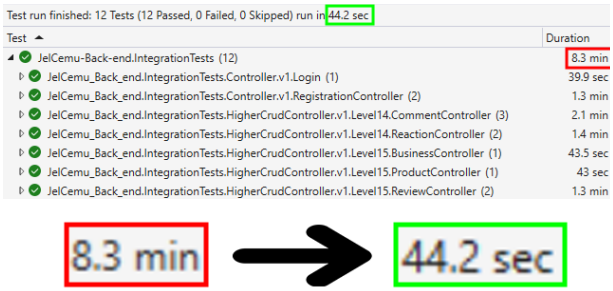

Fig. 2 Visual Studio 2022 Test explorer – integration test execution time

The approach of creating unique databases for each test class proved to be effective in parallelizing the test execution without resource conflicts. This strategy ensured that each test operated in isolation, mitigating potential interference between tests and database states. Additionally, organizing tests into separate classes further optimized concurrency within xUnit, allowing for maximum utilization of available resources during testing. As a result, the parallel execution of integration tests became a streamlined and efficient process, contributing significantly to the overall time reduction achieved.

Furthermore, the observed reduction in testing time not only enhances developer productivity but also promotes faster feedback loops during development cycles. With quicker test execution, developers can iterate more rapidly, leading to improved code quality and faster delivery of software updates. The successful implementation of concurrency in integration testing not only addresses performance challenges but also underscores the importance of leveraging modern testing frameworks and methodologies to optimize software development workflows.

The profound decrease in testing time not only augments developer productivity but also facilitates faster feedback loops throughout the development cycle. With the benefit of quicker test execution, developers can iterate more rapidly, which in turn enhances code quality and accelerates the delivery of software updates. The successful deployment of concurrency in integration testing addresses performance bottlenecks while highlighting the critical importance of leveraging contemporary testing frameworks and methodologies to optimize software development workflows.

Additionally, the implementation of these concurrency strategies underscores the potential for significant efficiency gains in other areas of software testing and development. By adopting such innovative approaches, development teams can achieve higher throughput, reduced bottlenecks, and a more agile response to changing requirements. Ultimately, the improved testing framework fosters a more robust and reliable software development process, ensuring that high standards of quality and performance are consistently met.

Moreover, the enhanced testing framework not only accelerates individual project timelines but also establishes a more sustainable and scalable approach for future development endeavors. The ability to execute tests concurrently and in isolation minimizes the risk of bottlenecks and deadlocks, which are common issues in traditional sequential testing environments. This improvement means that as the project grows in complexity and scope, the test suite can scale accordingly without a proportional increase in execution time. Consequently, the team can maintain high confidence in their codebase, even as new features and updates are rapidly integrated.

The adoption of concurrency in integration testing also sets a precedent for continuous improvement and innovation within the development team. By demonstrating the tangible benefits of modern testing techniques, this implementation encourages a culture of experimentation and refinement. Developers are more likely to explore and adopt other cutting-edge tools and methodologies, further enhancing the efficiency and quality of the software development lifecycle. This progressive approach not only benefits the current project but also serves as a valuable learning experience that can be applied to future projects, ultimately leading to more robust, reliable, and performant software solutions.

IV. FUTURE IMPROVEMENTS

Future improvements in this context could focus on enhancing the efficiency and scalability of the testing framework. One area to explore is optimizing the database creation process. While the current implementation creates a new database for each test to ensure isolation, this approach can lead to increased overhead, especially when dealing with a large number of tests. One potential improvement is to investigate using database snapshots or lightweight containers for faster setup and teardown of test environments. By minimizing the time spent on database creation, overall testing time can be further reduced.

Additionally, an investigation into the utilization of in-memory databases or lightweight alternatives to MSSQL for testing purposes could be beneficial. In-memory databases like SQLite or in-memory configurations of databases such as H2 for .NET offer faster data access times compared to traditional disk-based databases. By integrating such solutions into the testing framework, significant reductions in the time spent on database operations during test execution could be realized, thereby contributing to overall test speed improvements.

Another aspect to consider is fine-tuning the concurrency settings. While concurrent execution of tests can significantly reduce testing time, it's essential to balance between concurrency and resource utilization. Future iterations of the testing framework could include dynamic concurrency management based on system resources, such as CPU cores and memory availability [3]. This adaptive approach would optimize test execution for different hardware configurations, ensuring optimal performance across diverse environments.

Dockerization presents a promising avenue for enhancing the efficiency and portability of the integration testing solution. By containerizing the testing environment, developers can encapsulate dependencies, configurations, and the testing framework itself into a lightweight, isolated container. This approach not only streamlines the setup and teardown of test environments but also ensures consistency across different development and deployment environments. Furthermore, Docker enables easy scaling of test execution by leveraging container orchestration tools like Kubernetes, allowing for efficient utilization of resources and concurrent execution of tests on distributed clusters. Integrating Docker [4] into the testing workflow can simplify deployment pipelines, facilitate collaboration among team members, and enhance the overall reliability and reproducibility of integration tests. The use of separate MSSQL containers also simplifies the setup and teardown procedures traditionally associated with testing. In a non-containerized environment, preparing the database for testing often involves complex scripts to reset the database state, a process that can be error-prone and time-consuming. Docker streamlines this workflow by enabling the rapid creation and destruction of database containers. Each test can start with a pristine database state, ensuring consistency and reliability in test results. After the test completes, the container can be easily removed, freeing up resources and maintaining a clean environment. This level of automation reduces the manual overhead for developers and testers, allowing them to focus on more critical aspects of development.

Furthermore, the maintainability of the testing infrastructure is significantly enhanced through dockerization. Containers encapsulate not only the application but also its dependencies, including specific database versions and configurations. This encapsulation ensures that tests run in the same environment regardless of the underlying host system. Teams can version control their container configurations, making it easier to reproduce and diagnose issues. Any changes in the database schema or configuration can be managed through updates to the container image, ensuring that all team members are working with the same setup. This consistency is crucial for agile teams, where rapid iteration and continuous feedback are paramount.

Scalability is another critical benefit of using Docker containers for MSSQL in an agile context. As projects grow and the number of tests increases, managing a shared database environment can become a bottleneck. Docker containers, however, can be scaled horizontally. Multiple instances of the database container can be spun up on demand, distributing the load and ensuring that test execution remains efficient. This scalability ensures that the testing infrastructure can grow with the project without significant reconfiguration or investment in additional resources. Moreover, the ability to easily replicate containers aids in maintaining consistency across different environments. This approach simplifies the management of testing environments. By leveraging Docker containers, teams can achieve a more flexible and resilient testing setup.

V. CONCLUSION

In conclusion, the implementation of concurrency in integration testing for a .NET project utilizing MSSQL databases has resulted in a substantial reduction in testing time. By leveraging xUnit's concurrency features to create unique databases for each test and executing tests concurrently, the overall testing time has been significantly reduced by about 90%, and its maximum execution time is bound to the time of the longest test.

The approach of creating unique databases for each test has proven to be a pragmatic solution for managing test data dependencies and avoiding conflicts between tests. Each test operates within its own database environment, eliminating the need for complex setup and teardown procedures. This simplification not only streamlines the testing workflow but also enhances the maintainability and scalability of the testing infrastructure.

Overall, the successful implementation of concurrency in integration testing has demonstrated its efficacy in reducing testing time, enhancing test reliability, and improving development productivity. This approach can serve as a model for future testing strategies, showcasing how advanced techniques can be leveraged to optimize software testing processes and deliver high-quality software products efficiently.

This concurrency-based testing strategy is particularly beneficial for agile teams, which thrive on rapid iterations and continuous integration. Agile methodologies emphasize short development cycles and frequent releases, requiring a testing process that is both swift and reliable. By significantly reducing testing time, the concurrent testing approach aligns perfectly with the agile philosophy, enabling teams to integrate changes quickly and frequently without sacrificing quality. This enhances the team’s ability to respond to feedback, iterate on features, and deliver improvements continuously.

Moreover, the isolation of test environments ensures that tests can be run simultaneously without the risk of data contamination or dependency issues, which is crucial for maintaining the integrity of each iteration. Agile teams can thus maintain a high level of confidence in their testing results, knowing that each test accurately reflects the behavior of the code under test. This reliability is essential for sustaining the pace of agile development and for making informed decisions about when to release new features or updates.

The concurrency in testing not only supports faster turnaround times but also fosters a culture of continuous improvement and innovation within agile teams. Developers can experiment with new ideas and immediately validate their impact, facilitating a more dynamic and responsive development process. This agility and responsiveness are key competitive advantages in today’s fast-paced software industry, where the ability to quickly adapt to changing

requirements and market conditions can make a significant difference.

In essence, the adoption of concurrency in integration testing not only optimizes the technical aspects of testing but also enhances the overall agility of the development process. It empowers agile teams to deliver high-quality software at a faster pace, meeting the demands of modern development cycles and ensuring that products are both reliable and competitive. This approach exemplifies how leveraging advanced testing techniques can drive efficiency, innovation, and excellence in software development.

REFERENCES

[1] Parallel Programming Based on Microsoft.NET https://www.researchgate.net/publication/266646581_Parallel_Programming_Based_on_MicrosoftNET_TPL

[2] xUnit test patterns and smells: improving the ROI of test code https://www.researchgate.net/publication/221322013_XUnit_test_patterns_and_smells_improving_the_ROI_of_test_code

[3] Memory analysis of .NET and .Net Core applications https://www.researchgate.net/publication/361876455_Memory_analysis_of_NET_and_Net_Core_applications

[4] Testing using Testcontainers for .NET and Docker https://code-maze.com/csharp-testing-using-testcontainers-for-net-and-docker/