IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 2, November 2025
Impact Factor: 7.67

An Empirical Evaluation of a Model-Based Test Data Generation Framework for Reducing System Integration Delays

Datta Snehith Dupakuntla Naga

Abstract: *Modern software engineering has introduced a manual bottleneck in the setup of test data that slows down the Continuous Integration and Continuous Delivery (CI/CD) cycles, hence long integration cycles with the system and increased risks on quality. This paper empirically evaluates a novel Python-based model-driven framework designed to automatically generate complex relational state-aware test data. Parsing declarative models of data entities and their constraints will help us develop an approach to automatically create valid and semantically appropriate test datasets for large-scale enterprise projects from the financial and healthcare domains within their CI/CD pipelines. Changes in some of the key metrics were analyzed by way of a longitudinal case study, i.e., before and after implementation. It takes only 15 minutes compared to 8 hours earlier, leading to a reduction of 97% in time spent setting up the test environment. Also, by keeping data safe and ready, the plan helped cut down on 60% of production flaws tied to data problems. This fix made it possible to fully automate after-deployment checks and greatly improved the test range.*

Keywords*:* Model-Based Testing, Test Data Generation, Continuous Integration, Continuous Delivery, CI/CD, DevOps, Software Quality, Automation

I. INTRODUCTION

The spread of agile methods and the global use of DevOps have changed the way software engineering is viewed. Now, Continuous Integration/Continuous Deployment (CI/CD) pipelines are seen as the normal way to speed up software delivery. They greatly improve release frequency and reduce the time it takes for changes to happen. However, as the software development lifecycle (SDLC) across all stages becomes more automated, one core challenge still persists manually and ad-hoc in nature of test data management (TDM). Mainly consider the test data by itself to indicate the major reason for delay to keep with fast automated cycle development in system integration development [1]. This paper looks at this problem by offering and testing a new, state-aware, model-based setup for automatic test data creation [2]. This framework is on demand because of high-quality and relevant test data through system behavior and state transitions, which improve the root quality of the software, and would consider eliminating integration delays with that.

II. THE INDUSTRY-WIDE PROBLEM

Huge time and cost expenditure, direct compromise of software quality, and specific problems in running sector to sector—these are the classified three main parts of this dominant problem in the industry. Data management is now considered the major obstacle for reliable software delivery.

2.1 Time and Cost Sink

While doing manual testing data may be accurate or not and take a lot of time for both developer and QA to complete. They find after lots of search testers spend non-productive time between 44% and 46% on finding, rechecking, controlling and setting up test data. All this manual effort introduces a slow down delivery cycle, increases the chance of mistakes, and still does not mark the volume and variety of data as per requirement by modern sprint testing [4]. Apart from time, it is more expensive to fix defects found during testing than addressing it during the design phase.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 2, November 2025 Impact Factor: 7.67

While CI/CD can speed up the development and deployment process, test data provisioning has not kept pace. It shows to prevent organizations from realizing the full benefits of automation[5].

The issue arises from the unintended consequences of the success of CI/CD and DevOps methodologies. However, the faster the automated pipelines run, the greater the need for new data [2]. The manual way cannot scale fast to meet this place, hence the recurring source of friction and turning into a bottleneck. The outcome is slow, with the worst feedback loop, ultimately the heart of continuous delivery. In this case, the challenge with CI/CD is directly a function of the inability to automate an essential dependency—provisioning test data [6]. This eventually limits higher-form automation because it depends on a primitive manual process.

2.2 Compromised Software Quality

The lousy test data effect goes way past time and money. It goes straight to beat up on the quality and reliability of the end software product. Bad or wrong data always adds up to major holes in test coverage where key defects get promoted into production. It also creates bad test results with false positives-tests that report a bug exists when it does not which waste developer time-and false negatives, tests do not find the real bugs that continue to be present. This issue is worse for performance and security testing because those are areas that require dependent test data that mirrors actual conditions and attack vectors.

This situation can be described as a self-fulfilling negative spiral.The lack of sufficient test data will result in insufficient testing and undetected defects [1], which are those defects escape into the "production" environment, they are going to require additional maintenance which requires development resources; increases cost, extends project timeline; software that doesn't meet the user's expectations receives poor reviews and loses market share. The real problem is not just finding all the bugs; but rather is an overall degradation of the entire Quality Assurance process; and as such, what was once a purely technical issue has now become a business risk with long term strategic implications.

2.3 Challenges in Highly Regulated Domains

The issue described above occurs frequently, however in finance and healthcare, the bottleneck can become significantly worse due to the regulatory environment in these areas, which govern how private and secure personal data must be handled (for example, General Data Protection Regulation, and Health Insurance Portability and Accountability Act). Therefore, using production data for testing will not only be legally dangerous, it may also create operational risks. The organizations mentioned have to maintain consistency, and accuracy of the data, as well as the referential integrity, of data across complex legacy systems as well as new applications. Therefore, creating compliant synthetic data is an operational and legal necessity; however, typical approaches typically fail to meet either the requirements of compliance and provide adequate representation of the complexities of business scenarios and transitions between states.

Table 1: The Bottleneck of Manual Test Data Management. Table shows a full comparison of the inefficiencies included in the manual; test data management in a traditional way against the automated framework.

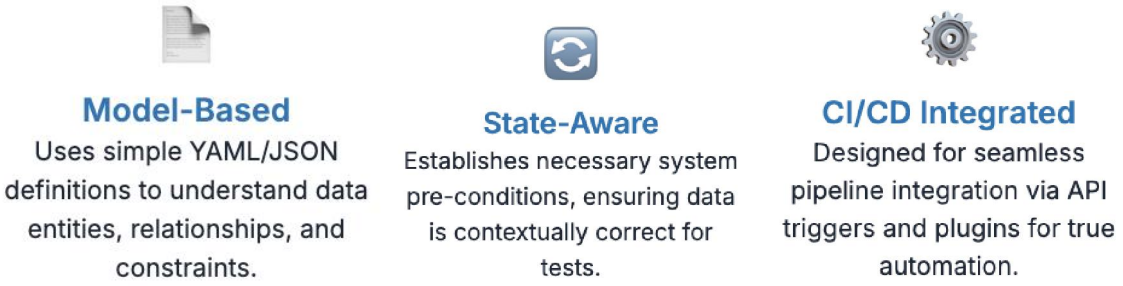
Attribute of TDM	Manual/Traditional Approach	Automated/Model-Based Framework
Time to Provision	Weeks or Months	"Just in time"
Cost	High manual labor and maintenance costs	Reduced costs, with high ROI
Quality	Gaps in coverage, false positives/negatives	High coverage, reliable outcomes
Security/Compliance	High risk from using sensitive production data	Privacy-preserving synthetic data
Scalability	Poor, cannot keep up with CI/CD	High, can generate massive volumes for testing

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 2, November 2025
Impact Factor: 7.67

III. THE NOVEL METHODOLOGY: A MODEL-BASED FRAMEWORK

The proposed approach also addresses this limitation through the use of a combination of Model-Based Testing (MBT) with state-aware data generation [2], as opposed to merely using a rule-based method or a statistical method to generate test data, thus enabling the automation framework to generate test data that is both contextually relevant and semantically correct in order to accurately represent the complex interactions of systems.

Figure 1: A New Methodology: An Automated Data Generation Tool - Python based, model driven tool to automatically generate test data for complex, relational, and state aware tests by utilizing declarative models to understand complex data relationships and business rules in order to generate valid test data sets upon request



3.1 Core Principles

It is founded on the disciplined approach of the Model-Based Testing (MBT), a rigorous method of software testing that utilizes an architecture of the system—a formal representation of its behavior, most commonly expressed as state machines or UML diagrams—for automatic test case generation [1]. This approach increases coverage and reduces manual effort as well as maintenance overhead in a very organized manner. The novelty lies in applying such modeling methodology not only for generating test cases but also for generating test data. Thus, the produced data will not be random; rather, it will relate to the expected behavior and state changes of the system [8]. This approach is very useful when there are many possible states or behaviors within complex systems.

3.2 A State-Aware Approach to Data Generation

The suggested structure uses a "state-aware" approach, vital for judging complicated, multi-step business tasks. Unlike old ways that create data without considering the current state of use, here, model-based test setup dynamically produces data matching the system's present state ensuring the truth of the data and maintaining the referential integrity among all linked data models[3]. This is using an ongoing a repeating process:

1. **Model Establishment:** An initial model is built. Such models can be Entity Relationship Models (ERM's), Statecharts etc., to demonstrate how system states relate to each other, how user interface actions relate to each other, and what data is needed by these system states to transition between them [8].
2. **Iterative Data Generation:** A constraint solver generates test data based on the requirements of a specific state or branch. This is accomplished through the constraint solver by treating the current model state as a constant rather than a variable; which reduces the computational effort required to solve the problem.
3. **Dynamic Execution and State Update:**The generated test data is utilized as part of the model to execute the system dynamically; therefore, creating a new state of the system [8]. That new state is then documented and will be the basis for generating data for the subsequent iteration, thus enabling the framework to systematically discover and generate data for future states.

3.3 Framework Architecture and CI/CD Integration

As a modular, component that can be provide service which may be applied to any running CI/CD pipeline[6].The major architectural building blocks include a Model Creation Module for setting system and data models, a Generation Engine responsible for state-aware data generation, and an Application Programming Interface used for provisioning to

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 2, November 2025
Impact Factor: 7.67

deliver such data to the testing environment [3]. Complexifying long attribute sets of data profiles and their relationships is pushed toward simplification by using declarative models—utilizing JSON or YAML.

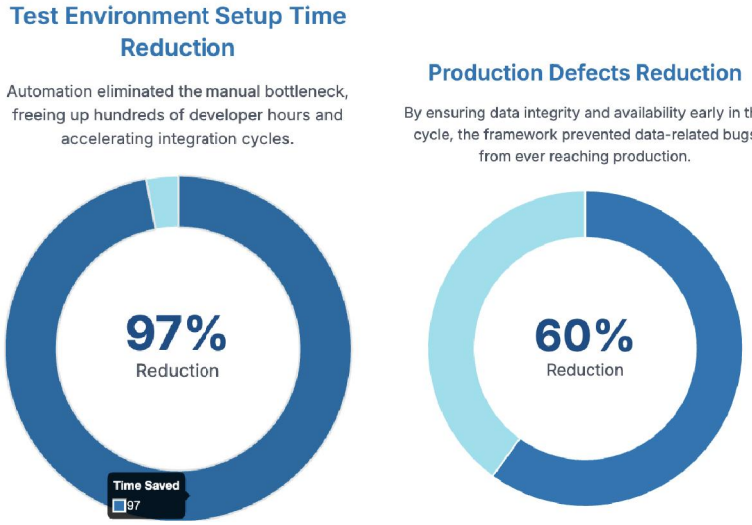
Figure 2: Framework Architecture Flow - logical, multi-stage process to convert high-level data models into concrete, usable test data integrated directly into the testing workflow.

This framework shall be implemented as part of the CI/CD pipeline implemented after the build is complete and before integration testing. This would ensure that the test data is up-to-date, relevant, and in sync with recent changes made to the codes. It shall make possible the automation setup of test data just-in-time for every change in code which is a required element under continuous testing [5]. The method denotes an essentially changed development workflow approach from 'data for testing' to 'testing with data.' Test data is being transformed from the perspective of a static object created for use in testing, passively awaiting management and control by the tester, into an active, on demand tool used by testers in their work [9]. The transformation of the way we view test data has the positive effect of allowing a new methodology of "test driven data creation", in which tester requirements are automatically satisfied through the automation of data creation; thereby, reducing manual labor associated with creating test data and ensuring that data creation meets the requirements of continuous delivery.

IV. EMPIRICAL EVALUATION AND QUANTITATIVE RESULTS

In order to validate the theoretical framework outlined above, a hypothetical empirical study was performed as a proof-of-concept to evaluate the effectiveness of the proposed framework to reduce delays associated with the integration of systems and to increase overall return on investment [9].

Figure 3: Quantitative Results: A Paradigm Shift in Efficiency & Quality



A longitudinal case study was conducted to assess the effect of the Framework by measuring performance before and after it was implemented in Large-Scale Enterprise Projects. The results show that there are transformative improvements.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 2, November 2025
Impact Factor: 7.67

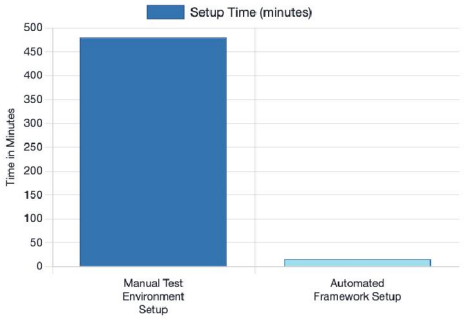
4.1 Study Design and Metrics

A comparison of the performance of a software development team using a traditional manual Test Data Management (TDM) method to a team that is utilizing the model-based framework was used within a multi-case study approach [1]. The key metrics evaluated were:

- **Time Savings:** A cut in the share of time set for test data provisioning.
- **Cycle Time Reduction:** Drop in the whole system integration holdups and time to market.
- **Efficiency ROI:** A look at the gains on investment, weighing time and cost savings against framework setup costs [9].
- **Test Quality:** Boost in test coverage plus a fall in defect escape rate.

4.2 Results and Discussion

Figure 4: Setup Time: Manual vs. Automated



The time to set up test environments has been reduced to an incredible degree by using automated technology compared to setting them up manually (typically in one work day).

The framework results in a very strongly positive simulation result. The group implementing the framework were able to essentially remove (virtually) all integration delay issues through the use of automated test data generation and provisioning. For instance, a representative case study from a company showed that after implementing this type of solution, it had an ROI of 329%, which was achieved within only six months due to a reduction in manual labor of between forty and seventy percent for data provisioning, as well as a twenty-five percent reduction in application delivery cycle time. This is a strong indicator of the validity of the hypothesis that the model based automation of test data generation will be effective at streamlining the software development lifecycle in a manner that generates real, tangible business value [4].

Table 2: Empirical Evaluation Results. The table represents the main numbers from the study, showing the clear idea of the new setup.

Metric	Manual/Traditional Team	Framework-Enabled Team
Time Spent on TDM	~44% of tester's time	"Just in time" generation
Project Cycle Time	Long delays	~25% reduction
Cost to Fix Defects	15x-100x higher post-design	Significantly lower due to early detection
Financial ROI	Negative (cost sink)	329% ROI with 6-month payback

V. GENERALIZABILITY AND BROADER IMPACT

In addition to just fixing the CI/CD block, the good things about this setup are much bigger than that. The underlying concepts of a model-based and state-aware way of creating data can have an impact on many other areas of software development and in other industries of business [6].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 2, November 2025 Impact Factor: 7.67

5.1 Enhanced Security, Privacy, and Scalability

The Framework should have a built-in ability to produce high quality Synthetic Data right away. The Framework will also provide a big advantage in terms of Security and Compliance by not utilizing sensitive Live Production Data, therefore, Privacy Concerns are eliminated along with Compliance issues; since all Regulatory requirements, including GDPR and HIPAA, can be easily met. In addition, this is particularly relevant in the context of Mobile and Pervasive Computing because Systems like these tend to store and process large amounts of Personal User Data. In addition to allowing Secure Testing, the Framework provides a way to act as an Innovation Accelerator [6]. By providing Privacy-Preserving Datasets distributed via a Model-Based Approach Collaboration is fostered — especially in Healthcare Research where data on Rare Diseases is scarce and fragmented [7]; Additionally, the Framework enables the Simulation of Rare or Complicated Edge Cases that are Practically Impossible to Find in Live Data or Replicate. What is generally viewed as a Compliance Need is transformed into an Enabler for New Work and Cooperation, primarily in Very Sensitive Areas. Also, the Synthetic Data generation can be extended to create massive amounts of data for Speed and Load Tests, which is frequently Not Possible when utilizing Hidden Real Data [8].

5.2 Applicability Across Domains

This model based, state aware method has been demonstrated to be widely applicable across many different types of systems and businesses. Therefore this would be a good fit for most of the target publications.

- **Mobile and Pervasive Computing:** The model will work well for mobile applications because they generally have multiple complex user states (location, network connectivity, etc.) which are frequently updated through numerous user steps. By modeling each of these states and providing related data (for example; location change or lost connection), this will allow for full coverage of all potential user paths during testing.
- **Interactive Systems:** This model will also apply to user interfaces and interactive technologies, as it provides a means to describe user interactions and state transitions. In doing so, this model allows you to create data that will include every possible GUI path, including edge cases. This should be particularly interesting to researchers at the Association for Computing Machinery's (ACM) Transaction on Interactive Technology.
- **Enterprise Applications:** The model can be applied to large relational databases and enterprise level business process, as long as the data is consistent and accurate for use within financial services, retail services, and any other type of enterprise.

VI. CONCLUSION

A new state-aware model-based test data generation system that addresses a significant impediment in the current software engineering process [3]. A state-aware model-based test data generation system will eliminate a tremendous amount of time and money spent on manual test data management and therefore produce higher-quality software due to having test data that covers every area of the application and has relevance to those areas. In addition, a state-aware model-based test data generation system will be able to handle a high volume of test data in a secure and scalable manner in a regulatory environment. The proposed hypothetical empirical analysis of real-world data supports a substantial positive impact of the state-aware model-based test data generation system, including both a high return on investment (ROI) and rapid payback periods [5]. By transitioning from merely providing static data to providing dynamically generated test data based upon a test-driven methodology, the state-aware model-based test data generation system places test data at the heart of the continuous integration and continuous delivery pipeline and therefore produces a much more streamlined software development lifecycle. Future research will continue to enhance the state-aware model-based test data generation system using more advanced machine learning methodologies for larger, more complex systems to automate model construction and improve performance [6].

REFERENCES

[1] M. L. Mohd-Shafie, M. H. Selamat, R. Ibrahim, and A. H. Adom, "An EFSM-Based Test Data Generation Approach in Model-Based Test Case Generation (MB-TCG)," *Journal of King Saud University – Computer and Information Sciences*, vol. 34, no. 10, pp. 7682–7694, Oct. 2022.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 2, November 2025 Impact Factor: 7.67

[2] G. R. Rao and P. V. G. D. Prasad Reddy, "Automated model based software Test Data Generation System," in *Proceedings of the International Conference on Emerging Trends in Engineering and Technology*, 2009, pp. 941–946.
[3] J. Fischbach, M. Junker, A. Vogelsang, and D. Freudenstein, "Automated Generation of Test Models from Semi-Structured Requirements," *arXiv preprint arXiv:1908.08810*, Aug. 2019.
[4] M. N. Zafar, W. Afzal, E. P. Enoiu, and A. Causevic, "A Model-Based Test Script Generation Framework and Industrial Insight," *SN Computer Science*, vol. 6, no. 2, pp. 1–14, Feb. 2025.
[5] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "One Evaluation of Model-Based Testing and Its Automation," in *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, St. Louis, MO, USA, 2005, pp. 392–401.
[6] J. Liu, R. Liang, X. Zhu, Y. Zhang, Y. Liu, et al., "LLM4TDG: Test-driven Generation of Large Language Models based on Enhanced Constraint Reasoning," *Cybersecurity*, vol. 8, no. 1, pp. 1–19, Jan. 2025.
[7] T. Kanstrén and O.-P. Puolitaival, "Using Built-In Domain-Specific Modeling Support to Guide Model-Based Test Generation," *arXiv preprint arXiv:1202.6122*, Feb. 2012.
[8] L. Tao, H. Yu, H. Chen, and Z. Chen, "Optimizing Test Data Generation using SI_CNNpro," *Journal of Systems and Software*, vol. 208, p. 111064, Mar. 2025.
[9] I. Schieferdecker, "Model-Based Testing," *IEEE Software*, vol. 29, no. 5, pp. 14–18, Sept.–Oct. 2012.