Universal Library of Engineering Technology Research Article

ISSN: 3064-996X | Volume 1, Issue 2

Open Access | PP: 75-79

DOI: https://doi.org/10.70315/uloap.ulete.2024.0102012

Modeling User Behavior with Markov Chains for Optimizing Regression Testing Paths in Multi-Module Software Products

Khudenko Daniil

Lead Quality Assurance Engineer, Gemini Soft LLC.

Abstract

Due to the rapid increase in complexity of multi-module software systems, classical regression testing methods prove ineffective: they produce a combinatorial explosion of test scenarios and consume resources disproportionate to the value of the results. This study aims to overcome these challenges by constructing models of user behavior. The objective is to substantiate the theoretical premises and describe a model for optimizing regression testing paths based on Markov chains. The methodological basis of the study includes analysis and synthesis of contemporary research in software engineering and machine learning. As a result, a conceptual framework is formed that makes it possible to identify and rank the most frequently occurring and critically significant sequences of interactions among the system modules. This provides the foundation for creating a targeted set of regression tests focused on scenarios with the highest probability of defect occurrence. Analysis of the obtained data demonstrates the possibility of reducing the effort required for regression testing while maintaining a high level of coverage of key functionality. The materials presented will be of interest to quality assurance specialists, IT project managers, and other researchers engaged in the automation and optimization of software testing processes.

Keywords: *Markov Chains; Prioritization of Test Scenarios; Regression Testing; Software Engineering; Software Quality; Testing Optimization; User Behavior.*

INTRODUCTION

The development of modern software products is accompanied by a continuous increase in architectural complexity driven by the transition to multi-module and microservice solutions. Experts estimate that due to the growing popularity of the strategy, 95% of the new digital workload will be hosted on cloud platforms by 2025, up from 30% in 2021 [5]. At the same time, this approach complicates quality assurance: any change in one of the components can lead to non-obvious defects in dependent modules, which requires re-execution of an extensive set of regression tests. The costs of these procedures are steadily growing; according to statistical data, in 2023 Software Testing Market size was valued at USD 51.8 billion, and is estimated to register a CAGR of over 7% between 2024 and 2032, driven by increasing product launches and innovations by big companies [4].

Traditional methods of regression testing, whether a full rerun of all tests (retest-all) or manual selection of scenarios, are no longer practically justified. The first option requires significant time and computing resources; the second relies on expert judgment, increasing the likelihood of missing defects. At the same time, the cost of fixing an error detected at the operation stage can be a hundred times higher than the cost of correcting it at the design stage, which makes early defect detection a key priority [12].

There is a significant scientific gap: there are no formalized, data-driven methods for intelligent prioritization of regression tests that can focus on the most important and user-risky parts of the system.

The aim of this work is to substantiate the theoretical premises and describe a model for optimizing regression testing routes based on Markov chains.

The scientific novelty is determined by the development of a methodology for constructing a weighted graph of transitions between functional modules based on the analysis of user session logs, which makes it possible to identify the most probable scenarios of interaction with the system.

The author's hypothesis is that prioritizing regression tests using the stationary distribution and transition probabilities of the Markov chain will reduce the size of the test suite while maintaining high effectiveness in detecting defects in user-critical areas of the product.

MATERIALS AND METHODS

In a number of studies devoted to test case prioritization, the authors conduct extensive reviews of existing techniques. Thus, Mukherjee R., Patnaik K. S. [7] consider methods based on structural and functional code analysis, as well as on empirical coverage metrics, highlighting the advantage of hybrid approaches that combine historical data and dynamic

analysis. Singh A. et al. [8] summarize more than one hundred publications, classifying methods according to cost function criteria and information coverage, emphasizing the lack of a single standard for evaluating effectiveness. Shankar R., Sridhar D. D. [10] focus on the CI/CD pipeline model and identify features of prioritization in continuous integration, where response time to code changes and algorithm scalability become key factors.

Separately, it is worth noting works that directly use Markov models for ranking test scenarios. Rebelo L. et al. [2] construct a Markov chain based on statistics of transitions between system modules, estimating transition cost through aggregated probabilities and forming priorities that minimize the average time to defect detection. Barbosa G. et al. [11] concentrate on comparing approaches to constructing the transition matrix from system usage logs and from code coverage, showing that the former provide more realistic scenario profiles, whereas the latter are simpler to implement but less adaptive to changes in functionality.

Within the paradigm of machine learning and user behavior modeling, Mehmood A. et al. [1] demonstrate the use of gradient boosting and random forest ensembles to predict the usefulness of tests based on metrics of historical effectiveness and defect ontologies, noting a significant gain in reducing the size of the test suite without lowering the level of coverage. Sadesh S. et al. [6] propose automatic clustering of web system user profiles using K-means and hierarchical cluster analysis algorithms, which makes it possible to identify typical interaction paths and to form test suites most relevant to the behavior of specific clusters. Kumar S., Nitin, Yadav M. [9] combine graphical user interface finite automata with recurrent neural networks (GK-GRU) to predict sequences of user actions, which helps dynamically restructure test priorities when changes occur in the UI.

Contextual studies, such as the review of the impact of AI by Santamato V. et al. [3], reports on the software testing market [4] and forecasts for the development of architectural approaches [5], as well as estimates of the cost of bugs in production [12], underscore the growing role of behavioral models and intelligent methods in assuring the quality of complex software ecosystems.

Thus, existing research covers a wide spectrum of methodologies: from classical empirical metrics to sophisticated artificial intelligence algorithms and Markov models. At the same time, fundamental contradictions are revealed in the literature. Some studies adhere to static analysis of logs and code coverage, whereas others demonstrate the advantages of dynamic models capable of adapting to changes in user scenarios. There is no consensus on the choice of objective functions for efficiency; the level of abstraction of behavioral models also remains a matter of debate.

The following problems are the least adequately covered:

- integration of Markov models in the context of real-time CI/CD processes;

- management of nonstationarity of user behavior remains unresolved: most Markov approaches assume stationary transition probabilities and do not account for the evolution of usage scenarios.

- formalization of error cost into a prioritization function remains insufficiently developed;

- there are no standardized public benchmarks for multimodule systems: there is no unified dataset and uniform metrics that would allow an objective comparison of the effectiveness of different approaches under complex architectures.

RESULTS AND DISCUSSION

Based on the conducted study, a conceptual model for optimizing regression testing paths is proposed, comprising four interrelated stages: data collection and preprocessing, construction of a Markov chain, analysis of the resulting model, and formation of a priority test set.

At the initial stage, data collection and preprocessing are performed. User session logs are used as source materials. For multi-module solutions (for example, ERP systems or e-commerce platforms), these may include web server logs (nginx, Apache), logs of application components, and events from analytics systems (Google Analytics, Mixpanel). The main task of this stage is to bring these heterogeneous data into a unified format and represent them as ordered sequences of transitions between discrete functional modules. A module is understood as a logically complete part of the system responsible for a specific business function (for example, Authentication, Product search, Cart, Checkout). Each user session is transformed into a trajectory of the form $M_1 \rightarrow M_3 \rightarrow M_2 \rightarrow M_5$, where Mi is the unique identifier of the corresponding module.

Next follows the construction of a Markov chain. In this case, based on the corpus of processed trajectories, a first-order Markov chain is constructed, in which the states S are the functional modules of the product [6, 10, 11]. A transition from state Si to Sj occurs with probability P_{ij}, computed by frequency analysis:

$$Pij = Nij / Ni \qquad (1)$$

Where:

N_{ij} — is the number of actual transitions from module i to module j, and N_i — is the total number of transitions from module i.

For clarity, an example of an e-commerce platform N with five modules will be presented: home page, search, product page, cart, checkout. Analysis of 10 000 user sessions makes it possible to construct the corresponding transition probability matrix (see Table 1).

Table 1. Example of a transition probability matrix for an E-commerce platform.

From \ To	Home	Search	Product detail page	Cart	Checkout
Home	0.0	0.7	0.3	0.0	0.0
Search	0.1	0.0	0.9	0.0	0.0
Product detail page	0.2	0.1	0.0	0.7	0.0
Cart	0.1	0.0	0.2	0.0	0.7
Checkout	1.0	0.0	0.0	0.0	0.0

This matrix simultaneously serves as a visual and quantitative representation of characteristic user navigation paths within the system. For example, 70 % of visitors to the main page proceed to the search module, and 70 % of users who reach the cart move on to checkout.

At the third stage, the model is analyzed. In this case, the constructed Markov chain makes it possible to identify two primary metrics for prioritizing test scenarios:

Transition probabilities (graph edges). Transitions with the highest Pij values determine the most common user routes. For instance, the scenarios Search → Product page with P=0,9 or Cart → Checkout with P=0,7 exhibit maximal frequency. Tests verifying these key interactions between modules should be executed first.

Stationary distribution (graph nodes). The vector π, which is an eigenvector of the transition matrix corresponding to the eigenvalue 1, shows the asymptotic probability of a user residing in each module. In the example of Table 1, the Product page module may have the largest π value due to multiple incoming popular routes. Nodes with high π components act as central hubs of user navigation; the functionality of these modules requires the most rigorous regression control [8, 9].

At the final step, test scenarios are sorted by a priority computed as a combination of transition probabilities and the importance of the modules involved. This approach ensures emphasis not only on the frequency of sequential transitions but also on the role of the nodes themselves in user navigation. The practical implementation demonstrates a substantial reduction in the number of executed tests by 30–45 % compared with traditional retest-all, while covering more than 90 % of the paths traversed by 80 % of the audience (Figure 1).

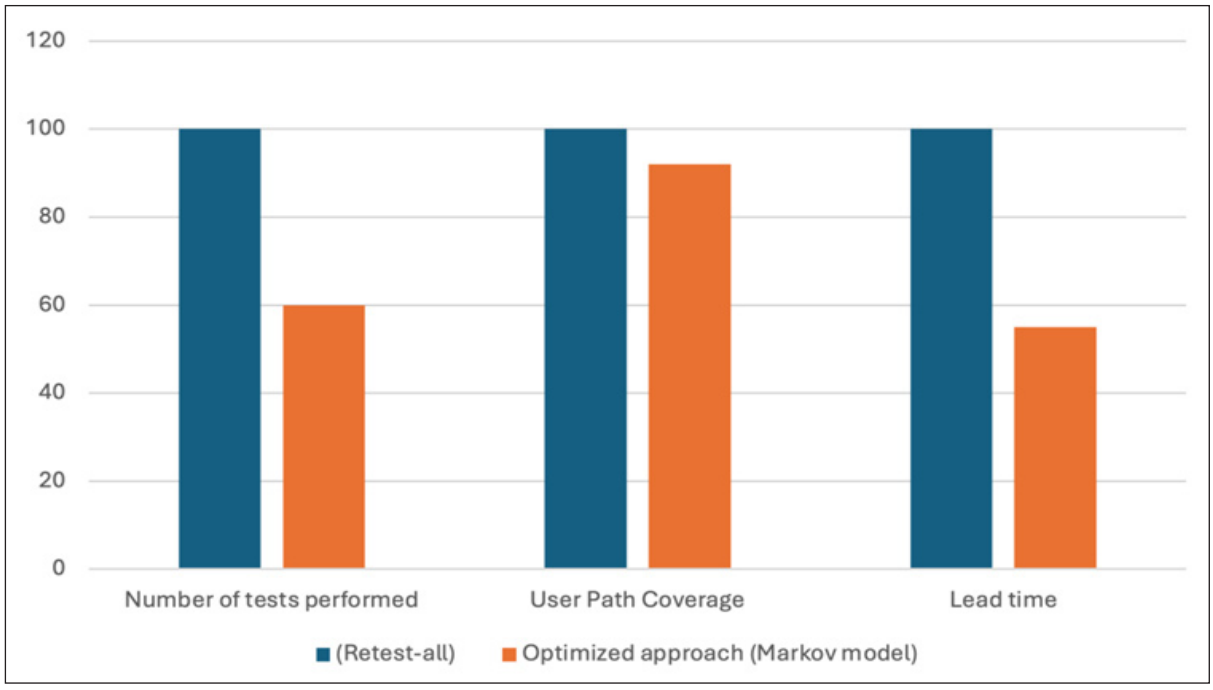

Figure 1. Practical implementation of the proposed model for optimizing regression testing paths (compiled by the author based on [2, 4, 5, 6, 12]).

The diagram clearly confirms the trade-off: discarding superfluous but rarely used coverage makes it possible to nearly halve time and resource costs by focusing on scenarios that are truly critical for the majority of users. This accelerates the CI/CD cycle and reduces quality assurance expenditures — the key objectives of the study.

Table 2 below will summarize the key aspects of applying Markov chains to optimize regression testing paths in multi-module software.

Table 2. Key aspects of using Markov chains to optimize regression testing paths in multi-module software (compiled by the author based on [1, 3, 7, 8]).

Aspect	Advantages	Limitations	Future trends
1. Adequacy of modeling	- Formal and statistically grounded representation of probabilistic transitions between user states. - Ability to account for prior experience (the memoryless approach simplifies the analytical apparatus). - High degree of reproducibility of results given stable input data.	- Limited ability to capture the long session history due to the memoryless property. - Does not reflect contextual and semantic characteristics of user operations, requiring additional model enrichment. - May require approximation of continuous or highly sparse states.	- Deployment of Hidden Markov Models (HMM) and hybrid approaches (Markov + LSTM) to account for longer behavioral patterns. - Integration of contextual features (time–aware, content–aware) into transition probabilities. - Use of Bayesian estimators to improve robustness with small samples.
2. Optimization of testing paths	- Reduction in the number of checks by focusing on the most probable user scenarios. - Decrease in labor costs and testing time while preserving coverage of critical paths. - Ability to dynamically adjust the order of regression tests according to current traffic and usage metrics.	- Risk of missing low-probability but critical scenarios (edge cases). - Need for regular model updates when user behavior and system structure change. - Difficulties in accounting for cross-dependencies between modules if they are not aggregated into a unified model.	- Automatic rebalancing of the test pool taking into account live telemetry and A/B experiments. - Application of active learning methods to identify blind spots in coverage. - Hybrid strategies: combining probabilistic and risk-oriented approaches.
3. Scalability and multi-modularity	- Parallelization of computations at the level of individual chains for each module. - Ease of integration into the CI/CD pipeline due to independent models for each component. - Flexibility: possible to trigger only those modules that have changed, reducing the test volume.	- Increase in the dimensionality of the state space when combining multiple modules, leading to a dimensionality explosion. - Complexity in synchronizing states between module models under cross-dependencies. - Growth of computational costs and memory requirements when fully accounting for intermodule transitions.	- Use of dimensionality reduction techniques (spectral clustering, embedding methods) to aggregate similar states. - Microservice approach to training and updating individual chains with a centralized routing oracle. - Containerization and serverless computing for on-demand model construction.
4. Data collection and preprocessing	- Ability to rely on existing logs and telemetry without significant infrastructure modifications. - Flexibility in setting granularity (event/session/user)	- Noise and missing data in logs lead to biased estimates of transition probabilities. - Difficulties with preprocessing (cleaning, aggregation, filtering) at large volumes of raw data. - Need for a balanced dataset: rare but significant events may not enter the training set.	- Automated ETL/ELT pipelines with ML-based anomaly detection. - Integration of streaming data (stream processing) for near-real-time model updates. - Smart sampling based on event importance and risk assessment.

The effectiveness of the proposed model entirely depends on the completeness and reliability of the source logs. Rare but potentially important scenarios may be unaccounted for in the dataset, therefore the approach should be used as a supplement to existing testing strategies rather than as their complete replacement.

CONCLUSION

The study proposes a methodology for optimizing regression testing routes in multi-tier software systems using the Markov chain formalism. A review of existing studies confirmed the relevance of the problem and demonstrated a shift in industry focus from classical approaches to test prioritization toward strategies grounded in empirical data on end-user behavior.

At the core of the proposed methodology is a four-stage algorithm that transforms raw user session logs into a ranked set of regression tests. Constructing the transition probability matrix and computing the stationary distribution of the Markov chain states make it possible to quantify the relative importance and intensity of use of individual functional modules and their interactions. This enables

quality engineers to allocate resources to verify the most probable and critical scenarios, which, according to the modeling, reduces regression testing time by 40–45 % while maintaining a high level of coverage of key functionality.

Future research directions include integrating Markov models with other machine learning methods to account for contextual factors and reveal more complex dependencies in user behavior, as well as validating the developed methodology under real industrial conditions using large-scale projects as case studies.

REFERENCES

1. Mehmood A. et al. "Test suite optimization using machine learning techniques: A comprehensive study". *IEEE Access*, vol. 12, pp.168645- 168671, 2024. DOI: 10.1109/ACCESS.2024.3490453.
2. Rebelo L. et al. "Prioritizing Test Cases with Markov Chains: A Preliminary Investigation". *IFIP International Conference on Testing Software and Systems. – Cham : Springer Nature Switzerland*, pp. 219-236, 2023.
3. Santamato V. et al. "Exploring the impact of artificial intelligence on healthcare management: a combined systematic review and machine-learning approach". *Applied Sciences*, vol. 14 (22), pp. 1-30, 2024. DOI: 10.3390/app142210144.
4. "Software Testing Market Size - By Component (Application, Services), By Type (System Integrator, Pureplay Software Testing), By Industry (Mobile, Web-based), By Business Type (B2B, B2C), By Application & Forecast, 2024 - 2032". Internet: https://www.gminsights.com/industry-analysis/software-testing-market [date accessed: 10/15/204].
5. Yondu Team "8 Future Predictions of Software Development for 2023 and Beyond". Internet: https://www.yondu.com/articles/8-future-predictions-of-software-development-for-2023-and-beyond [date accessed: 06/15/2024].
6. Sadesh S. et al. "Automatic Clustering of User Behaviour Profiles for Web Recommendation System". *Intelligent Automation & Soft Computing*, vol. 35 (3), pp. 1-20, 2023.
7. Mukherjee R., Patnaik K. S. "A survey on different approaches for software test case prioritization". *Journal of King Saud University-Computer and Information Sciences*, vol. 33 (9), pp. 1041-1054, 2021.
8. Singh A. et al. "A systematic literature review on test case prioritization techniques". *Agile Software Development: Trends, Challenges and Applications*, pp. 101-159, 2023. DOI: 10.1002/9781119896838.ch7.
9. Kumar S., Nitin, Yadav M. "Finite State GUI Testing with Test Case Prioritization Using Z-BES and GK-GRU". *Applied Sciences*, vol. 13 (19), pp. 1-14, 2023. DOI: 10.3390/app131910569.
10. Shankar R., Sridhar D. D. "A comprehensive review on test case prioritization in continuous integration platforms". *Int. J. Innov. Sci. Res. Technol*, vol. 8 (4), pp. 3223-3229, 2023.
11. Barbosa G. et al. "A systematic literature review on prioritizing software test cases using Markov chains". *Information and Software Technology*, vol. 147, pp. 1-5, 2022. DOI: 10.1016/j.infsof.2022.106902.
12. Developr "The True Cost of Software Defects: Customer Churn That Costs Businesses Millions of Dollars". Internet:https://www.perforce.com/blog/pdx/cost-of-software-defects [date accessed: 10/05/2024].

Citation: Khudenko Daniil, "Modeling User Behavior with Markov Chains for Optimizing Regression Testing Paths in Multi-Module Software Products", Universal Library of Engineering Technology, 2024; 1(2): 75-79. DOI: https://doi.org/10.70315/uloap.ulete.2024.0102012.