Journal Homepage: - www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI: 10.21474/IJAR01/23330
DOI URL: http://dx.doi.org/10.21474/IJAR01/23330

CONFERENCE PAPER

STATE MODEL TESTING WITH MULTI COVERAGE CRITERIA ANALYSIS

Sonali Pradhan and Adya Anindita Mishra

Manuscript Info

Manuscript History
Received: 8 February 2026
Final Accepted: 10 March 2026
Published: April 2026

Key words:-
Coverage Criteria, State Coverage, Transition Coverage

Abstract

State-based testing is a powerful technique in software testing that focuses on validating the behavior of a system based on its states and transitions. This approach is particularly effective for systems that exhibit different responses depending on their internal states and some workflow-driven applications. This paper explores state-based testing using various coverage criteria, including state coverage, transition coverage, transition pair coverage, and path coverage. By applying these criteria, testers can systematically derive test cases that ensure comprehensive examination of system behavior. Each criterion provides a different level of thoroughness, from verifying that all states are reachable to ensuring all possible state sequences are tested. This paper highlights the importance of selecting appropriate coverage criteria based on testing goals to achieve optimal state based fault detection. The study underscores how state-based testing not only improves defect detection but also enhances the clarity and structure of the testing process.

Introduction:-

State-based testing is a black-box testing technique used to validate the behavior of a system by modeling it as a finite state machine (FSM) [1, 3]. In many software systems, the output and behavior depend not only on the current input but also on the history of inputs, which determines the current state. State-based testing captures this dynamic behavior by representing the system in terms of states, transitions, events, and actions. In a state machine, each state represents a particular condition or situation of the system at a given time. Transitions are occurring based on events or inputs, causing the system to change from one state to another. State-based testing ensures that the system correctly handles all defined transitions and that it behaves as expected in each possible state.

This technique is especially useful for systems like protocol handlers, embedded systems, reactive systems, and user interfaces, where different sequences of inputs can lead to different behaviors. Test cases are derived from the state transition diagram or state table, and they are evaluated using various coverage criteria [2, 3, 4] such as state coverage, transition coverage, transition pair coverage, and path coverage.

Corresponding Author:- Sonali Pradhan

State-based testing helps in uncovering issues like:

- Incorrect transitions between states,
- Missing or undefined states or transitions,
- Invalid state behavior,
- Redundant or unreachable states.

By providing a structured way to test complex systems with multiple states and transitions, state- based testing improves test thoroughness and enhances state based faults detection, and contributes to the overall reliability and quality of the software system.

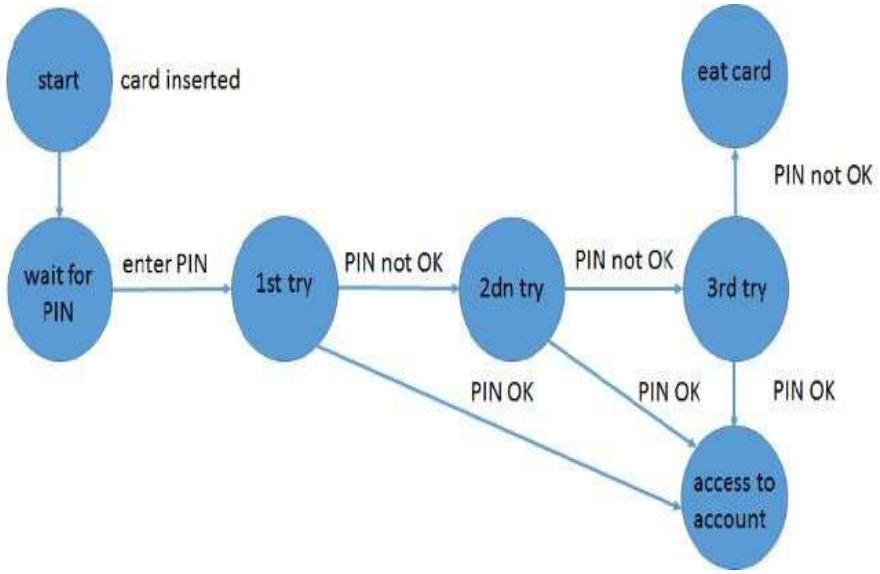

Fig1: State transition of ATM Card Transaction

In fig. 1 the ATM goes through a number of transitional states, depending on the inputs of the user. By entering the right PIN, the user will end up at the "Access Granted" state. If they enter the wrong PIN, they will pass to another state whereby it will give a second chance. If they enter the wrong PIN again, it will reach the third chance state. If they then enter the PIN incorrectly again, another state arises, typically one that indicates that their account has been blocked.The remainder of this paper is divided into four sections. Section 2 contains related work. Section 3 describes Coverage Criteria and test case generation, Section 4 describes Result discussion and finally Section 5 shows the conclusion and future scope.

Background and Related Studies:

State-based testing has been widely studied and applied as a technique for validating systems with dynamic behavior. It builds on the concept of finite state machines (FSMs), which has long been recognized as a powerful way to model system behavior, especially for reactive and event- driven systems. Chow (1978) was among the earliest researchers to formalize test case generation from finite state machines, introducing the W-method for testing minimal state machines [1]. This work laid the foundation for later techniques in test generation and coverage analysis.Several researchers have extended state-based testing by defining different coverage criteria, such as state coverage, transition coverage, transition pair coverage, and path coverage. Binder (1999) elaborated on these techniques in the context of object-oriented software, demonstrating their utility in uncovering state-dependent defects [2]. Offutt and Abdurazik (1999) explored automated test generation from UML state charts, highlighting the scalability of model-based approaches to real-world applications [3]. Their research emphasized how integrating formal models with automated tools can increase testing efficiency and reduce human error. In recent years, model-based testing has seen a surge in interest, with tools and methodologies developed to extract and test systems using state models. Utting and Legeard (2007) provided a comprehensive overview of model-based testing techniques, including state-based strategies and their application in industrial settings [4, 7].

Furthermore, research by Hierons et al. (2009) examined test selection strategies and optimization in FSM-based testing, which is essential for handling large and complex systems where exhaustive path testing is not feasible [5, 6, 7, 8]. Advancements have also been made in automated test case generation from state models using graph traversal algorithms and AI-based methods, enabling more effective coverage of the state space [9, 10]. These efforts continue to enhance the practicality and robustness of state-based testing in modern software development environments.

TEST CASE GENERATION USING COVERAGE CRITERIA:

Based on the state transition diagram the ATM Card Transaction system is done. We derive test cases according to different coverage criteria in state-based testing:

We find the following states in State Coverage Criteria. The states are as follows: Start, Wait for PIN, 1st Try, 2nd Try, 3rd Try, Access to Account and Eat Card

TC (Test Case) for State Coverage:

TC1:Start→Wait for PIN→ 1st Try (PINOK) →Access to Account

TC2:Start→Wait for PIN→1st Try (PIN not OK) →2nd Try (PIN not OK) →3rd Try (PIN not OK) → Eat Card.

In Transition Coverage, we have following transitions:

- Start→ Wait for PIN
- Wait for PIN→1st Try
- 1st Try→ 2nd Try (PIN not OK)
- 2nd Try→ 3rd Try (PIN not OK)
- 3rd Try→ Eat Card (PIN not OK)
- 1st Try Access to Account (PINOK)
- 2nd Try Access to Account (PINOK)\
- 3rd Try Access to Account (PINOK) Test Cases for Transition Coverage:

TC1:Start→Wait for PIN→ 1st Try (PIN OK) →Access to Account

TC2:Start→Wait for PIN→1st Try (PIN not OK)→2nd Try (PIN OK)→Access to Account TC3: Start → Wait for PIN → 1st Try (PIN not OK) → 2nd Try (PIN not OK) → 3rd Try (PIN OK) → Access to Account

TC4:Start→Wait for PIN→1st Try (PIN not OK)→2nd Try (PIN not OK)→3rd Try (PIN not OK) → Eat Card

Test Cases for Transition Pair Coverage:

TC1:

Start→Wait for PIN→1st Try (PINOK)→Access to Account TC2:

Start→Wait for PIN→1st Try (PIN not OK)→ 2nd Try (PIN OK)→Access to Account TC3:

Start→Wait for PIN→ 1st Try (PIN not OK)→ 2nd Try (PIN not OK)→ 3rd Try (PIN OK)

→Access to Account

TC4:

Start→Wait for PIN→1st Try (PIN not OK)→ 2nd Try (PIN not OK)→3rd Try (PIN not OK) → Eat Card

These test cases ensure that each pair of transitions is covered.

Test cases for Path Coverage Possible

Paths in the diagram are:

Successful Login on First Try:

Start→Wait for PIN→ 1st Try (PIN OK)→ Accessto Account

Successful Login on Second Try:

Start→Wait for PIN→1st Try (PIN not OK)→ 2ndTry (PIN OK)→Accessto Account

Successful Login on Third Try:

Start→Wait for PIN→1st Try (PIN not OK)→ 2nd Try (PIN not OK)→3rd Try (PIN OK) → Access to Account

Card Eaten after Three Failed Attempts:

Start→Wait for PIN→1st Try (PIN not OK)→ 2nd Try (PIN not OK)→3rd Try (PIN not OK) → Eat Card

RESULT DISCUSSION:

We find the four number of test cases covering four criteria which is shown in Table-1.

I. TABLE-1

TestCase	Scenario Description	CriteriaCovered
TC1	Correct PIN on 1st try	All
TC2	Incorrect 1st, correct 2nd	All
TC3	Incorrect 1st and 2nd, correct 3rd	All
TC4	All 3 attempts fail –card-eaten	All

Conclusion and Future work:

State-based testing provides a structured and effective approach to validating systems whose behavior depends on their internal states and transitions. By modeling the system as a finite state machine, testers can systematically analyze different scenarios, ensuring that all possible states and transitions are exercised. In the case study ATM Card Transaction, test cases were designed to fulfill various coverage criteria including state coverage, transition coverage, transition pair coverage, and path coverage. This ensures thorough validation of system behavior under different input sequences and improves the detection of faults such as incorrect transitions, unreachable states, or missed scenarios. In future we will explore on automation tools that can automatically generate test cases from state diagrams and evaluate coverage metrics to reduce manual effort and human error. We also work on integration with Model-Based testing that combine with other UML diagrams to derive test cases from formal specifications or system models.

References:-

1. Binder, R. V. (1999). *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley.
2. Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3), 178–187.
3. Hierons, R. M., et al. (2009). Using formal methods to support testing. *ACM Computing Surveys*, 41(2), 9.
4. Kolchyn, O., & Potiyenko, S. (2024, April). Model-Based Test Cases Generation for Extended Data Flow Coverage Criteria. In*Computer Science On-line Conference*(pp. 568-581). Cham: Springer Nature Switzerland.
5. Offutt, J., & Abdurazik, A. (1999). Generating tests from UML specifications. In *Proceedings of the 2nd International Conference on the Unified Modeling Language* (pp. 416– 429). Springer.
6. Su, Z., Yu, Z., Wang, D., Chang, W., Gu, B., & Jiang, Y. (2024, October). Test Case Generation for Simulink Models using Model Fuzzing and State Solving. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (pp. 117-128).
7. Utting, M., & Legeard, B. (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann.
8. Zafar, M.N., Afzal,W.,Enoiu,E.P.,Haider,Z.,& Singh,I.(2025).A Model-BasedTest Script Generation Framework and Industrial Insight. *SN Computer Science*, *6*(4), 1-19.
9. Pradhan, S., Ray, M., Jena, M., Tripathy, M., & Panda, I. (2025). Adopting state transition testing techniques in scenario-based test case generation. *International Journal of System Assurance Engineering and Management*, 1-14.
10. Pradhan, S., Ray, M., & Swain, S. K. (2022). Transition coverage based test case generation from state chart diagram. *Journal of King Saud University-Computer and Information Sciences*, *34*(3), 993-1002.