UtilitasMathematica

ISSN 0315-3681 Volume 122, 2025

Comprehensive Software Test Strategies for Subscription-Based Applications and Payment Systems

Savi Grover
Independent Researcher, New Jersey, USA
savig447@gmail.com

Abstract

Subscription-based digital platforms and payment systems operate at the intersection of reliability, performance, and financial compliance. Failures in these systems can result in financial loss, regulatory repercussions, and reputational damage. Ensuring resilience—the ability to maintain service availability and data consistency under stress or failure conditions—is therefore paramount. This paper presents a comprehensive set of test strategies to validate and improve the resilience of subscription apps and payment platforms. The proposed methodologies span chaos testing, load testing, disaster recovery simulation, and fault injection, all integrated within automation and CI/CD to enable continuous resilience assurance**.**

Keywords

Resilience testing, subscription apps, payment systems, chaos engineering, load testing, idempotency, retry logic, automated testing, transaction integrity.

1. Introduction

The global proliferation of Software-as-a-Service (SaaS), digital media platforms, and fintech solutions has led to widespread adoption of subscription-based billing and automated payment mechanisms. The emergence of a broad range of financial technology (FinTech) applications enables consumers to move beyond the conventional cash-based payment system. Digital payments are becoming the norm in people's daily lives.[1] These rapid developments in the financial sector led to the invention of many digital payment technologies, through which payers and payees both use digital apps to send and receive money. Thus, the payment system is rapidly changing from coin-based and paper-based money to digital forms of payments that are convenient, fast and cost effective [2]. The meaning of subscription billing is – periodic deduction of user's payment based on their active subscription(enrollment) for usage of an app or service (annual or monthly). It can vary from physical services- like booking an online carpet cleaning service subscription offering definite number of sessions per year, or even a gym membership is a monthly subscription. Here the billing structure used is called an automatic payment mechanism which works on charging the customer directly using their card info/e-wallet info.

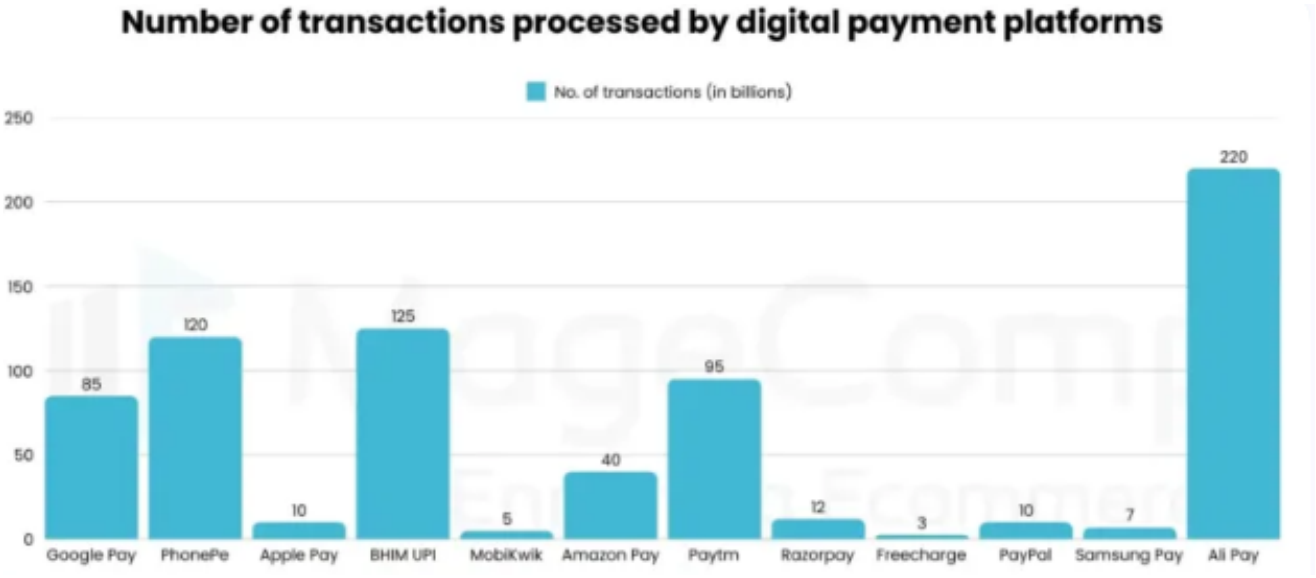


Figure-1- Number of transactions processed by digital payment platforms. Source [17]

Why are digital payments preferred – the ease of implementation, usage- payment by a single swipe or click, no need of cash withdrawal, cash handling, saves time, purchasing a good or service – located at a distance. These are easy to adopt, educate and can be easily effective with people in different demographics. Environment friendly, with no forms, no paper and still reliable. **Figure-1** statically shows the number of transactions recorded (in billions) by digital apps in respective payment apps and integrators used by different countries of the world in the year 2024.

What are User expectations- Users expect uninterrupted access, instant feedback on transactions, and high reliability—even during partial system failures. Service disconnection and connection ease on the fly, log of history transactions, better user interface experience, access to old invoices, profile, posts, requests easily. Privacy, robustness and security since these services directly involve user payment and personal details.

To meet such expectations, platforms must be designed with resilience in mind. According to [3], resilience refers to the ability of a system to withstand and recover from unexpected conditions without service degradation or data loss. Testing such systems thus requires more than conventional functional validation; it necessitates comprehensive test strategies that simulate real-world disruptions.

Study backtracking and research evolution- The evolution of this research starts with – a study on Digital Payment Systems & Consumer Perception [4] finished in **2022** in Journal of Positive School Psychology, focusing on user acceptance and customer perception of different modes of digital payments. Extending this research to discover certain challenges in adoption and awareness of online payments in [1], **Figure 2** mainly categorises challenges

in 5 classes- Technical, Social, Awareness, Economics and Legal. This classification illustrates a major socio-economic barrier in advancement of digital payment sector and was done in a study review by [1]

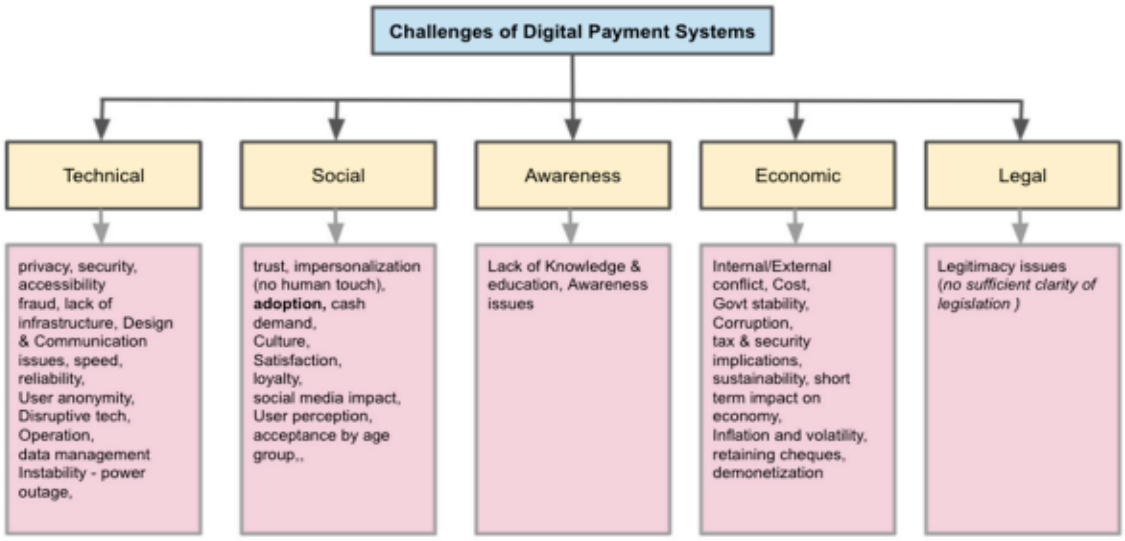


Figure-2- Classification of digital payments technology challenges; Source [1]

Here **Figure-3** illustrates - compared to other challenges, more studies have been covered on technical challenges associated with emerging digital payment technologies. Thus, 32 studies up to the year 2022 included technical issues, such as privacy, security, access, fraud, lack of infrastructure, design and communication issues, speed, reliability, user anonymity, disruptive technology, data management and instability (power outage).

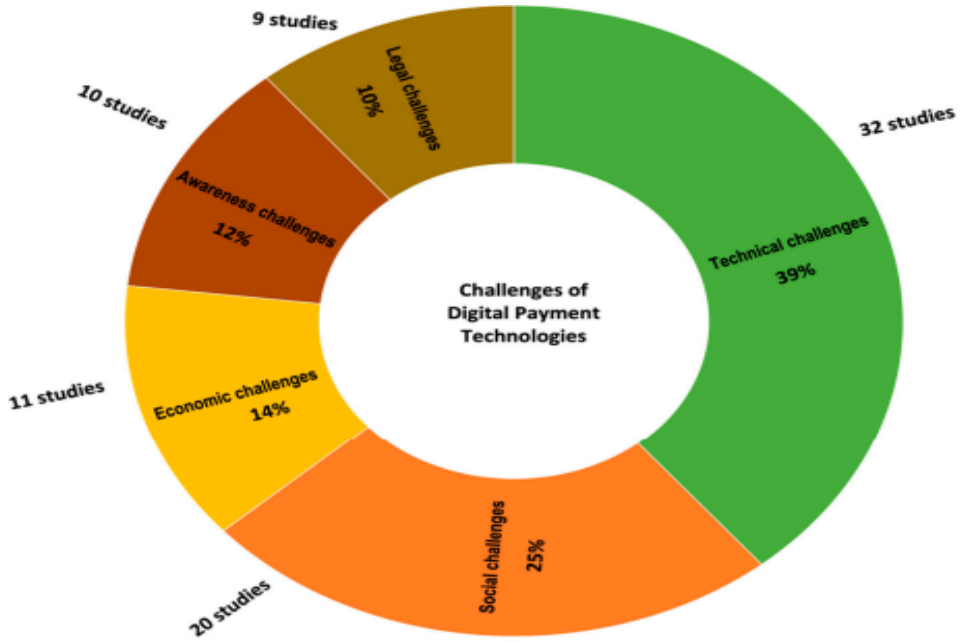


Figure-3- Number of studies done on payment technology challenges in different sectors up to 2022. Source [1]

Overall, the technical challenges relate to 'safety and reliability issues' and 'privacy issues'. The studies highlighted security breaches and lack of protection against fraud and cyberattacks as reasons for reducing consumers' use of digital payment technologies. Similarly, in terms of 'privacy', the interception of confidential information, incomplete and interrupted transactions and frequent service denial faced by consumers lowered their trust and engagement towards digital payment technologies. It was indicated that the scope and magnitude of privacy risk had substantial effects on consumers' confidence and negatively impacted the global economy in general. Furthermore, it was found that although there are other negative impacts imposed by the privacy risks, the significant effect is the loss of customer trust [5].

Digital payment technologies have their own share of technical glitches. The payments do not always run smoothly. For example, UPI-based payment technology in India called BHIM (Bharath Interface for Money) reported having problems when installing and operating [6]. This shows that although consumers are ready to use digital payment technologies, there are challenges regarding infrastructure and availability. Furthermore, network failures negatively impacted system popularity [6]. Similarly, Kaleeth and Chellammal [7] emphasized improving infrastructure facilities in the rural areas of India where poor network connectivity disrupts a transaction to result in incomplete transactions.

Ensuring quality and security in the digital journey in 2025 is the focus of [8], where a new governance regulation is created to monitor inter and intra-banking transactions- ISO 20022 SWIFT. The paper also highlights the importance of collaboration among stakeholders, including financial institutions, software vendors, and testing teams, to ensure a smooth and successful transition to the new standard The paper presented a comprehensive set of testing strategies tailored for the ISO 20022 Swift 2025 transformation, covering test planning, test case design, test data management, test automation, interoperability testing, performance testing, security testing, and defect management.

Present Day Focus- The following paper measures, characterizes, qualifies and outlines on software quality techniques needed for resilient subscription models and optimized payment systems- payment gateways and processes along with establishing user data privacy, fault tolerance application which can sustain network outage, user load, signal loss and gateway unavailability. Creating norms for billing operations, third party integrations and test automation advancement in tech infrastructure – to create a readily available set of sub systems to trigger and initiate collective test enablement performance.

2. Characteristics of Resilient Subscription and Payment Systems

2.1 **Fault Tolerance**-

Duplicate Critical Components: Replicate essential components like servers, databases, and network paths to ensure continuous operation even if one component fails. Ability to continue operation despite component or service failures [9].

Offline capabilities: transactions can be processed online and offline. In an offline transaction, the card and merchant terminal exchange data and decide whether a transaction should be accepted based on issuer-set risk parameters and configuration. This adds critical resilience features that are necessary when network connectivity is unavailable. [12]

Tamper-resistant hardware: EMV implementations provide tamper-resistant hardware for customers (cards), merchants (PoS terminals), and banks (HSMs). Smartcards are offline dedicated hardware devices used to authorize EMV transactions, which are difficult to compromise because of multiple tampering countermeasures, and are only used for payment processing. This trusted hardware guarantees the correctness of the EMV protocol, by protecting the keys used to generate MACs and signatures.[12] GSM and OTP methods of transactions and app activity capture.

Active-Active vs. Active-Passive Failover: Implement failover mechanisms to automatically switch to a redundant component or system in case of failure.

- Active-active failover: Both primary and secondary systems are actively processing requests and sharing the load. If one fails, the other continues to operate seamlessly.
- Active-passive failover: The secondary system remains on standby and becomes active only when the primary fails.

2.2 **Data Consistency-** user profile data updating, transactions reflecting ACID properties, Data Testing with quality test. High-Quality Test Data is a Business Imperative [13**]**

Backend Testing- Adhering to transactional guarantees across distributed components [10]. fast, frictionless, and secure payment- process higher transaction volumes at faster speeds while maintaining flawless accuracy, using serverless SAAS like Microsoft fabric which works on Polaris computation Engine- with advantages like distributed query processing, columnar storage, auto-scaling and failure proof node structure.

Conflict Resolution- Employing strategies like Last-Writer-Wins or vector clocks to resolve inconsistencies when data conflicts arise in eventually consistent systems.

Change Data Capture (CDC)- Capturing and propagating database changes in real-time to keep different systems synchronized. Maintaining a real-time change log and transactional historical synapses.

Regular Testing and Validation- Conducting regular tests of data backups and recovery processes to ensure data resiliency measures are functioning as expected.

2.3 **Network Sustenance & Graceful Degradation-** Maintaining partial service functionality when subsystems fail. Payment gateways utilize a dedicated network for clearing transactions. This network is not built on top of the Internet and is less vulnerable to Distributed-Denial-of-Service (DDoS) attacks; this provides resilience for network connections.

Contingency Planning and Alternative Payment Methods- Having a comprehensive contingency plan that includes alternative payment methods (e.g., other processors, cash options) can help mitigate the impact of an outage.

Investing in Robust Network Connectivity- Ensuring reliable and high-speed internet connections, especially for online transactions, is essential, according to tnsi.com.

Collaboration for Traffic Management- Banks and payment processors should collaborate to manage increased traffic effectively, including regular load testing and failover mechanisms, says tnsi.com.

Idempotency- Design operations to be idempotent, meaning performing the same action multiple times produces the same result as doing it once. This is critical for preventing duplicate charges or unwanted side effects if a transaction fails and is retried.

Retry Mechanisms- Implement retry mechanisms with exponential backoff to handle transient errors and avoid overwhelming the system.

Graceful Degradation- Ensure the system maintains partial functionality even in the face of failures, for example, by temporarily disabling non-essential features rather than crashing completely.

2.4 **Billing Operational Excellence-** retry logic, idempotency, verification, auditability- Preventing duplicate charges or inconsistent states during retries. Trial periods, Refunds, returns, cancellation excellence along with delinquents' avoidance.

Multiple Payment Options- Offering a variety of payment methods, including online payments, mobile payments, and direct debit, can improve customer satisfaction and convenience.

Early Payment Discounts- Strategic use of early payment discounts can incentivize customers to pay promptly, improving cash flow and potentially strengthening customer relationships.

Clear and Concise Invoices- Providing clear and concise invoices that are easy to understand, and the process can reduce customer confusion and disputes.

Customer Support- Offering excellent customer support for billing inquiries can improve customer satisfaction and resolve issues quickly.

Data Security- Protecting sensitive customer data throughout the billing process is crucial, especially with the increasing prevalence of online payments.

2.5 **Observability, Error detection and recovery-** Visibility into system health through logs, metrics, and traces [11].

Circuit Breaker Pattern- Protect services from cascading failures by detecting non-responsive services and preventing further calls to them. The circuit breaker "opens" when a service starts failing, returning an immediate error instead of waiting for a timeout.

Monitoring and incident management-

Real-time Monitoring & Alerting- Continuously monitor system performance, logs, and metrics to quickly identify and respond to issues before they escalate.

Structured Logging- Implement structured logging to store logs in a centralized, searchable format, facilitating easier debugging and troubleshooting, especially in distributed systems.

Incident Response Protocols- Establish a clear incident response plan defining roles, responsibilities, and communication channels for effective and efficient recovery during a disaster.

2.6 **Load Handling, Scalable, Self-Healing AI-**

Microservices: Break down the system into smaller, independent services to isolate potential failures and allow for independent scaling and deployment. If one microservice fails, the entire application isn't brought down.

Distributed Databases: Use distributed database systems (like Apache Cassandra or Amazon DynamoDB) to ensure data availability even in case of hardware failures.

Load Balancing: Distribute network traffic across multiple servers to prevent overload and support fault tolerance by rerouting traffic if a server fails.

Personalized Payment Experiences- AI leverages customer data like transaction history, browsing behavior, and preferences to offer tailored payment options, personalized recommendations, and customized loyalty programs, enhancing customer satisfaction and loyalty.

Automated Payment Processing and Reconciliation- AI streamlines tasks like invoice matching, reconciliation, and data entry, reducing manual effort, improving accuracy, and accelerating transaction clearing and settlement.

Dynamic Pricing and Revenue Management- AI algorithms can analyze market data and customer behavior to optimize pricing strategies in real time, adjust prices based on demand and other factors, and personalize offers to maximize revenue and profitability.

2.7 **Cloud Maintainability, Efficiency and Scalable Solution** - Legacy testing environments are quickly becoming obsolete. Web-enabled and cloud-based testing solutions are enabling financial institutions to conduct secure, scalable, and cost-effective testing across global markets.[13]

Dynamic resource allocation- Cloud platforms can automatically scale resources (compute power, storage, etc.) up or down based on demand, allowing businesses to handle fluctuating transaction volumes and seasonal spikes seamlessly

Adaptability to growth- As businesses expand, cloud solutions can accommodate increased transaction volumes without requiring major infrastructure overhauls or significant upfront investments.

Global reach- Cloud platforms support multi-currency and multi-channel payments, facilitating business expansion into new markets and catering to a diverse customer base. Cost-effectiveness- Scalability in the cloud often translates to cost savings, as businesses pay only for the resources they utilize (pay-as-you-go model), rather than investing in excess capacity for potential future growth.

2.8 **Governance and Compliance-** Artificial intelligence is poised to reshape the payment testing landscape. While we are still in the early stages of AI-driven testing, its potential for enhancing efficiency and accuracy is significant. From predictive analytics that detect vulnerabilities before they occur to automated test case generation that speeds up development cycles, AI can transform payment testing from a reactive process into a proactive, intelligent function.[13]

Risk Assessment- Identify and assess potential threats and vulnerabilities to the payment system, considering both internal and external factors like cyberattacks, hardware failures, and human error.

Security Controls- Implement robust security controls and policies to protect the payment system from unauthorized access, modification, or disclosure. Complying with PCI DSS helps reduce the risk of cyberattacks, data breaches, and associated costs while boosting customer confidence in your brand. It supports your overall risk management strategies and aligns with other compliance frameworks like NIST and GDPR. Examples include encryption, authentication, firewalls, and regular security audits.[14]

Disaster Recovery Planning (DRP)- Develop a comprehensive DRP outlining the goals, resources, procedures, and responsibilities for restoring payment systems after a disruptive event. This plan should define recovery point objectives (RPO) and recovery time objectives (RTO).

3. Testing Methodologies for Resilience in Payment Systems

3.1. Chaos Testing

Chaos testing introduces deliberate failures into the environment to evaluate system responses. Tools such as Gremlin, Chaos Monkey, and LitmusChaos help simulate outages in critical services like payment gateways or subscription engines [15]. To add intelligent errors/chaos into the system -like wrong payment card, expired card, foreign card, invalid cvc, extra digit in card number, past expiration dates, current date, negative balance cards, gift cards, prepaid cards, nonexistent card, and enhance payment resilience in these diverse test scenarios. Simulating third-party API outages by analyzing how the payment application handles a disruption in a payment gateway's API or a data feed from a financial service. Introducing network latency or failures and observing how the application responds to slower network connections or temporary connection losses during transactions. Simulating database failures by assessing the application's ability to recover from a database outage without data loss or transaction errors. Testing authentication and authorization failures and attempting failed logins or unauthorized access attempts to uncover vulnerabilities in security mechanisms.

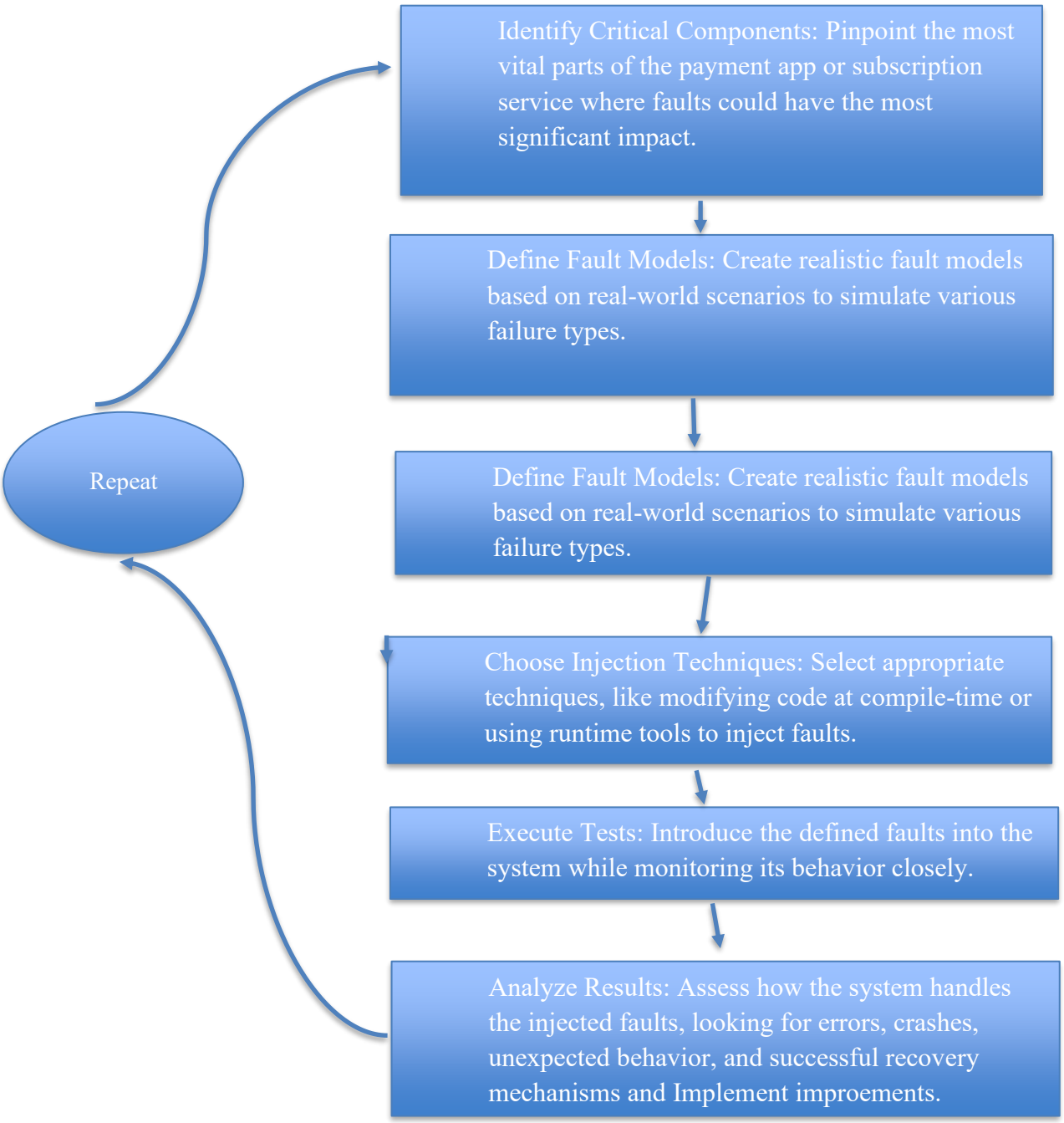
3.2. Fault Injection Testing

By injecting network latency, service timeouts, or database lock conditions, testers can assess how gracefully systems recover from degraded performance. Fault injections are a specific technique where controlled failures are introduced to test a system's response to those failures, while chaos engineering is a broader methodology that uses controlled experiments to discover unknown weaknesses and improve overall system resilience. Fault Injection Testing (FIT) is a crucial technique for enhancing the robustness and resilience of software, especially for critical applications like payment apps and subscription software. These applications deal with sensitive data and financial transactions, where system failures or vulnerabilities can lead to significant financial loss and reputational damage.

There are many benefits to payment apps by using FIT such as- Identifies Vulnerabilities deliberately by introducing errors or faults to uncover potential weaknesses and vulnerabilities in the system's design or code that could be exploited by malicious actors. Ensuring Robust Error Handling and Recovery and verify the system's ability to gracefully handle unexpected errors, recover from failures, and continue operating reliably under adverse conditions. FIT helps validate and strengthen fault tolerance mechanisms such as redundancy, failover systems, and error recovery procedures, ensuring that the system can maintain functionality even in the face of unexpected events. Reduces Downtime and Costs by proactively addressing potential issues before deployment, FIT minimizes the risk of costly

system failures and associated downtime in production environments. Ensures Data Integrity and Consistency in financial transactions are handled correctly and securely, even during system failures, ensuring eventual consistency and preventing erroneous charges or missed payments.

Steps for performing FIT- the below are ordered steps to exercise fault injection test



3.3. Load and Spike Testing

Load and spike testing are crucial aspects of performance testing for applications that handle financial transactions and manage user subscriptions. In the fintech industry, where real-time transactions and high availability are paramount, these testing methodologies ensure the application remains stable and responsive under varying user loads. These tests validate system performance under high loads such as- Billing cycle peaks (e.g., 1st of the month) or Flash sales triggering thousands of simultaneous subscriptions. Tools like Apache JMeter and k6 provide controlled stress environments [16].

3.3.1. Load testing

Load testing simulates the expected user traffic and transaction volume that the application will encounter during normal and peak usage periods. This helps identify performance bottlenecks, gauge the system's capacity, and ensure it can handle anticipated workloads without significant degradation or failure. Simulates concurrent transactions, payment processing, and interactions with third-party payment gateways. Verifies the efficiency of load balancers, database performance, and overall system infrastructure in handling large transaction volumes. Assesses the throughput rates necessary to sustain peak load scenarios.

Load testing in subscription software- include Tests the performance of subscription management processes, including user registration, subscription upgrades/downgrades, and cancellation flows. Checking for concurrency issues when multiple users try to manage their subscriptions simultaneously. Evaluates the system's ability to handle large volumes of subscription data and ensures data consistency and accuracy.

3.3.2. Spike testing

Spike testing is a specific type of performance testing that evaluates the application's ability to handle sudden, extreme increases and decreases in user load. It simulates unexpected traffic surges, like those caused by a marketing campaign or a flash sale and assesses how the system responds and recovers from these sudden bursts of activity.

Spike testing in payment applications: Simulates sudden increases in payment requests during peak sales periods, such as Black Friday or Cyber Mondays, to ensure the system can handle the load without downtime or transaction failures. Tests the ability of payment networks to handle rapid surges in transactions without technical outages or connectivity issues. Evaluates the system's response to situations where multiple users attempt to complete transactions concurrently, potentially leading to race conditions or data inconsistencies.

Spike testing in subscription software: Simulates sudden surges in user sign-ups or subscription changes to see if the system can scale effectively and manage the increased load. Tests the system's ability to handle rapid increases in requests for subscription modifications or cancellations without errors or delays. Assesses the impact of sudden load changes on the accuracy and consistency of subscription data.

3.4. Transaction Consistency Testing

Subscription apps testing often includes the following checks for each multi-step transaction.

- **Payment authorization** → capture → confirm card→subscription update. Failures at any step require rollback mechanisms to preserve consistency.
- **Valid Card checks-** card number valid, expired card, lost card, gift prepaid cards, debit, credit, multicurrency cards check to undergo payment card testing, changing card and valid UX designs.
- **Trial duration:** When testing on the trial length, factors such as the learning curve and discovery time of the app should be considered. By experimenting with various trial lengths, you can identify which duration retains and converts users most effectively, ultimately leading to increased revenue generation.
- **Subscription durations**: Testing different subscription durations, such as monthly, quarterly, and annual options, can help identify the most attractive and profitable choices for customers. Quality measures to ensure customers are granted full perks underneath selected tier and able to upgrade/downgrade tiers easily.
- **Pricing Testing-** Multi-currency and Tax Variations, Localization accuracy, invoiced detailed line items, payment integrators – PayPal, Venmo, Zelle.
- **Fines, late fee, extra charges, delinquents testing-** User scenarios like using extra services, fine charges, late fee and missing payments should be a highly focused testcase. Not only late fee and missing payments of one tier but if customer transitioned to another tier, price calculations logic must be evaluated along with rollover bills and new charges.
- **Returns, cancelation, trial period cancelations-** subscription cancellation or return of subscribed products, retrieval of product, trial period cancellations and tier cancellations, in and out of grace period, efficient deactivation and post deactivation activities.
- **Retry and Idempotency Validation-** Missed payments retry payment/re-attempt charging card, if failed all attempts- addition of late fee. Loss of network or device battery while payment attempt, provision of authentication of OTP if payment wants to be done via call/message. Payment’s gateway requires tests/sandbox environment**-** where Subscription Expiry During Downtime, prevent access

disruption, implement grace periods and retry logic, mock failure and verify fallback workflows can be tested.

- **Rewards, Promo codes, discounts, refunds, partial refunds-** Promo Code Abuse, customer rewards, referral code, coupon code, refunds credits, guest referral and bonus credit procedure testing.
- **Double Charging, Fraud detection and disaster recovery testing-** missing charges, double charges testing, fraud detection, blocking fraud transaction, invoice integrity – simulation of data testing and fraud mitigation.

3.5 Usability testing-

Usability- Successful usability testing takes time and expertise to plan, recruit participants, manage flows, analyze data, and turn the results into action, to manage the end-to-end usability study as well as a network of participants that match the target users. Customer experience, pain points and delivery of data-driven innovation without the hassle.

UI testing-User interface testing (UI functionality, specifically. UI and UX testing are another 3,000-word guide) primarily seeks to validate the function of two types of user interactions: inputs and visual elements. Can your web app accurately read a mouse click? Does a user keyboard integrate into your app and perform as expected? Are buttons/links/submission forms functional?

Localization testing- Localization is the process of ensuring an application looks and feels right to the target user. Localization testing verifies an application's functionality and usability in a specific region. It checks native UI, language, currency, date & time formatting to meet the standards of that country. It must use language correctly and align with cultural norms. [14]

Efficient Signup and Deactivation experience – Users should have a clear path of contacting customer service, navigation of apps, cancelling membership, opting for additional product, taxing, invoices, and cancellation and deactivation path.

Responsive Testing- Device compatibility tests ensuring access to applications on different apps and being able to access important parts of application successfully in all of their devices. In case of B2B apps, ensuring permits and role-based subscription version and responsive test across different authorization.

User privacy- scenarios like recovery emails, OTP, MFA (multifactor authentication), credit card masking, Marketing email subscription and discontinuation, account numbers hiding, joint account capabilities and account data safety should be a part of maintaining user data privacy.

3.6 Functional Component Testing

Functional testing validates that your payments software is executing as expected, ensuring a quality product. Functional QA testing services help you identify and prioritize web and mobile app issues before your users do.

Integration Testing This is the phase of software testing where software modules are combined, requiring group testing. Integration testing evaluates the compliance of software or apps with specific functional requirements. Integration tests segment components or modules and verify the functionality of each one individually and as part of the group.

4. Automation Testing Measures for resilient payment and subscription apps

4.1. User journey and UI/UX testing

Test coverage for critical user flows: Focus on automating tests for crucial user journeys, such as new user registration, subscription creation, plan upgrades/downgrades, choosing trial plan, payment failures, and cancellation flows. Verify the visibility and functionality of all UI elements across different devices, screen sizes, and operating system versions, ensuring consistent user experience. Automated UI tests help catch issues arising from dynamic elements or updates. Onboarding and first-time user experience: Automate tests to ensure smooth onboarding, focusing on ease of use, clear instructions, and intuitive interaction with key features. This testing also is called Happy Path Testing.

4.2. Billing and subscription management testing

Recurring billing and payment gateway integration: Automate tests to ensure the accurate processing of recurring payments, including various payment methods, billing frequencies, and retry mechanisms for failed payments. Subscription plan management- Verify that users can seamlessly upgrade, downgrade, or change their subscription plans, and that the associated billing changes are reflected correctly. Dunning management and churn prevention: Automate tests for dunning processes, such as sending reminders for failed payments and offering alternative payment methods, to reduce revenue leakage and improve subscriber retention.

API testing and contract testing while integrating with third party payment services with using tools like POSTMAN and swagger. Introducing edge cases, parameterizations in CRUD endpoints and detecting errors prior to UI testing. These testing methods include microservices system testing – like payments table, card tables, billing transaction table, customer, product, plan and subscription back-end

tables. Also including chaos-testing and fault injections in source code as a part of developer unit testing can also result in early detection of defects. Service Virtualization by creating sandboxed payment processor mocks for sandbox environments.

Exception handling, detailed logs, event tracking, notifications, test reports and error log analysis leads to root cause analysis of mitigating these defects.

4.3. Operational, Managerial and Performance and stability testing

- Load and stress testing: Simulate peak usage scenarios and high user traffic to assess the app's performance under stress, measuring metrics like response time, throughput, and resource utilization.
- Build stability: Track the frequency of successful vs. failed builds to ensure that code changes don't introduce defects that break the build, impacting the continuous integration and delivery pipeline.
- Test execution time: Monitor and optimize the total time taken to execute the automated test suite, allowing for faster feedback loops and quicker iterations in the development process.
- Measure Flaky tests- test cases indicating failure intermittently should be noticed- and troubleshooting with an allocated categorization to tech issue, functionality issue or data issue.
- Managerial tool for analysis of user activities, login attempts, clicks, churn rates, drop off pages, ads click should ideally be maintained and tracked by administrators, customer care, business analysts and test team to ensure customer sync and data quality, rate of efficiency.

4.4. Test quality and efficiency metrics

- Test automation coverage: Measure the percentage of application code covered by automated tests, ensuring thorough validation of critical functionalities and identifying gaps in the testing strategy.
- Defect density and defect leakage: Monitor the number of defects found per unit of software size (defect density) and the number of defects that escape to later stages (defect leakage), indicating the effectiveness of the test suite in catching issues early.
- Automation return on investment (ROI): Quantify the financial benefits of test automation by comparing cost savings from reduced manual effort and fewer defects with the investment in automation.
- Test maintenance effort: Track the resources spent on updating and maintaining test scripts to gauge the long-term sustainability of the automation effort.
- Choosing the right tools: Evaluate various tools like Selenium, Appium, Keploy, Cypress, and Playwright based on your specific needs

and budget, considering factors like platform support, language flexibility, and integration capabilities.

- Handling dynamic elements: Use robust locator strategies (e.g., XPath, CSS selectors) and explicit wait conditions to manage dynamic UI elements effectively.
- Maintaining test data and environments: Implement automated data management techniques (e.g., data seeding, snapshots) and ensure consistent testing environments to avoid data-related issues.
- Integrating with CI/CD: Ensure automated tests run quickly and reliably within your continuous integration and delivery pipelines to enable faster deployments and feedback cycles.
- Integrating performance tools like JMeter and automation code optimization tools like GitHub co-pilot.

5. Conclusion

Resilience testing is vital for maintaining trust in subscription-based and payment-driven applications. As failures are inevitable, robust testing strategies that simulate and validate failure recovery, ensure transactional consistency, and incorporate automated observability are crucial. Integrating these practices early in the development lifecycle provides a competitive edge in reliability and user satisfaction.

References

1. Review The Emerging Technologies of Digital Payments and Associated Challenges: A Systematic Literature Review Khando Khando, M. Sirajul Islam and Shang Gao - Published: 30 December 2022
2. Premchand, A.; Choudhry, A. Future of Payments—ePayments. Int. J. Emerg. Technol. Adv. Eng. 2015, 5, 110–115.
3. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*, O'Reilly Media, 2016.
4. A Study on Digital Payments System & Consumer Perception: An Empirical Survey. Shinki Katyayani Pandey Assistant Professor, Kalinga University, Naya Raipur, CG shinki.pandey@kalingauniversity.ac.in
5. Akanfe, O.; Valecha, R.; Rao, H.R. Assessing country-level privacy risk for digital payment systems. Comput. Secur. 2020, 99, 102065. [CrossRef
6. Seethamraju, R.; Diatha, K.S. Adoption of Digital Payments by Small Retail Stores; UTS ePress: Ultimo, Australia, 2018
7. Kaleeth, A.B.L.; Chellammal, T. Adoption of Digital Payment Methods in Rural Areas of Ramanathapuram District. Ann. Rom. Soc. Cell Biol. 2021, 25, 7831–7837.

8. ENSURING QUALITY IN THE MODERNIZATION JOURNEY: SOFTWARE TESTING STRATEGIES FOR PAYMENT ISO 20022 SWIFT – 2025 TRANSFORMATION Praveen Kumar- Journal of Software Quality Assurance (JSQA)
9. N. Dragoni, S. Dustdar, S. Larsen, and M. Mazzara, "Microservices: Migration of a Mission Critical System," *IEEE Software*, vol. 35, no. 3, 2018.
10. E. Brewer, "CAP Twelve Years Later: How the 'Rules' Have Changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.
11. M. Sigelman et al., "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," *Google Research Publication*, 2010.
12. Resilient payment systems, Khaled Baqer, University of Cambridge 2018
13. https://www.paragonedge.com/blog/2025-trends-in-payment-testing-what-you-need-to-know
14. https://www.vikingcloud.com/blog/pci-dss-compliance-guide#1
15. Netflix Tech Blog. "The Netflix Simian Army," [Online]. Available: https://netflixtechblog.com.
16. A. Cockcroft, "Chaos Engineering," *InfoQ Software Architecture Guide*, 2020.
17. https://magecomp.com/blog/digital-payment-statistics/