Availability: Smart testing strategies for 24/7 operational systems

Summary

This document was prompted in part by the evidence from the Treasury Select Committee's report[1] that nearly half of the outages over a period of two years in the major nine banks were due to introducing system changes[2]. It suggests service-related and technical indicators to assess testing approaches to reduce the risks associated with change for existing 24/7 operational systems. This builds on the experience of BCS members from across financial services and from testing experts[3] and is intended to provide a framework for developing testing strategies for testing changes to 24/7 operational systems. But of course - even 24/7 operations have scheduled down time – testing scheduled then may be the best way of reducing the impact of outages on users.

Testing in this environment has a different focus from that for new systems development, that of Non-Functional-Requirements (NFR[4]). For instance testing for unanticipated data or transaction volumes, via stress or scenario testing, moves up the aganda.

An Appendix defines some of the terminology of testing.

For a broader consideration of testing, and the frameworks associated with testing, one standard reference is the DevOps Handbook[5], while the Ministry of Testing[6] web site is a rich source of information.

---

[1] https://committees.parliament.uk/committee/158/treasury-committee/news/205611/more-than-one-months-worth-of-it-failures-at-major-banks-and-building-societies-in-the-last-two-years/

[2] https://www.bcs.org/media/rxdmjr5h/availability-6-report-nine-banks-data-roundtable-220425.pdf

[3] In this text and others in the series of ITLF Availability Papers, "we" refers to the Availability/Service Resilience Working Group of the IT Leaders Forum of the BCS – the Chartered Institute for IT, colleagues from our network and from BCS's SIGIST who have provided additional insights.

[4] Non-functional requirements are the criteria that define how a system should behave, rather than what it is supposed to do. https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples

[5] The DevOps Handbook Continuous Delivery

[6] https://www.ministryoftesting.com/

The challenges of testing changes to existing Operational Systems

First, we capture some of the characteristics of existing operational systems that present a challenge to recovery and business continuity for complex 24/7 operational systems, and hence affect testing strategies.

- Complexity of Real-World Environments: Operational systems must be tested under conditions that accurately mimic real-world scenarios, including diverse user behaviors, variable network traffic, hardware differences, and geographic dispersion.
- Managing Test Data and Environments: Test environments must closely mirror production to provide reliable results, but setting up, maintaining, and updating these environments is resource-intensive. Ensuring diverse, up-to-date, and privacy-compliant test data adds to the complexity. In addition, test data management must now extend (as the focus on AI dependability, ethics and bias increases) into ensuring that the test data is validated in the same way as operational data, and does not create biases or ethical challenges within AI enabled systems.
- Interdependency and Integration: Modern systems rely on third-party services, APIs, and integrations. Identifying whether failures originate from external (3rd party) systems or within the core product can require new forensic analysis skills, and testing components built using different architectures or standards can cause unexpected disruption.
- Non-Deterministic Behaviour: Systems with concurrent operations, real-time interactions, or machine learning components may exhibit unpredictable behaviour. Testing must account for and detect failures that are intermittent or hard to reproduce, which may require extensive automation and expert validation and is a potential role for AI[7].
- Collaboration and Communication: Testing often involves multiple stakeholders (developers, testers, product owners), who may have different goals or interpretations of requirements. Lack of clear, consistent communication can limit test effectiveness and coverage.
- Managing changes to existing datasets or data models, and managing the integrity of existing data and long running processes: testing changes to existing systems which hold critical data, or indeed processes in which data is accumulated over long periods of time will require explicit consideration of the management of the data.

[7] See ITLF Papers – "Use of AI to improve service resilience"

- Service Level Agreements may impact the ability to test changes, or may impact the length of time within which to make the change. The challenges of either zero, or very short, unavailability within which to make a release will lead to the adoption of techniques such as canary releases, blue-green testing and alpha/beta releases.

Further, it is worth reiterating that all testing is constrained by time and cost, and that therefore the need is to be able to assess the risk presented by changes to existing systems, and prioritise the testing to minimise the risk to availability of the systems to users. Critical analysis continues to be a core part of creating a good test strategy, and the engagement of a range of perspectives can identify potential hurdles and vulnerabilities based on past experience.

Defining metrics as part of the testing framework is important to ensure that you have clarity on the results of testing: for instance, is the Availability increasing? Relevant metrics include

- Uptime/Downtime: Measures the percentage of time a system is operational and available for use;
- Test Case Effectiveness: Measures how well test cases detect defects;
- Change Failure Rate: Measures the percentage of deployments that result in failures.

Key factors influencing testing strategy for existing operational systems.

With the characteristics of these systems in mind, we can define a number of criteria which shape the design, focus and execution of testing for changes to existing operational systems.

These headings provide a framework for discussion on what is important to any particular organisation, or system, and will enable a focus on those aspects of the solution which are likely to be highest risk of causing service outages.

Service related criteria

1. Is it a Critical Business Process?
Failures here can have severe financial, reputational, or regulatory impacts.

2. Is it a Customer-Facing Service?

User experience, usability, accessibility, and cross-platform consistency are key for all users, but particularly for customers.

3. Are there stringent performance targets?
Testing will typically need to focus on load, stress, and scalability testing, both in terms of meeting those targets, but also focussing on the implications on existing timeouts and execution paths.

4. Are there stringent RPO (Recovery Point Objective) targets?
Typically, these requirements determine acceptable data loss in a disaster and any changes associated with back up frequency, data replication, data restoration, data integrity and housekeeping.

5. Are there stringent RTO (Recovery Time Objective) targets?
RTO defines the maximum acceptable downtime, and changes in processes can impact response times.

6. Does the system have high scalability requirements?
Varying loads, including peak and sustained scenarios for processes or services which have changed in order, may need testing to identify throughput bottlenecks.

7. Does the system have high availability requirements?
*If the s*ystem must meet uptime/service level agreements then the testing strategy should consider availability, failover, and redundancy testing. It must simulate outages and validate recovery mechanisms.

Technology/implementation related criteria

8. What percentage of the application is batch processing?
Systems which are batch-heavy systems, and where there are changes to the batch programmes suggest testing of validation of scheduled jobs, data integrity, and error handling and recovery from failures.

9. What percentage of the application being changed is legacy, which may mean that tech is unsupported, very complex/tangled or undocumented?
Legacy systems may have had significant change, over time, or the integrations may be highly complex. In these cases, there may be a requirement for increased regression and integration testing.

10. With high level of componentisation (modularity) of the operational system, how many components are impacted by the change?
Highly compartmentalized systems may isolate failures but increase integration complexity, leading to increased requirements for end-to-end testing.

11. What is the change frequency and stability?
If changes are very rarely made to a large and complex system, then the organisational memory of managing those changes could be low, and the reliance on subject matter experts correspondingly high.

12. What level of test automation is present?
High levels of automation enable rapid, repeatable testing; low automation increases manual effort and risk. Investing in automation for testing may be very cost effective for systems subject to large amounts of change.

13. How much have the data structures and data changed?
Frequent or large data changes increase risk of defects and data integrity issues.

14. How large is the supply chain impacted by the change?
Deep supply chains increase complexity and risk of upstream/downstream failures.

15. What is the Implementation or deployment Process?
If the process of deployment involves a 'Big Bang' or a large cutover, which could be visible and high risk, then the testing requirements will be different to a more phased implementation with the ability to deploy as a pilot or beta test. In a cutover situation, depending on the size of the change, then regression, cutover, and rollback testing may be required.

Not every organisation, or indeed every change, requires all the different testing approaches. The table below links the indicators that we have outlined above with the testing approaches that they might influence. So, for example, load, stress, scalability, and resource utilization testing are indicated for changes to an existing application with stringent performance and throughput requirements. And if this same application has a deep supply chain, then end-to-end, integration and disruption simulation testing are indicated.

We are conscious that the range of tools marketed for testing is vast and growing: this is not the subject of this paper, however we recommend that SIGIST considers developing or promoting such a guide.

Key service indicators	Testing Approaches Influenced
Critical Business Process	Risk-based, end-to-end, and resilience testing
Customer-Facing Service	Usability, accessibility, real-user, and compatibility testing
NFR: Performance	Load, stress, scalability, and resource utilization testing
NFR: RPO	Backup, data replication, and restoration testing
NFR: RTO	Disaster recovery, failover, and rapid restoration testing
NFR: Throughput	Throughput and volume testing under peak and sustained loads
NFR: Availability	Uptime, failover, redundancy, and outage simulation testing
Key technical indicators	
% Batch Changes	Batch job, scheduling, and data integrity testing
% Legacy Changes	Regression, integration, and manual testing for legacy components
Level of Componentisation	Integration, interface, and cross-boundary error handling testing
Level of Test Automation	Automated regression, performance, and integration testing
Amount of Data Change	Regression, migration, and data integrity validation
Depth of Supply Chain	End-to-end, integration, and disruption simulation testing
Implementation Process	Full regression, cutover, rollback, and deployment scenario testing

Recommendations

1. ITLF to publicise the risk-based testing approach for operational systems. This prioritizes critical business processes, customer-facing services, and operational continuity—helping organizations distinguish between high and low-risk changes in operational systems. New technology and automation can affect testing across all types of changes, and so highlight best practice including microservices, cloud, and complex supply chains.

2. ITLF to discuss with SIGIST the development of education and guidance for BCS members on tailoring testing approaches to business criticality, customer impact, and system complexity, and the use of testing tools.

3. ITLF to include operational testing in a paper on *The use of AI for improving service resilience*.

4. We are conscious that the range of tools marketed for testing is vast and growing: this is not the subject of this paper, however we recommend that SIGIST considers developing (or promoting if it already exits) such a guide.

Appendix A: Testing Terminology

Software testing encompasses a broad range of approaches, each designed to validate different aspects of a system. Below is a structured overview of the main types of testing that can be performed on operational systems[8].

Functional Testing: that the system performs its intended functions

- Unit Testing: Tests individual components or functions in isolation to ensure they work as intended.
- Integration Testing: Checks the interaction between integrated units or modules to confirm they work together properly.
- System Testing: Validates the complete and integrated system against requirements.
- End-to-End Testing: Simulates real user scenarios to ensure the entire application flow works as expected.
- Acceptance Testing: Confirms the system meets business requirements and is ready for deployment. This includes User Acceptance Testing (UAT).
- Smoke Testing: Verifies basic functionality after a new build to ensure the system is stable enough for further testing.
- Sanity Testing: Focuses on specific functionality after changes to confirm that bugs have been fixed and no further issues are introduced.

Non-Functional Testing: how the system operates rather than what it does

- Performance Testing: Evaluates speed, responsiveness, and stability under load[9].
- Load Testing: Measures system behaviour under expected user loads.
- Stress Testing: Assesses performance under extreme conditions.
- Scalability Testing: Determines the system's ability to scale up or down.
- Stability Testing: Checks if the system can run continuously without crashing.
- Security Testing: Identifies vulnerabilities and ensures data protection.
- Usability Testing: Assesses user-friendliness and overall user experience.
- Compatibility Testing: Ensures the system works across different devices, browsers, and operating systems.

[8] Source – Sarah can you provide please?

[9] See for instance https://www.lambdatest.com/blog/performance-tests-with-control-charts/

- Accessibility Testing: Verifies that the system is usable by people with disabilities

Specialized Testing: focus on specific aspects or scenarios.

- Regression Testing: Ensures that new changes do not adversely affect existing functionality.
- Recovery Testing: Tests the system's ability to recover from failures or crashes.
- API Testing: Validates the functionality, reliability, and security of application programming interfaces.
- GUI (Graphical User Interface) Testing: Checks the user interface for comprehensibility by real users, for consistency, layout, and usability.
- Localization Testing: Verifies the system's adaptability to different languages and regions.
- A/B Testing: Compares two versions of a feature to determine which performs better.
- Mutation Testing: Evaluates the quality of test cases by introducing small changes to the code and checking if tests catch them.
- Exploratory Testing: Involves informal, unscripted testing to discover defects through exploration.
- Ad-hoc Testing: Performed without planning or documentation, focusing on finding defects by intuition.
- Stress testing: using potentially extreme scenarios to drive out failure conditions. Also sometimes called Challenge testing, to try to break the system.
- Testing to demonstrate regulatory conformance.