Live Data Patterns Compendium

Appendix to *Data Patterns in Performance Testing* by Ross Collard

Twelve categories of live data patterns that can be used in performance testing are described and broken down into sub-types

1. Mainstream Patterns -- Unenhanced

Patterns which are unlikely to be significantly enhanced include:

Routine Live Data
Live data is captured or extracted “as is”, without massaging. The data is not tied to any special event; it is ordinary.

Baseline
A baseline test measures performance in the existing situation, as the “before” part of a before-and-after comparison. The same test data (or as close as possible) is re-run after a change, and the results compared.

Batch Volume or Parallel
In test mode, we run sizeable volumes of live batch transactions “in parallel”, before and after a system modification, then compare their performance.

Service Level Agreement (SLA) Compliance
SLAs are agreements among service providers and their clients, specifying the levels of service in terms of response times, throughput, error rates, etc, under pre-defined conditions (loads, resources).

Benchmark
A benchmark is a standard work load for testing, and does not represent any particular user’s perspective of reality. Benchmarks are used to compare competing systems or system versions.

Cyclic
Much live data cycles over time, and each cycle contains a basic pure sine wave with superimposed harmonics.

Bellwether
In the stock market, a bellwether stock indicates trends for its market sector, such as pharmaceuticals. As the price of the bellwether moves, we expect it to reflect the sector’s aggregate, but there is no guarantee.

Background Noise
When resources are shared, other applications besides the system under test (SUT) can place concurrent demands on the same infrastructure. Collectively, these demands are called background noise.

Peak
Testing with “normal” peak loads, is important: many systems cannot handle peak demands.

Load Variation
In this type of testing, the load varies during the performance measurement, to reflect typical load ebbs and flows over time.

Ramp-Up
A ramp-up test measures the time needed to turn on or initiate a process and reach steady state.

Emergency Shut-Down
When a life-critical system encounters a problem that imperils safety, we may need to verify that it shuts down within a guaranteed time limit.

Breakpoint
A breakpoint test increases load until the SUT reaches its breaking point.

2. Mainstream Patterns – Enhanced

Pristine
Data massaging attempts to undo prior data cleansing and conversion. Little or no data is available in truly raw form.

Truncated
To reduce test execution time, we truncate (a) an input data stream, by dropping transactions, and (b) databases accessed by the transactions.

Minimally Redundant
We trim data volume for test efficiency, while preserving referential integrity and thus data coordination.

Growth
Trends are extrapolated, with variations based on growth assumptions (optimistic, most likely, pessimistic).

Smoothed
Live data is adjusted by curve fitting and removal of outliers, e.g., to produce a composite “best fit” curve.

Restored / Refreshed
Data updates incurred during testing are reversed, in order to re-run the same tests, and dates are updated to counter the aging of the test data.

Privacy Protected
Data masking and scrambling preserve anonymity while also preserving referential integrity (essentially, cross-referencing among data items).

Corner Case
A corner case is a combination of extreme but valid data values.

Service Level Agreement (SLA) Violation
We adjust test data deliberately to fit an unacceptable scenario according to the SLAs, then run the data to see if it triggers an alert, initiates a recovery process, or takes other corrective action.

Peak-Peak
Live data is enhanced to exaggerate spikes in demand.

Pre- / Post-Baseline Usage Change
Often a functional change such as re-engineering a user work flow has a byproduct: it triggers a change in the usage pattern.

Data Stream
Streaming means continuous transmission rather than message-by-message, and lost or corrupted data may not be able to be re-sent. Characteristics of data streams include high data volumes and high throughput or bandwidth.

CRUD
CRUD stands for create-read-update-delete. A CRUD diagram shows each test data field, and what tests create, read, update, or delete it.

3. Measurable Behavior

Response Time (External)
This testing measures how long the system takes to complete a task or group of tasks. It usually represents the user viewpoint, i.e., we measure the likely delay as perceived by an external user.

Response Time (Internal)
We measure the efficiency of an internal software activity or hardware component which is not directly accessible by the user.

Throughput
Throughput testing measures how much traffic passes through a system or how many work units were completed, within a specified period of time and under a specified load.

Availability
Traditional availability is the percentage of uptime for a system or component, so testing availability is essentially a process of recording when the system is up or down.

Resource Utilization
Monitoring the levels of utilization of system resources provides insights into how the system works, helps to identify bottlenecks, assess spare capacity and scalability, and how to improve efficiency.

Testability
Much of a system's behavior may be hidden and not directly observable from the outside, which severely limits the effectiveness of non-invasive black-box testing. To be testable, a system has to be (a) observable and (b) controllable.

Capacity Forecasting
Capacity is the ability of a system to grow or to support an additional work load without degrading performance to an unacceptable degree.

This type of testing aims to measure whether the allocated resources are sufficient for the job, how much spare capacity still remains in system for further growth of demand, and at what point in the growth the resources supporting the system will need to be upgraded.

Measurement of Delays (Latency)
To be able to measure response times for particular events, we need to assume a straightforward cause-and-effect relationship: this stimulus triggers that outcome.

Loss Measurement
In networks especially, losses are a way of life: signals can attenuate (weaken). In a congested switch, blocking may cause a loss – all ports or connections into the switch are already busy, and the system simply drops an incoming message when the input hopper (buffer) is already full. In packet-switched digital networks, a data packet can be lost in transit.

Error Rate Measurement
Since this type of testing counts the incidence of errors or failures, we need a catalog of errors. Some lists contains items relevant to system and network administrators, such as “race conditions: timing out of sync.”, “memory leaks”, “page locking”, and “processor saturation”, but which are meaningless to the end users.

Revenue and Cost Measurement
These metrics can help justify costs. They usually are localized to the situation.

4. Stress Patterns

Accelerated Playback
Replaying test data at higher-than-live speed is a crude, fast and simple stress test. Sometimes test data must be adjusted to avoid execution problems in replay.

Zero Think Time
Test data will be replayed with user reaction pauses (“think times”) set to zero.

Super-User
Having a fictional “super-user” who does the work of several is a way to reduce the number of virtual users, which is important if a tool vendor prices by the virtual user headcount.

Hot Spot
Hot spot testing focuses demands on a specific, limited portion of the system, in order to detect if it is a weak point.

Shared Resources Contention
Test data is adjusted to allow for resource contention, e.g., when a database back-up runs during live operation. Not the same as background noise.

High Availability
Measures availability of "always on" service, 24x7 or 24x365. The higher a system's availability, the harder to prove it can be met.

Positive / Negative (P/N)
In this context, P/N does not mean above or below zero, but valid or invalid. Valid (positive) values of individual data items are replaced by negative ones in the test data.

Divide-by-Zero
Test data is deliberately seeded with one or multiple opportunities to divide by zero.

Boundary Value (BV)
A boundary value is a data value which lies on a boundary, or just inside or just outside. Boundary value testing uses test data on the boundaries.

De-Stabilization
This method uses random perturbations of test data to evaluate the robustness of a system. Techniques such as mutation analysis can de-stabilize the code base or change configuration settings.

Bottleneck Identification / Localization
These patterns focus on problem isolation, diagnosis and debugging, rather than purely performance measurement.

Endurance
This testing places a load on the system for an extended period of time, to detect slow-to-appear, delayed-fuse bugs such as memory leaks, wild pointers or buffer overflows.

Spike and Bounce
Utilizes an intense spike in the work load, usually for a very short duration, to observe how the system handles abrupt increases in demand. A variation follows the spike with a very low load, and then repeats the ups and downs.

Breakpoint
In this testing, we steadily increase the load until the system fails.

Extreme Configuration
Many systems contain internal switch settings which allow them to be customized. Extreme configurations are worst-case settings, either the largest and the most complex or the minimalist, resource-constrained ones. Similar but not the same as a corner case;

Dirty Configuration
A dirty configuration is one which is not supported, but users expect systems to run regardless. This testing examines robustness in dirty environments.

Perturbation (Chaos Butterfly Effect)
In an unstable system, tiny perturbations to inputs and initial conditions cause large changes in outputs and the system behavior.

5. System Architecture-Based

Component, Subsystem or Tier-Specific
This type of testing examines the robustness of one component or subassembly. Can be done as soon as the component is ready, well before the fully integrated system is ready for testing. May require component test drivers, which can be expensive to build.

Calibration or Settings Measurement
This type of testing is interleaved with tuning, and its purpose is to provide feedback on the consequences of each iteration of tuning.

Scalability
This type of testing investigates a system's ability to grow. Growth can occur in several ways, which we may need to separately test: increase in the total load; increase in the number of concurrent users; increase in the size of a database; increase in the number of devices connected to a network, and so on.

Compatibility and Configuration
This method considers the various configurations in which a system can be used, and how to check for compatibility or consistent behavior across these configurations.

6. Interactions

Rendez-vous
This is a type of spike testing where at least two, and more likely many, events rendezvous. I.e., they happen simultaneously, or within a small enough interval that they could influence each other.

Feature Interaction / Interference
This type of testing attempts to stress a system by having features processes or threats interfere with each other.

Interoperability and Interface
Errors can occur because of mismatches at interfaces.

Deadlock
This stresses a system by locking a database, either directly or through deadlock transactions which interfere with each other.

Synchronization
This type of testing attempts to stress a system by causing timing problems and out-of-synch process. These are also called race conditions.

7. Human Error

"Bad Day"
A user scenario test is one which employs a real-world set of activities, based on how the users actually utilize the system.

Soap Opera
This melodrama is a user scenario test where we exaggerate the day-by-day actions of the users. (Many storm-in-teacup crises are packed into a half-hour soap.)

8. Catastrophe

Disaster Recovery
This type of testing uses the disaster scenarios which were identified in the organization's disaster recovery plans as a source of test cases.

9. Physical Failure

Environmental
Testing for physical factors like the loss of power, vibration, G (gravity) forces, air pollutants in factories such as paint solvents, electric shock, electromagnetic radiation, extremes of temperature, humidity and so on.

10. Handling Changes

Live Change
Many systems must keep running, no matter what. One example is an aircraft flying over the ocean. What happens when an emergency fix or routine maintenance must be done without taking the system down?

System Change Impact Assessment
Assesses the impact of a change or group of changes to an existing system.

Infrastructure Impact Assessment
Assesses the impact of a change on an existing infrastructure.

11. Handling Errors

Error Detection & Recovery
We reverse-engineer the system (i.e., analyze the flow backwards from outcome to cause), for each error condition that we expect the system to issue, and devise test data to trigger it.

Degraded Mode of Operation
The purpose of degraded mode testing is to determine whether the system can still provide the reduced level of service as expected.

Fault Injection
Software fault injection provides testers with the capability to easily and safely trigger or simulate system errors which otherwise might be very difficult to observe in the test lab.

12. Risk-Based

Risk Prioritization
This method uses a risk assessment to identify and prioritize the likely risks which the system faces in live operation. We use this risk assessment to allocate test resources to the various aspects of the system, i.e., to focus the test effort to the areas which need the depth and intensity.

Hazard or Threat Identification
This method, which is similar to risk-based testing, considers the dangers and obstructions associated with using a system.

Failure Modes Effects and Assessment (FMEA)
We test is to see if we can make the system fail within the relatively safe and controlled confines of the test lab, in order to observe the conditions under which the system fails, how it fails (what happens), and whether it recovers in an acceptable manner.

