Are you in it for the long haul?

- Introduction
- Defining “long-haul”
- Why do we need long-haul testing?
- Can vs. should: goals and non-goals
- One category, many flavors
- Automation design considerations
- What success looks like
Introduction

• Modern application workloads involve **hours or days of continuous operation**
• Unit/functional/integration testing are **necessary** but **not sufficient**
• Consider **long-haul testing** to measure **availability/reliability** over **longer term use**

---

Defining “long-haul”

• Like stress, long-haul is a specialization of **load testing**
• But they are **not** the same thing!

<table>
<thead>
<tr>
<th>Long-haul</th>
<th>Stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercises <strong>typical</strong> application workloads and <strong>controlled</strong> faults concurrently and continuously</td>
<td>Exercises <strong>extreme</strong> situations and <strong>heavy</strong> faults concurrently and continuously</td>
</tr>
<tr>
<td>Runs for hours to days</td>
<td>Runs for hours to days</td>
</tr>
<tr>
<td>Stays within nominal system limits</td>
<td>Exceeds limits, pushes past the breaking point</td>
</tr>
<tr>
<td>Expects system to remain operational</td>
<td>Expects system to fail in some way</td>
</tr>
<tr>
<td>Demonstrates adherence to predefined SLAs</td>
<td>Demonstrates graceful degradation and recovery</td>
</tr>
<tr>
<td>Fairly broad and probabilistic</td>
<td>Relatively constrained and repeatable</td>
</tr>
</tbody>
</table>
Why do we need long-haul testing?

• Traditional testing is heavy on functional and integration tests
  • Straightforward and predictable, pre-planned
  • Provide quick feedback, short-term quality indicators

Why do we need long-haul testing?

• What are we missing?
  • Complex, concurrent multi-feature/multi-user interactions
  • Operational behavior over extended periods of time
  • Controlled chaos – impact of occasional faults on typical workloads
Why do we need long-haul testing?

- **Enter long-haul!**
  - Relies less on pre-planned workflows – gives new data without new (explicit) test cases
  - Compresses time and scale – minimizes execution cost
  - Stays within operational limits – a good measure of your system’s SLAs

Why do we need long-haul testing?

- A sampling of long-haul bugs
  - **Slow leak**
    - Repeated operations over a period of hours resulted in slow but steady memory growth.
  - **State poisoning**
    - Race condition resulted in a momentary undefined state change; state was saved and system crashed any time state was restored.
  - **Too much information**
    - Tracing was too verbose; diagnostic data files were too big to be useful after the system ran for long enough time.
Why do we need long-haul testing?
• What about single-user, low concurrency, limited operational duration systems?
  • Long-haul principles are still useful
  • Can get broad coverage of positive and negative code paths with reasonable test cost

Can vs. should: goals and non-goals
• Goal: reduce the cost of testing
  • Less orchestration, less planning
  • Random actions and faults
  • Overall, fewer tests with wider product coverage
• Non-goal: supplant functional testing
  • Functional testing is still best for guaranteed regression coverage
Can vs. should: goals and non-goals

• **Goal:** uncover race conditions and invalid states
  • Long-haul = randomness + parallelism + time
  • Breadth of long-haul means lots of state coverage
• **Non-goal:** exhaustively validate all behavior
  • Long-haul relies more on heuristics than strict expected results

Can vs. should: goals and non-goals

• **Goal:** provide valuable feedback for ship-readiness
  • Demonstrates longer-term reliability
  • Validates (or challenges) assumptions about system capabilities
• **Non-goal:** provide quick feedback
  • Long-haul needs time
  • Many other tests exist to provide fast results
Can vs. should: goals and non-goals

- **Goal:** leverage controlled chaos
  - Think “*many* things go” not “*anything* goes” – not too predictable but not too random
- **Non-goal:** “monkey” testing
  - Too much undirected randomness is a liability
  - Results are very difficult to analyze, resulting “bugs” are difficult to prioritize

One category, many flavors

- Long-haul tests come in all shapes and sizes
- Exact taxonomy depends on the team, product, and context
  - The examples I give are adapted from my past experience
  - Not a definitive or exhaustive list
One category, many flavors

• Low-level
  • “One-box” test, written close to the code
  • Observes internal state for more detailed validation

• Feature/subsystem
  • Focuses on a specific area of the product
  • Purposely constrained to limit complexity
  • Virtuous feedback loop between functional tests and feature long-hauls

One category, many flavors

• Customer workload
  • Long-running acceptance test for particular use case
  • Best employed for uncommon or specialized configurations

• Full-system
  • Exercises cross-component workloads and system-level faults
  • Can be quite powerful but also difficult to develop and analyze
Automation design considerations

• First, decide on the basic test architecture and topology
  • Typical example: “A long-haul test drives continuous concurrent actions and faults with periodic validations.”
  • Many decision points here, depending on the needs of the project

Automation design considerations

• **Test driver:** simple loop, distributed work scheduler?
• **Action:** function, class/interface, separate executable?
• **Concurrency:** fixed, parameterized, adaptive?
• **Faults:** external vs. internal, scope/targets, how to recover?
• **Validations:** complete vs. partial, sync vs. async?
Automation design considerations

• Separate actions from validations
  • Same action can be performed at different times with different expected results
  • Think of an action as a data producer and a validation as a data consumer

Automation design considerations

• Parameterize test inputs
  • Long-haul tests often require tuning
  • Ensure configurability without requiring code changes
  • Consider parameterization of data values and even actions/validations
Automation design considerations

• Use profiles to guide decisions
  • Mine production data or do market research to define typical workloads
  • Example: a “casual user” makes 10 requests per hour, a “power user” makes 1000 requests per hour; 20 : 1 ratio of casual to power users.

Automation design considerations

• Partition disparate test actors
  • Consider a test of an online folder shared by many users
  • Indiscriminately adding and removing files makes validation difficult
  • Instead, use the shared folder itself as a logical partition to group different user workloads
    • Folder 1: one writer, many readers
    • Folder 2: multiple writers (different files)
    • Folder 3: multiple writers (same files)
    • ... and so on, until your test matrix is satisfied
Automation design considerations

- Coordinate invasive faults
  - Too many faults at the same time or in quick succession can turn long-haul into stress
  - Some care is required in scheduling faults to avoid undue pressure and stay within limits

- Optimize for diagnosability
  - Long-haul can never guarantee reproducibility
  - Instead, strive for diagnosability
  - Test and product should have sufficiently detailed logs to aid in root cause analysis
  - Be mindful of log sizes; consider circular/segmented logs to keep the data manageable
What success looks like

• A successful long-haul testing effort involves **communication** and **collaboration** across the team
  • Use a **common vocabulary**
  • Agree on the **scope and target**
  • Build a **realistic schedule**
  • **Iterations**: crawl, walk, run
  • **Hold the bar**

What success looks like

• **Use a common vocabulary**
  • Make sure your team understands and uses the same terms to describe long-haul tests
  • Be ready to compare/contrast similar load testing activities
What success looks like

• Agree on the scope and target
  • Which flavors of long-haul tests?
  • Which behaviors will you focus on?
  • What validations will you apply?
  • What are the expected results?
  • How long will the tests run?

What success looks like

• Build a realistic schedule
  • Long-haul tests need sufficient time to design and execute
  • Last minute issues found by long-haul tests can add days to the end of a cycle
  • A proper schedule must account for these risks and uncertainties
What success looks like

• Iterations: crawl, walk, run
  • Slowly build up the breadth and rigor of long-haul tests and focus on small, realistic objectives
    • Example: in iteration 1, one long-haul test of one major feature area, target duration of four hours, validate the product does not crash
    • In iteration 2, expand to two feature areas, target duration of eight hours, etc.

What success looks like

• Hold the bar
  • Assess risks and make informed decisions as a team
    • Do not unilaterally relax exit criteria or ignore issues
  • Be clear about changes to scope, validation, etc. of long-haul tests
    • Do not subject the team to a moving target
Are you in it for the long haul?

• Questions?