Watering the Roots of Resilience: Learning from Failure with Decision Trees
Systems resilience depends on the ability to adapt and evolve to changing conditions.
Our software systems are complex, sociotechnical, and difficult to reason about.
Humans are the mechanism for adaptation in our complex software systems.
How can SREs align their mental models of the system with reality to sustain resilience?
I. Adaptation in Complex Systems

II. Resilience Stress Testing with Chaos Experiments

III. Refining Mental Models with Decision Trees
I. Adaptation in Complex Systems
Complex systems present a large variety of possible states; prediction is impossible.
Getting from point A to point B in complex systems is more as a cat zoomies vs. crow flies.
The reality is failure is inevitable; it’s a natural part of complex systems as they operate
Complex systems are adaptive: they evolve in response to changes in their environment.
Adaptive capacity: how poised a system is to change how it works based on context
Resilience is “the ability to prepare and plan for, absorb, recover from, and more successfully adapt to adverse events.”
What is the resilience potion recipe? There are five ingredients to sustain resilience...
Define the system’s critical functions
Define the system’s safe boundaries
Observe system interactions across spacetime
Feedback loops and a learning culture
Flexibility and willingness to change
How does this look in our computer systems? What is “adaptation” in them?
Our software systems have machines and humans continually influencing each other.
When machine processes fail in ways that are noticeable, humans jump in.
Software has limited ability to adapt on its own. Humans are the primary adaptive capability.
Consider Log4Shell: real-world harm was low due to the socio part of the system.
How we learn influences how we adapt to stressors, surprises, and adverse conditions.
The software we design, build, and operate reflects our mental models of reality.
Naturally, our mental representations of reality are incomplete and inconsistent.
Surprise is “the revelation that a given phenomenon of the environment was, until this moment, misinterpreted.”
We must “prepare to be surprised.”
II. Resilience Stress Testing with Chaos Experiments
Our goal is to uncover “baffling interactions” in our systems that defy our expectations.
We can do so through chaos experiments: resilience stress tests for software systems.
Experiments can generate evidence of how much our mental models deviate from reality.
Chaos experiments help us more quickly learn about system behavior and its context.
We conduct system-level adverse scenarios rather than evaluating specific components.
Many weaknesses only emerge once the system is, in effect, a living thing...
We can fix things in production by learning from adverse conditions via experiments.
We’re curious about assessing the nature of the system and its interconnections.
How are chaos experiments different than any other kind of test?
<table>
<thead>
<tr>
<th>Chaos experiments</th>
<th>Typical tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support resilience</td>
<td>Support robustness</td>
</tr>
<tr>
<td>Sociotechnical (includes humans as part of the system)</td>
<td>Technical (excludes humans as part of the system)</td>
</tr>
<tr>
<td>System-focused</td>
<td>Component-focused</td>
</tr>
<tr>
<td>Capable of being run in production</td>
<td>Must run in dev or staging</td>
</tr>
<tr>
<td>Random or pseudo-random</td>
<td>Scripted</td>
</tr>
<tr>
<td>Adverse scenarios</td>
<td>Boolean requirements</td>
</tr>
<tr>
<td>Observe and learn from failures</td>
<td>Detect and prevent failures</td>
</tr>
<tr>
<td>N-order effects</td>
<td>Direct effects</td>
</tr>
</tbody>
</table>

Source: Security Chaos Engineering (O'Reilly Media, 2023)
Interlude: a Case Study
Example: physical parts database for its manufacturing sites on both coasts in the USA
How do their systems behave when replication is severed or when it gets too far behind?
Potential chaos experiment: severing the database replication between the two sites
Hypothesis proven incorrect: replication didn’t work; requests still served from the West Coast
Design change: halting the west coast data-center if it isn’t caught up to the primary
Re-run the experiment: replication was faster than expected (yay!) but no alerts fired (oof!)
Disruptions can include security issues, too, since they can become reliability problems.
Attackers love to take advantage of interactions between components to compromise a system.
Introduction to Security Chaos Engineering
Security Chaos Engineering (SCE):
a socio-technical transformation that enables the organizational ability to gracefully respond to failure and adapt to evolving conditions.
SCE aligns mental models with reality and improves our systems’ resilience to attack.
How do we create a security chaos experiment in practice?
Like any experiment, we start with a hypothesis: our assumptions (mental models) about reality.
We can target our “this will always be true” assumptions that exist all over our stack.
Parsing this string will always be fast
Messages on this port will always be post-authentication
An alert will always fire if a malicious executable appears.
Example hypothesis: “If a user accidentally or maliciously introduced a misconfigured port, we would immediately detect, block, and alert on the event.”
Figure 2-6. An example security chaos experiment simulating a misconfigured port injection scenario

Result: Hypothesis disproved. Firewall did not detect or block the change on all instances. Standard Port AAA security policy out of sync on the Portal Team instances. Port change did not trigger an alert and log data indicated successful change audit. However, we unexpectedly learned the configuration mgmt tool caught change and alerted the SoC.
What other experiments are relevant to SREs? There’s ample overlap with security.
Each exposes how our sociotechnical system behaves in an adverse scenario *end to end*. 
When we reveal the resilience properties of our systems, how do we capture this knowledge?
III. Refining Mental Models with Decision Trees
The question isn’t “is the system resilient?”
It’s instead: “the resilience of what, to what?”
Decision trees are a visual representation of different events possible in a scenario.
Decision Tree Walkthrough
In general, decision trees map how adverse events and mitigations unfold across spacetime.
Security decision trees map attacker choices and visualize their paths through the system.
The point isn’t perfection; it is iteration that keeps us honest about our mental models.
Adverse Scenario - Missing Logs

Reality

- #yolo giving all your hosts the same hostname
- #yolo misconfigured log daemon
- #yolo blocking network egress; logs can't be shipped
- #yolo forgetting to install the log daemon

Missing logs (good luck with incident response)
Adverse Scenario - Missing Logs

Reality

YoLo

misconfigured log daemon

expired TLS certificates

configuring the wrong path to gather logs

Missing logs (good luck with incident response)
Adverse Scenario - Missing Logs

Reality

#yold

misconfigured log daemon

expired TLS certificates

configuring the wrong path to gather logs

monitor the root certificate store

standardize log paths; all prod services write to same path

smoke tests to verify working logs

Missing logs (good luck with incident response)
Adverse Scenario - Missing Logs

Reality
- forgetting to install the log daemon
  - yolo
  - include daemon in base image
- forgetting to start the log daemon
  - yolo
- smoke tests to verify working logs
- monitor the log daemon
- start the log daemon

Missing logs (good luck with incident response)
Applying Decision Trees in Practice
Decision trees cover all issues relevant in your system — including any “solved” ones.
Decision trees capture system architecture and flows, plus any gaps where we’re unprepared.
Experiments form a feedback loop with decision trees to refine your mental models.
Decision trees help us continuously refine system architecture to sustain resilience.
They’re also valuable during incident reviews — see where your assumptions held true or false.
Did events unfold in the flow modeled in the decision tree?
Did your mechanisms alter the flow or sequence of events as anticipated?
What events or actions were missing from your assumptions?
Were there mitigations you didn’t expect?
You can also document decision trees before your chaos experiments as hypotheses.
We can also assess the potential efficacy of an architectural change through experiments.
Each design + experiment iteration refines decision trees and thus your mental models.
Starting experimentation along the easiest branches verifies “obvious” assumptions
Once you gain confidence in resilience to obvious failures, you can move onwards
You may never get to the “super hard” failures branch when there are many ongoing changes.
IV. Conclusion
Our software systems are inherently socio-technical in nature; humans are how they adapt.
To adapt successfully in a complex system, we must continuously refine our mental models.
We can expose gaps in our understanding of the system's reality through chaos experiments.
With this evidence in hand, we can capture our evolving knowledge in decision trees.
From there, we can complete the feedback loop on which resilience and reliability depends.
Preorder the book & stay tuned for its release in late April (next month!): 

[Amazon](#)  
[Bookshop](#)