Reliability Engineering

What is the Core of Reliability Engineering?

1. Insuring each part of the product does what it is supposed to do: **DESIGN, TEST COVERAGE**
2. Insuring each component and the entire system can respond gracefully to its environment: **FAILURE MODE ANALYSIS (FMA)**
3. Insuring each component and the system is robust enough to manage 'unexpected' events: **TESTING**
4. Insuring each component and service is monitored and alerts when things are not going well: **MONITORING, ALERTING**
5. Changes that have system impact are formally managed and communicated: **CHANGE MANAGEMENT**
6. Insuring operations knows how to characterize and respond to any problem or alert: **RUNBOOK**
7. Minimizing response time to return system to full operation. **INCIDENT RESPONSE AUTOMATION**
8. Continuous feedback to improve quality: **INCIDENT ANALYSIS**

Success Factors

DESIGN

- All components have well defined and documented interface api's.
- All specifications have quantitative functional metrics.
- Engineers write unit tests for every component/functional module and provide for testing every spec metric.
- Data validation done on all communication paths.
- Errors are meaningful to operations and have explicit runbook entries.
- Maximizing of centralized configuration parameters.
- Documentation of failure modes and thresholds. (Production Operations Passport)
- Explicit linkage between bug/feature JIRA ticket, SVN code checkin, and unit test so all are kept in sync.

TEST COVERAGE

- All component unit tests are run for every functional metric.
- Subsystem tests verify and test intra-module communication api's.
- Isolated and clean test environment.
- Load testing and external environment simulation.
- Hostile and malicious attack testing.
- Continuous extension of all tests and test cases.
- Chaos testing: Inject random data or shut off machines in QA, staging, (and production).

FAILURE MODE ANALYSIS: COIN

Evaluate every functional module, subsystem, and service based on the acronym COIN:

- **C**onfusion: How will I respond if I don't get the type, class, integrity, quality, range, or rate of input I expect?
- **O**verload: How will I respond if my input increases drastically, whether slowly or rapidly?
- **I**solation/Throttle: How will I respond if I am unable to get full access/bandwidth to supporting data or provide data to a downstream source?
- **N**otification: Are all of the critical failures communicated by timely, meaningful, and helpful error messages.

All falure modes are originated and updated by developers for use by operations.

- Production Operations Passport

TESTING

- Every critical function is tested.
- Every critical performance metric is evaluated and failure points determined and documented.
- New failure modes get rolled into regression testing.

MONITORING

- Representative paths covering majority of user geography are monitored.
    - Distributed Client Simulators
- Every parameter that impacts reliability and functionality of system monitored.
    - Monitoring Audit and Gap Analysis
    - Collectd Production Monitoring Configuration

- Monitoring system provide trend analysis for pro-active intervention before negative user impact observed.
    - Production Operations Dashboard
- Trend analysis is done and results are fed back to engineering for quality/robustness improvements.

ALERTING

- Every platform and service function monitor that detects a negative impact on the user experience generates an alert.
- All alerts provide accurate characterization of the problem to maximize response effectiveness.
- All critical alerts have a response defined in the runbook.
- There is a dedicated 7x24 team to respond to alerts with deterministic escalation paths and service level agreements.

CHANGE MANAGEMENT

Product Change Management

- Product changes are fully regression tested in QA and stage prior to going to production.
    - Production Deploy Test Drive Process
- Production changes are communicated to all support parties and time correlatable with monitoring and alerting systems.
- All production changes rollback-able.

Infrastructure Change Management

- Production changes are communicated to all support parties and time correlatable with monitoring and alerting systems.
- All production changes rollback-able.
- Software and configuration data comes from source as close to hosts as possible, not from HQ
- Data sources and movement insure data accuracy and integrity of data delivered to production
    - Production Configuration Update System
- Single build/management system insures consistent changes and updates to entire system.
    - Centralized Infrastructure Management
- Uniformity and 'known-state' of production systems simple, clear, deterministic, and atomic.

RUNBOOK

- Operations runbook addresses every known observable failure.
    - Production Operations Passport
- Every critical alert has a procedure to return full operation status.

INCIDENT RESPONSE AUTOMATION

- Clearly defined and promulgate Production Incident Response Process.
- Simple runbook procedures automated.
- Automated response scripts are centrally managed under same strict change management as product.
- Automated responses are aggressively reduced in number by adding more robustness into product design.
- Automated responses monitored and analyzed for input to design enhancement.

INCIDENT ANALYSIS (AND FEEDBACK)

- Every performance or functionality degradation creates an incident report.
- Incidents are characterized by user experience and revenue impact.
- Every incident has an investigation for root cause.
- All root causes are explicitly addressed by a change to product design, change processes, system monitoring, and/or runbook.

Improvement Plan

Measurement and Baseline Characterization

1. Zabbix production monitor coverage audit
2. Collectd data coverage audit
3. Product failure history analysis

4. Zabbix production monitor coverage enhancement
5. Collectd data collection coverage enhancement
6. Trend analysis and automated alerting enhancement

Incident Tracking, Analysis, Correction

1. Create incident tracking system
2. Create root cause analysis and resolution process
3. Feedback failure modes to engineering for feature/process correction

Monitoring and Alerting Enhancement

1. Identify critical health parameters
2. Create and deploy monitors for all critical health parameters
3. Automate monitor deployment with new systems
4. Create operations escalation process
5. Create Ops Runbook and set up for continuous improvement

Infrastructure Scaling and Change Management Enhancement

1. Reverse engineer current deployment into specifications
2. Select deployment/configuration management system (SaltStack?)
3. Migrate to unified deployment system