Principles of Software Construction: Objects, Design, and Concurrency

DevOps

Christian Kästner  Vincent Hellendoorn
Topics

From CI to CD

Containers

Configuration management

Monitoring

Feature flags, testing in production
Where we are

<table>
<thead>
<tr>
<th>Small scale: One/few objects</th>
<th>Mid scale: Many objects</th>
<th>Large scale: Subsystems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtype Polymorphism ✓</td>
<td>Domain Analysis ✓</td>
<td>GUI vs Core ✓</td>
</tr>
<tr>
<td>Information Hiding, Contracts ✓</td>
<td>Inheritance &amp; Del. ✓</td>
<td>Frameworks and Libraries ✓, APIs ✓</td>
</tr>
<tr>
<td>Immutability ✓</td>
<td>Responsibility Assignment, Design Patterns, Antipattern ✓</td>
<td>Module systems, microservices ✓</td>
</tr>
<tr>
<td>Types ✓</td>
<td>Promises/Reactive P. ✓</td>
<td>Testing for Robustness ✓</td>
</tr>
<tr>
<td>Static Analysis ✓</td>
<td>Integration Testing ✓</td>
<td>CI ✓, DevOps, Teams</td>
</tr>
<tr>
<td>Unit Testing ✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Design for understanding change/ext. reuse robustness ...

Small scale: One/few objects

- Subtype Polymorphism ✓
- Information Hiding, Contracts ✓
- Immutability ✓
- Types ✓
- Static Analysis ✓
- Unit Testing ✓

Mid scale: Many objects

- Domain Analysis ✓
- Inheritance & Del. ✓
- Responsibility Assignment, Design Patterns, Antipattern ✓
- Promises/Reactive P. ✓
- Integration Testing ✓

Large scale: Subsystems

- GUI vs Core ✓
- Frameworks and Libraries ✓, APIs ✓
- Module systems, microservices ✓
- Testing for Robustness ✓
- CI ✓, DevOps, Teams
Recall: Continuous Integration
Continuous Integration

- Automation
- Ensures absence of obvious build issues and configuration issues (e.g., dependencies all checked in)
- Ensures tests are executed
- May encourage more tests
- Can run checks on different platforms

- What can all be automated?
Any repetitive QA work remaining?
Releasing Software
Semantic Versioning for Releases

- Given a version number MAJOR.MINOR.PATCH, increment the:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards-compatible manner, and
  - PATCH version when you make backwards-compatible bug fixes.

- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

http://semver.org/
Versioning entire projects
Release management with branches
Release cycle of Facebook's apps

- Development
  - 2 weeks of development
  - Master branch
    - Release branch
      - 3 daily dog-food builds
      - Stabilizing
      - Soak
      - Cherry-picks
      - Deployment
Facebook Tests for Mobile Apps

Unit tests (white box)
Static analysis (null pointer warnings, memory leaks, ...)
Build tests (compilation succeeds)
Snapshot tests (screenshot comparison, pixel by pixel)
Integration tests (black box, in simulators)
Performance tests (resource usage)
Capacity and conformance tests (custom)

Release Challenges for Mobile Apps

Large downloads
Download time at user discretion
Different versions in production
Pull support for old releases?

Server side releases silent and quick, consistent

-> App as container, most content + layout from server
From Release Date to Continuous Release

● **Traditional View: Boxed Software**
  ○ Working toward fixed release date, QA heavy before release
  ○ Release and move on
  ○ Fix post-release defects in next release or through expensive patches

● **Frequent releases**
  ○ Incremental updates delivered frequently (weeks, days, …), e.g. Browsers
  ○ Automated updates (“patch culture”; “updater done? ship it”)

● **Hosted software**
  ○ Frequent incremental releases, hot patches, different versions for different customers, customer may not even notice update
Continuous Delivery

Unit Test | Platform Test | Deliver to Staging | Application Acceptance tests | Deploy to Production | Post deploy tests
Auto | Auto | Auto | Auto | Manual | Auto

Continuous Deployment

Unit Test | Platform Test | Deliver to Staging | Application Acceptance tests | Deploy to Production | Post deploy tests
Auto | Auto | Auto | Auto | Auto | Auto
The Shifting Development-Operations Barrier
WORKED FINE IN DEV

OPS PROBLEM NOW
Common Release Problems?
Common Release Problems (Examples)

- Missing dependencies
- Different compiler versions or library versions
- Different local utilities (e.g. unix grep vs mac grep)
- Database problems
- OS differences
- Too slow in real settings
- Difficult to roll back changes
- Source from many different repositories
- Obscure hardware? Cloud? Enough memory?
The Dev – Ops Divide

- Coding
- Testing, static analysis, reviews
- Continuous integration
- Bug tracking
- Running local tests and scalability experiments
- …

- Allocating hardware resources
- Managing OS updates
- Monitoring performance
- Monitoring crashes
- Managing load spikes, …
- Tuning database performance
- Running distributed at scale
- Rolling back releases
- …

QA responsibilities in both roles
QA Does not Stop in Dev
QA Does not Stop in Dev

- Ensuring product builds correctly (e.g., reproducible builds)
- Ensuring scalability under real-world loads
- Supporting environment constraints from real systems (hardware, software, OS)
- Efficiency with given infrastructure
- Monitoring (server, database, Dr. Watson, etc)
- Bottlenecks, crash-prone components, ... (possibly thousands of crash reports per day/minute)
Efficiency of release pipeline

https://www.slideshare.net/jmcgarr/continuous-delivery-at-netflix-and-beyond
DevOps
Key Ideas and Principles

Better coordinate between developers and operations (collaborative)

Key goal: Reduce friction bringing changes from development into production

Considering the entire tool chain into production (holistic)

Documentation and versioning of all dependencies and configurations ("configuration as code")

Heavy automation, e.g., continuous delivery, monitoring

Small iterations, incremental and continuous releases

Buzz word!
Common practices

- Code: Version control, dependency management, review
- Build: Continuous integration, independent builds
- Test: Automated test execution on every build
- Package: Deploying binary to repository/staging area
- Release: Change management, deployment, rollback of packages
- Configure: Manage and configure infrastructure, automated
- Monitor: Monitor performance, crashes, … and possibly automated reaction
Common Practices

All configurations in version control
Test and deploy in containers
Automated testing, testing, testing, ...
Monitoring, orchestration, and automated actions in practice
Microservice architectures
Release frequently
Heavy Tooling and Automation
Heavy tooling and automation -- Examples

Infrastructure as code — Ansible, Terraform, Puppet, Chef
CI/CD — Jenkins, TeamCity, GitLab, Shippable, Bamboo, Azure DevOps
Test automation — Selenium, Cucumber, Apache JMeter
Containerization — Docker, Rocket, Unik
Orchestration — Kubernetes, Swarm, Mesos
Software deployment — Elastic Beanstalk, Octopus, Vamp
Measurement — Datadog, DynaTrace, Kibana, NewRelic, ServiceNow
DevOps: Tooling Overview
DevOps Tools

- Containers and virtual machines (Docker, …)
- Orchestration and configuration (ansible, Puppet, Chef, Kubernetes, …)

- Sophisticated (custom) pipelines
- Lightweight virtualization
- Sub-second boot time
- Sharable virtual images with full setup incl. configuration settings
- Used in development and deployment
- Separate docker images for separate services (web server, business logic, database, …)
Scripts to change system configurations (configuration files, install packages, versions, …); declarative vs imperative

Usually put under version control

```
$nameservers = ['10.0.2.3']   (Puppet)
file { '/etc/resolv.conf':
  ensure => file,
  owner => 'root',
  group => 'root',
  mode => '0644',
  content => template('resolver/r.conf'),
}
```
Container Orchestration with Kubernetes

Manages which container to deploy to which machine
Launches and kills containers depending on load
Manage updates and routing
Automated restart, replacement, replication, scaling
Kubernetes master controls many nodes
Monitoring

- Monitor server health
- Monitor service health
- Collect and analyze measures or log files
- Dashboards and triggering automated decisions
  - Many tools, e.g., Grafana as dashboard, Prometheus for metrics, Loki + ElasticSearch for logs
  - Push and pull models
Testing in Production
Testing in Production

"Don’t worry, our users will notify us if there’s a problem"
Chaos Experiments
Crash Telemetry

Crash2.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, [click here].

[Send Error Report] [Don't Send]
A/B Testing

Original: 2.3%

SaaS & eCommerce Customer Support.

"Managing customer support requests in Groove is so easy. Way better than trying to use Gmail or a more complicated help desk."

97% of customers recommend Groove.

Learn More

How it works
What you get
What it costs
How we're different

You'll be up and running in less than a minute.

Long Form: 4.3%

Everything you need to deliver awesome, personal support to every customer.

Assign support emails to the right people, feel confident that customers are being followed up with and always know what's going on.

ALLAN USES GROOVE TO GROW HIS BUSINESS. HERE'S HOW:

WHAT YOU'LL DISCOVER ON THIS PAGE

- Three reasons growing teams choose Groove
- How Groove makes your whole team more productive
- Delivering a personal support experience every time
- Take a sneak preview
- A personal note from our CEO

1000+ HAPPY CUSTOMERS
WHAT IF...?

... we hand plenty of subjects for experiments

... we could randomly assign subjects to treatment and control group without them knowing

... we could analyze small individual changes and keep everything else constant

▶ Ideal conditions for controlled experiments
Experiment Size

With enough subjects (users), we can run many many experiments

Even very small experiments become feasible

Toward causal inference
IMPLEMENTING A/B TESTING

Implement alternative versions of the system
- using feature flags (decisions in implementation)
- separate deployments (decision in router/load balancer)

Map users to treatment group
- Randomly from distribution
- Static user - group mapping
- Online service (e.g., launchdarkly, split)

Monitor outcomes per group
- Telemetry, sales, time on site, server load, crash rate
FEATURE FLAGS

Boolean options
Good practices: tracked explicitly, documented, keep them localized and independent

External mapping of flags to customers
- who should see what configuration
- e.g., 1% of users sees one_click_checkout, but always the same users; or 50% of beta-users and 90% of developers and 0.1% of all users

```python
if (features.enabled(userId, "one_click_checkout")) {
    // new one click checkout function
} else {
    // old checkout functionality
}
```

```python
def isEnabled(user):
    Boolean = (hash(user.id) % 100) < 10
```
Comparing Outcomes

Group A

base game
2158 Users
average 18:13 min time on site

Group B

game with extra god cards
10 Users
average 20:24 min time on site
Experiment Created

The percentage of visitors who clicked on a tracked element.

Variations

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Conversions / Visitors</th>
<th>Conversion Rate</th>
<th>Baseline</th>
<th>Chance to beat Baseline</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test it out</td>
<td>462 / 3,568</td>
<td>12.9% (±1.1%)</td>
<td></td>
<td>100.0%</td>
<td>+25.4%</td>
</tr>
<tr>
<td>Give it a try</td>
<td>440 / 3,479</td>
<td>12.6% (±1.1%)</td>
<td></td>
<td>99.9%</td>
<td>+22.5%</td>
</tr>
<tr>
<td>Try it out</td>
<td>395 / 3,504</td>
<td>11.3% (±1.0%)</td>
<td></td>
<td>90.2%</td>
<td>+9.2%</td>
</tr>
<tr>
<td>Original Page</td>
<td>378 / 3,662</td>
<td>10.3% (±1.0%)</td>
<td></td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

Conversion Rate Over Time

- Original Page
- Try it out
- Test it out
- Give it a try
The Morality Of A/B Testing

Josh Constine @joshconstine / 11:50 PM EDT • June 29, 2014

We don't use the "real" Facebook. Or Twitter. Or Google, Yahoo, or LinkedIn. We are almost all part of experiments they quietly run to see if different versions with little changes make us use more, visit more, click more, or buy more. By signing up for these services, we technically give consent to be treated like guinea pigs.

But this weekend, Facebook stirred up controversy because one of its data science researchers published the results of an experiment on 689,003 users to see if showing them more positive or negative sentiment posts in the News Feed would affect their happiness levels as deduced by what they posted. The impact of this experiment on manipulating emotions was tiny, but it
Canary Releases
Canary Releases

Testing releases in production
Incrementally deploy a new release to users, not all at once
Monitor difference in outcomes (e.g., crash rates, performance, user engagement)
Automatically roll back bad releases
Technically similar to A/B testing
Telemetry essential
Canary Releases
Canary Releases at Facebook

Phase 0: Automated unit tests
Phase 1: Release to Facebook employees
Phase 2: Release to subset of production machines
Phase 3: Release to full cluster
Phase 4: Commit to master, rollout everywhere

Monitored metrics: server load, crashes, click-through rate

Real DevOps Pipelines are Complex

- Incremental rollout, reconfiguring routers
- Canary testing
- Automatic rolling back changes

Chaos Experiments
Two more things
TAing in Spring 2022?

Enjoyed content of this class?
Practicing critiquing other designs?
Thinking through design problems with other students?

If interested, talk to us or apply directly at
https://www.ugrad.cs.cmu.edu/ta/S22/ (select 17214)
Course feedback please:

Summary

Increasing automation of tests and deployments

Containers and configuration management tools help with automation, deployment, and rollbacks

Monitoring becomes important

Many new opportunities for testing in production (feature flags are common)