SWEN438 - DevOps

Lecture 10 - Change Mitigation

Dr David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Reducing Batch Size

- **Code Progressively**
- **Find Stepping Stones**
- **Develop on Trunk**
- **Use Feature Flags**

Code Progressively

```
/**
 * Move blinky towards a given target square.
 * This is determined by calculating ...
 */
int moveTowards(Board board, int tx, int ty) {
  ...
```

- What changes can we make that won't break production?
- Can **add** new classes / fields / methods
- Can **move** methods or classes around
- Can **rename** methods or classes

Code Progressively

June 12, 2012 — Product

GitHub API v2: End of Life

Risk Olson

We have shut down API v1 and v2 as promised.

Just a reminder, you should follow @GitHubAPI on Twitter for the latest updates about the API. API v3 keeps a changelog of breaking changes, with plans for the next revision of v3.

Feedback

Let us know through support@github.com or our Contact form if you have any feedback. Last time, we received a lot of concerned questions regarding search support, and documented its API v3 port.

- **API Versioning**. Can run multiple versions of API side-by-side!

Small Stepping Stones

Implement Pinky functionality:

- **Add** new class `Pinky` based on `Blinky`
- **Add** **`new`** `Pinky(..)` to `Board()` constructor
- **Move** `moveTowards()` into `Ghost`
- **Update** targeting logic in `Pinky`
- **Remove** `@Disabled` from tests now passing

Trunk-Based Development

> *"Trunk-based development is a version control management practice where developers merge small, frequent updates to a core* **trunk** *or main branch. Since it streamlines merging and integration phases, it helps achieve CI/CD and increases software delivery and organizational performance."*
>
> *– K. Zettler*

- Developers create **short-lived** branches with **few commits**
- Alternative to the "Gitflow" approach which encourages **long-lived** branches
- Ideal is to merge branches on a **daily basis**

See "Trunk-Based Development", Atlassian.com

Feature Flags

"At the culminating moment when the feature goes live, no new code is pushed into production. Instead, we merely change a **feature toggle** *or configuration setting. The new feature is slowly made visible to small segments of customers, automatically rolled back if something goes wrong"*

– The Phoneix Project

Feature Flags

> *"**Feature toggles** are usually implemented by wrapping application logic or UI elements with a conditional statement, where the feature is enabled or disabled based on a configuration setting store somewhere. This can be as simple as an application configuration file"*
>
> *– DevOps Handbook*

Benefits of Feature Flags

- **Roll Back**. If something goes wrong, can untoggle flag and quickly revert to running state.
- **Graceful Degredation**. If production is to high, can untoggle features to reduce computational overheads.
- **Incomplete Features**. Can deploy code to production that depends on services not yet available, provided is hidden behind feature toggle.

Launch Darkly

> "**Feature toggles** *allow us to deploy features into production without making them accessible to users, enabling a technique known as* dark launching*."*
>
> *– The DevOps Handbook*

- Can test features in **production environment**
- Can sample first (e.g. 1% of users)
- Enables **progressive** roll-out (e.g. to a few users, then more, etc)

Hidden Chat

“As part of their **dark launch** *process, every Facebook user session, which runs JavaScript in the user browser, had a test harness loaded into it — the chat UI elements were hidden, but the browser client would send invisible test chat messages to the back-end chat service that was already in production, enabling them to simulate production-like loads throughout the entire project, allowing to find and fix performance problems long before the customer release.”*

– The DevOps Handbook

Feature Flags: Example

```
class Compiler {
  private boolean useNewTypeChecker = false;

  public void compile(SourceFile file) {
    ...
    if(useNewTypeChecker) {
      new NewTypeChecker().check(file);
    } else {
      new TypeChecker().check(file);
} } }
```

- Minimal feature toggle which cannot change dynamically

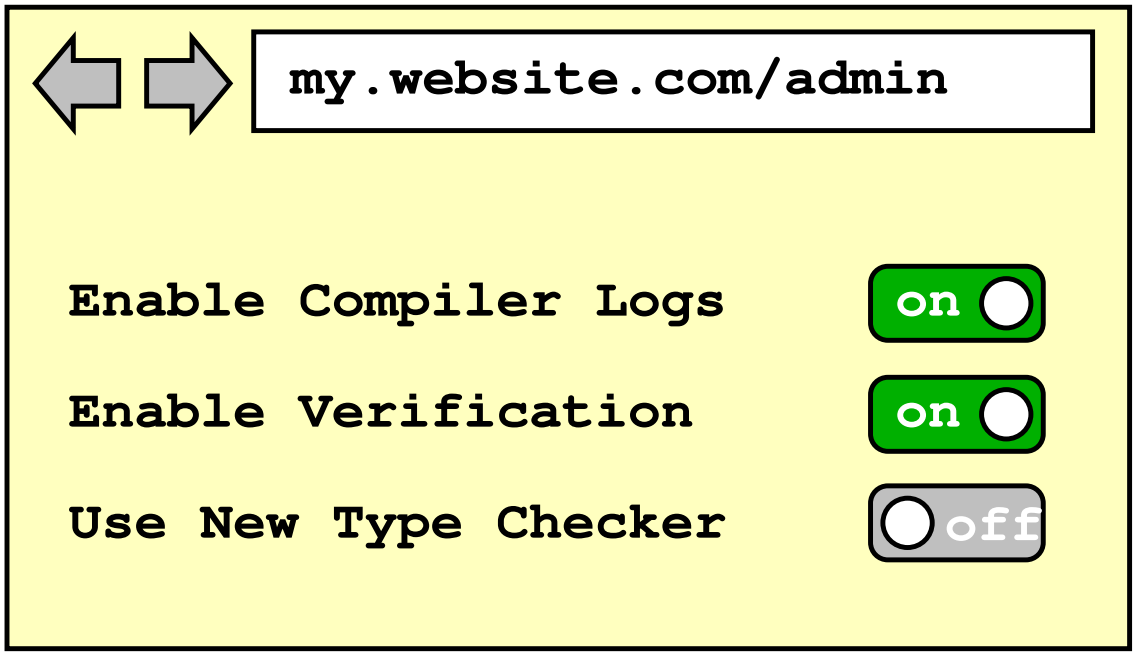
Feature Flags: Example

```
class Compiler {
  private Set<String> features;

  public void compile(SourceFile file) {
    ...
    if(features.contains("useNewTypeChecker") {
      new NewTypeChecker().check(file);
    } else {
      new TypeChecker().check(file);
} } }
```

- Feature set passed into `Compiler` constructor
- Feature set can be populated dynamically (e.g. from JSON)

Feature Flags: Decision Making



- **Static** based on configuration file
- **Dynamic** through e.g. an admin interface
- **Dynamic** based on HTTP request context (e.g. cookie)

Feature Flags: Roll Out

- **Canary Release:**
  - » Release to **small percentage** of users (e.g. 1%)
  - » Systems remembers **which users** have feature enabled
  - » Monitor both groups for **key metrics** (e.g. number of requests)
- **A/B Testing:**
  - » Enable feature for **larger cohort** of users
  - » Monitor metrics for **feedback** on feature

Feature Flags: Types

- **Release Toggle.** Allow incomplete and/or untested code to be shipped in production. They **facilitate** trunk-based development
- **Experiment Toggle.** Used for **A/B testing** where “toggle router” chooses which features are enabled each cohort
- **Ops Toggle.** Protects a feature which could have operational consequences (e.g. causes high load)
- **Permissioning Toggles.** Allow features based on cohort (e.g. internal users, premium users, etc)

References

- **DevOps Radio, Episode 09**

  “Pete Hodgson on Feature Flagging and Scaling withing Startups”

- **The ChangeLog, Episode 447**

  “The foundations of Continuous Delivery”

- **“Feature Toggles”, Martin Fowler’s Blog**

  martinfowler.com/articles/feature-toggles.html

- **“What Are Feature Flags?” LaunchDarkly**

  launchdarkly.com/blog/what-are-feature-flags