Code Review

CS350 Introduction to Software Engineering

Shin Yoo

Code Inspection

Michael Fagan, IBM Systems Journal, 1976

- A systematic inspection of both design and code between each important phase (e.g., inspect design before implementation, inspect code before testing)
- Inspection
  - Six stages: planning, overview, preparation, inspection meeting, rework, follow-up
  - Four roles: moderator (manages the process and leads the team), designer, implementor, tester

Code Inspection

Michael Fagan, IBM Systems Journal, 1976

- Preparation (whole team): designer describes the overall area, moderator picks up specific aspects of scrutiny
- Preparation (individual): understand the design as well as recent error types
- Inspection (whole team): first, collectively review the design, and subsequently, try to find errors in the code
- Rework: address all problems, either design or implementation
- Follow-up: moderator ensures that all issues are properly addressed

Code Inspection

- Properly done, this would be very effective!
- Also, slow (as in *not very agile*), time-consuming, and synchronous (involving multiple individuals)
- What are the modern equivalent?

Modern Code Review

Modern Code Review: A Practice at Google, ICSE SEIP 2018

- Informal: much fewer fixed roles, fewer steps, etc, when compared to Fagan
- Tool-based: logistics are handled by tools, instead of the moderator
- Asynchronous: no meetings, done via online communication
- Focused on code changes: rather than inspecting an entire lifecycle stage
- The paper is a strongly recommended read, if you want to understand the modern software development lifecycle and the daily activities of developers.

Motivation

Modern Code Review: A Practice at Google, ICSE SEIP 2018

- Original adoption: *“to force developers to write code that other developers could understand”*, which was considered important since code is the *“teacher for future developers”*
  - Additional benefits: checking for consistent style and design, ensuring adequate test cases, and improved security (no one can commit arbitrary code without oversight)
- Current expectations: education, maintaining the norm, gatekeeping, and accident prevention

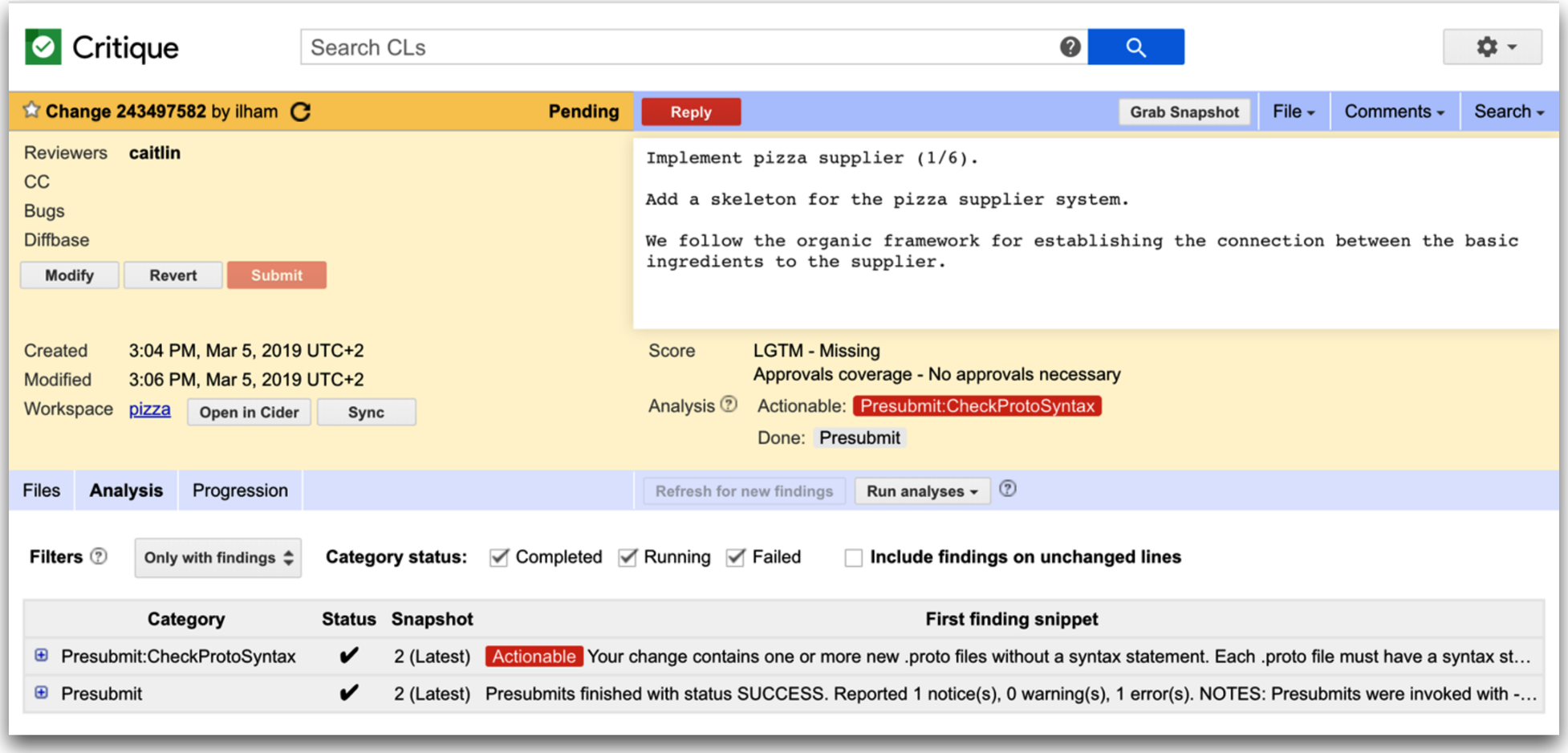
Process

Modern Code Review: A Practice at Google, ICSE SEIP 2018

- Creating the change by adding, removing, editing the code
- Preview: using the code review tool, Critique, the developer analyses the change (static code analysis is involved); then the diff is sent out for review

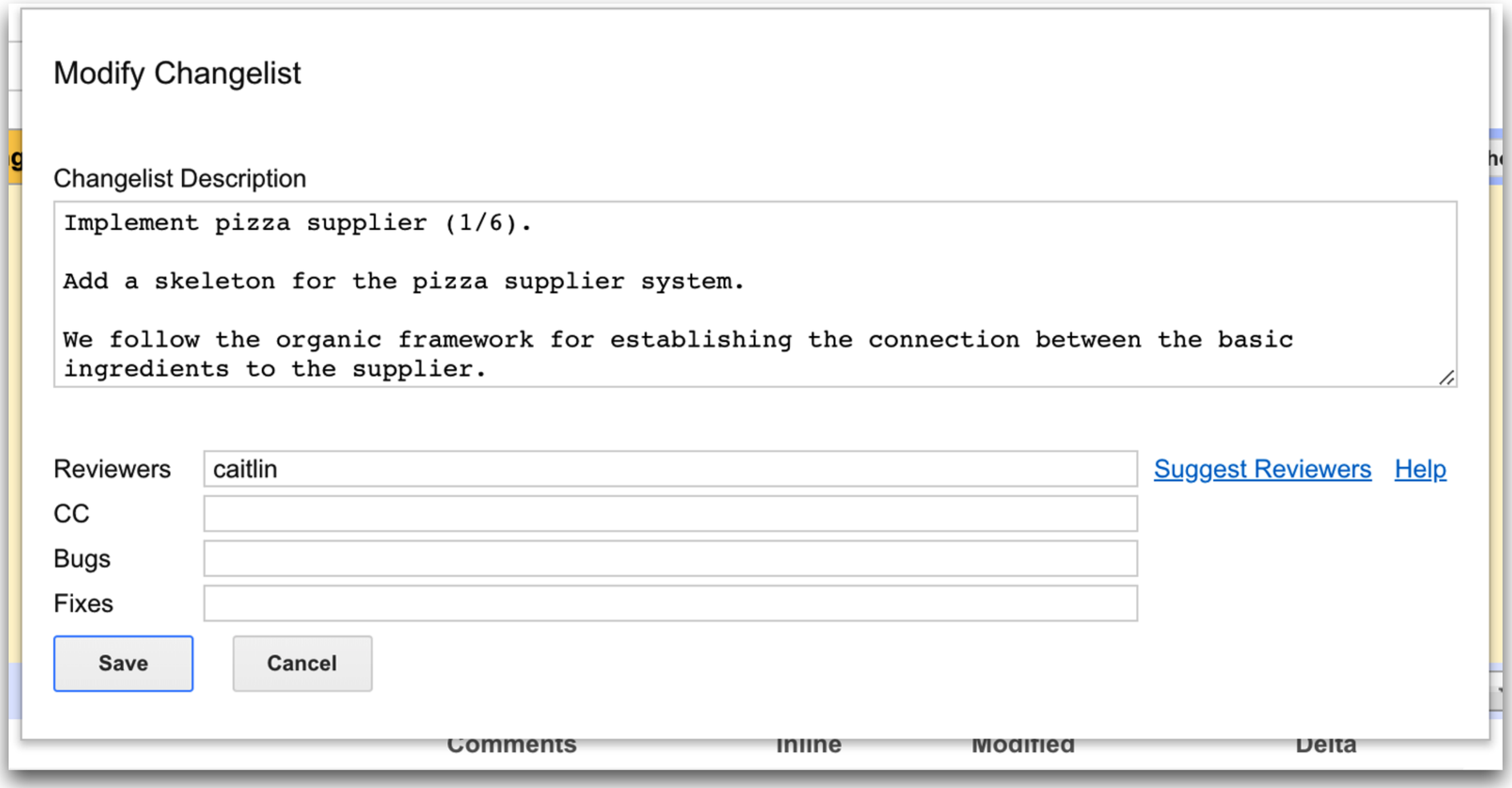
Process

Modern Code Review: A Practice at Google, ICSE SEIP 2018



Process

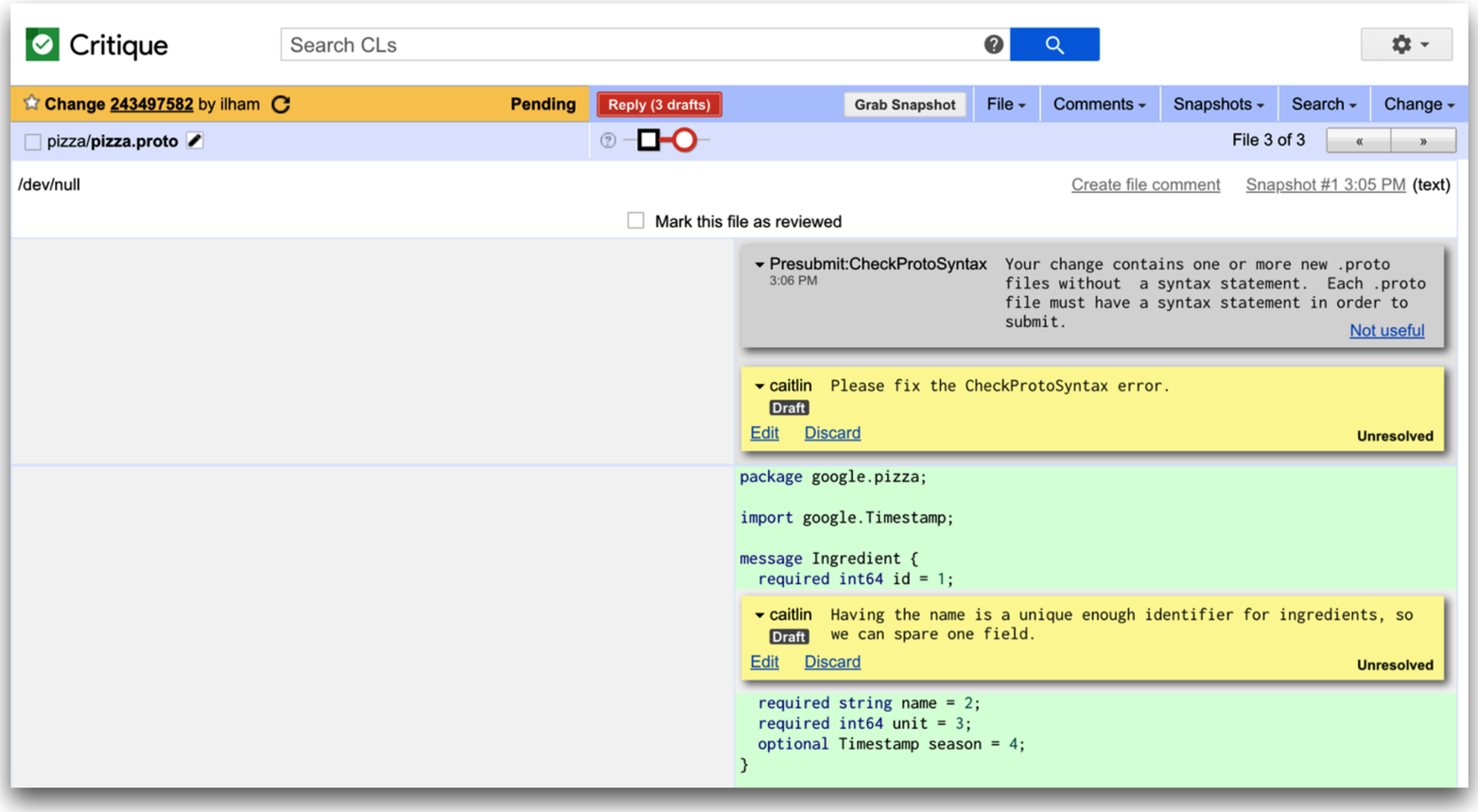
Modern Code Review: A Practice at Google, ICSE SEIP 2018



Process

Modern Code Review: A Practice at Google, ICSE SEIP 2018

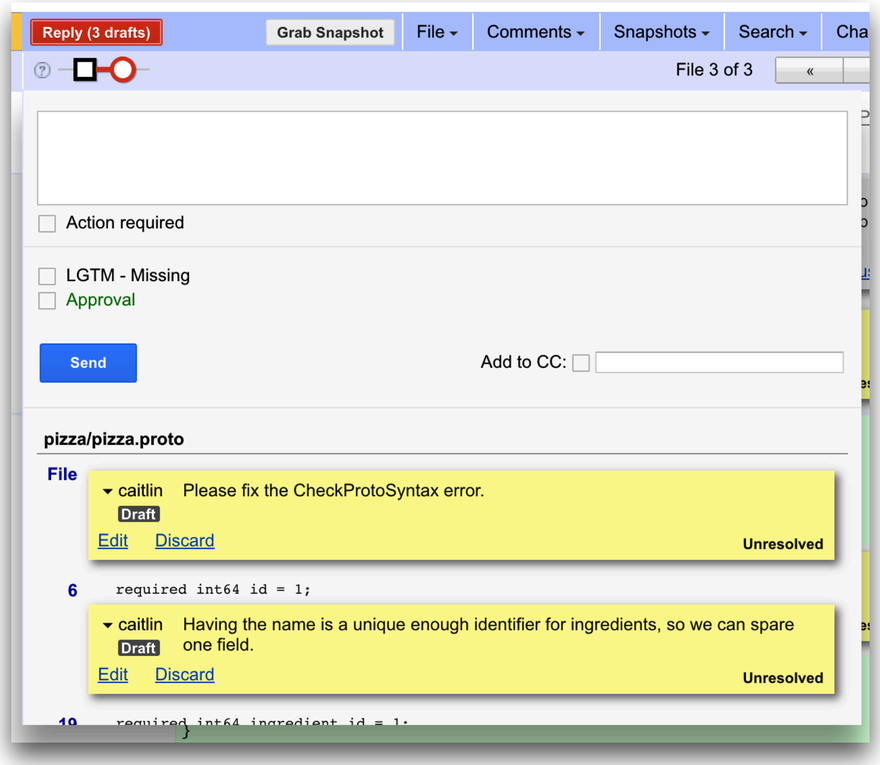
- Commenting: reviewers use GUI to comment on the change and the analysis results



Process

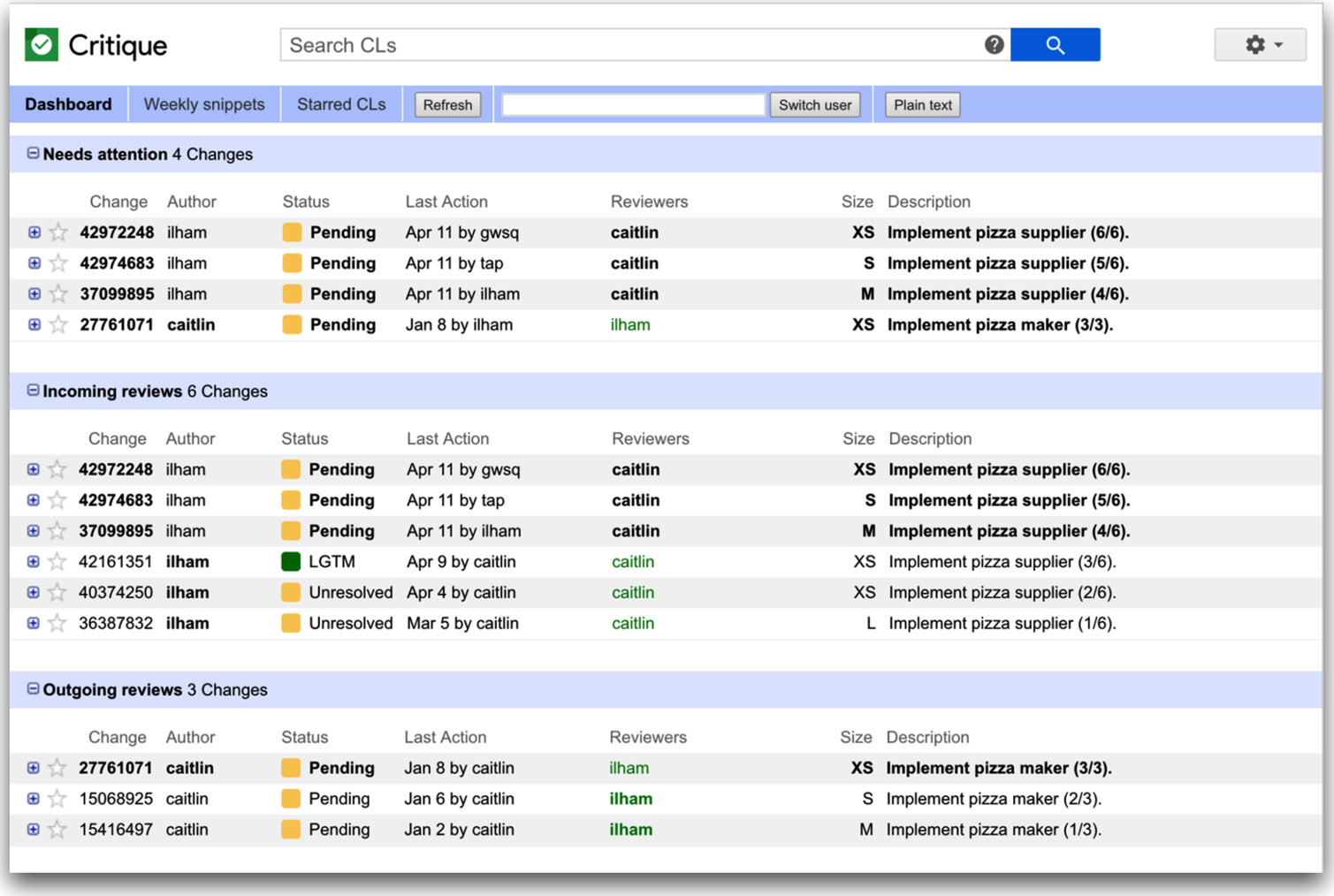
Modern Code Review: A Practice at Google, ICSE SEIP 2018

- Addressing the feedback: developer either updates the code change, or respond to the comments, until all are resolved



Process

Modern Code Review: A Practice at Google, ICSE SEIP 2018



Process

Modern Code Review: A Practice at Google, ICSE SEIP 2018

- Creating the change by adding, removing, editing the code
- Preview: using the code review tool, Critique, the developer analyses the change (static code analysis is involved); then the diff is sent out for review
- Commenting: reviewers use GUI to comment on the change and the analysis results
- Addressing the feedback: developer either updates the code change, or respond to the comments, until all are resolved
- Approving: the change is finally approved, and the commit is made

Process

Modern Code Review: A Practice at Google, ICSE SEIP 2018

- Creating the change by adding, removing, editing the code
- Preview: using the code review tool, Critique, the developer analyses the change (static code analysis is involved); then the diff is sent out for review
- Commenting: reviewers use GUI to comment on the change and the analysis results
- Addressing the feedback: developer either updates the code change, or respond to the comments, until all are resolved
- Approving: the change is finally approved, and the commit is made

Reviewer Suggestion

Modern Code Review: A Practice at Google, ICSE SEIP 2018

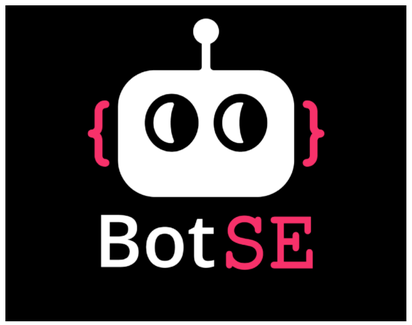
- Within a team: typically round robin, without any need for tool support, pending holidays and current reviewing load
- Outside the team: Critique identifies the smallest set of reviewers that can process the change under consideration
  - Prioritises people who recently modified/reviewed the changed file
  - Prioritises people who are new members of the team that owns the file (so that they can gain reviewing credits)

Let’s look at a real world example

- In GitHub Workflow, code review is typically performed for incoming PRs
- Here are a few recent PRs:
  - https://github.com/pandas-dev/pandas/pull/52974
  - https://github.com/google/guava/pull/6308

Use of Automated Bots

- If we are to enforce some checks, easy and obvious ones should be automated!
- There are many bots that automatically act on incoming PRs.
  - https://github.com/reviewboard/ReviewBot
  - This is an up-and-coming, active research area: http://botse.org/



Okay, what to look out for?

- Language specific patterns are perhaps better detected by static analysis
- Humans are better at detecting higher level concerns
  - (Potential) bugs
  - Better coding style
  - Inappropriate design concepts

Potential Bugs

- Off-by-one errors
- Deviations from the specification
- Variable scopes (misuse of global, for example)
- Magic numbers
- Optimistic coding
- Do not Repeat Yourself (DRY)

Better Coding Style

- Inadequate variable naming
- Inconsistent formatting
- Too long/complicated method/control flow
- Having too much/too few comments :)

Inappropriate Design Concepts

- Incomplete/inconsistent specification
- Mutability/immutablity
- Incomplete data abstraction (revealing inner representation)

Example

Taken from https://web.mit.edu/6.005/www/fa15/classes/04-code-review/

```
public static int dayOfYear(int month, int dayOfMonth, int year) {
    if (month == 2) {
        dayOfMonth += 31;
    } else if (month == 3) {
        dayOfMonth += 59;
    } else if (month == 4) {
        dayOfMonth += 90;
    } else if (month == 5) {
        dayOfMonth += 31 + 28 + 31 + 30;
    } else if (month == 6) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31;
    } else if (month == 7) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30;
    } else if (month == 8) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31;
    } else if (month == 9) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;
    } else if (month == 10) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;
    } else if (month == 11) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;
    } else if (month == 12) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 31;
    }
    return dayOfMonth;
}
```

What would you comment on?

Example

Taken from https://github.com/MinBZK/woo-besluit-broncode-digid-app/

```
private static string GetPercentageRounds(double percentage)
{
    if (percentage == 0)
        return "○○○○○○○○○○";
    if (percentage > 0.0 && percentage <= 0.1)
        return "●○○○○○○○○○";
    if (percentage > 0.1 && percentage <= 0.2)
        return "●●○○○○○○○○";
    if (percentage > 0.2 && percentage <= 0.3)
        return "●●●○○○○○○○";
    if (percentage > 0.3 && percentage <= 0.4)
        return "●●●●○○○○○○";
    if (percentage > 0.4 && percentage <= 0.5)
        return "●●●●●○○○○○";
    if (percentage > 0.5 && percentage <= 0.6)
        return "●●●●●●○○○○";
    if (percentage > 0.6 && percentage <= 0.7)
        return "●●●●●●●○○○";
    if (percentage > 0.7 && percentage <= 0.8)
        return "●●●●●●●●○○";
    if (percentage > 0.8 && percentage <= 0.9)
        return "●●●●●●●●●○";

    return "●●●●●●●●●●";
}
```

Dutch government was forced to reveal the source of their DigiID authentication app on iOS.

This is a code snippet from their code repository. Is this good or bad? :)

Exercise For You (5~10 minutes)

```
public class Account {
    double principal,rate; int daysActive,accountType;
    public static final int STANDARD=0, BUDGET=1, PREMIUM=2, PREMIUM_PLUS=3;
}

// ...

public static double calculateFee(Account[] accounts) `{
    double totalFee = 0.0;
    Account account;
    for (int i=0;i<accounts.length;i++) {
        account=accounts[i];
        if(account.accountType==Account.PREMIUM|| account.accountType == Account.PREMIUM_PLUS )
            totalFee += .0125 * ( // 1.25% broker's fee
            account.principal*Math.pow
            (account.rate,(account.daysActive/365.25))
            - account.principal); // interest-principal
    }
    return totalFee;
}
```

```java
/** An individual account. Also see CorporateAccount. */
public class Account {
    private double principal;

    /** The yearly, compounded rate (at 365.25 days per year). */
    private double rate;

    /** Days since last interest payout. */
    private int daysActive;
    private Type type;

    /** The varieties of account our bank offers. */
    public enum Type {STANDARD, BUDGET, PREMIUM, PREMIUM_PLUS}

    /** Compute interest. **/
    public double interest() {
        double years = daysActive / 365.25;
        double compoundInterest = principal * Math.pow(rate, years);
        return compoundInterest â€“ principal;
    }

    /** Return true if this is a premium account. **/
    public boolean isPremium() {
        return accountType == Type.PREMIUM || accountType == Type.PREMIUM_PLUS;
    }

    /** The portion of the interest that goes to the broker. **/
    public static final double BROKER_FEE_PERCENT = 0.0125;
```

```
/** Return the sum of the broker fees for all the given accounts. **/
    public static double calculateFee(Account accounts[]) {
        double totalFee = 0.0;
        for (Account account : accounts) {
            if (account.isPremium()) {
                totalFee += BROKER_FEE_PERCENT * account.interest();
            }
        }
        return totalFee;
    }
}
```

Code Review is also communication

- Remember that one major motivation of code review is education
- Be polite: no sarcasm, insults, and any other derogative behavior. It is not okay to say you only meant that the “code” is stupid…
- Be constructive: the aim is not to evaluate, but to build something together - suggest improvements.
- Be positive: code review does not only have to be about fault-finding - I think it is okay to compliment exceptionally good/elegant design and creative solution; also you can thank people on specific feature sets, if the context is open source