Code review

Code review is a process in which code is reviewed by someone other than the author, often before the introduction of that code into a codebase. Although that is a simple definition, implementations of the process of code review vary widely throughout the software industry.

Some organizations have a select group of "gatekeepers" across the codebase that review changes. Others delegate code review processes to smaller teams, allowing different teams to require different levels of code review. **At Google, essentially every change is reviewed before being committed, and every engineer is responsible for initiating reviews and reviewing changes.**

Code reviews generally require a combination of a process and a tool supporting that process.

At Google, we use a custom code review tool, **Critique**, to support our process.

## Code Review Flow:

Code reviews can happen at many stages of software development. At Google, code reviews take place before a change can be committed to the codebase; this stage is also known as a *precommit review*.

A typical code review at Google goes through the following steps:
1. A user writes a change to the codebase in their workspace. This *author* then creates a snapshot of the change: a patch and corresponding description that are uploaded to the code review tool. This change produces a *diff* against the codebase, which is used to evaluate what code has changed.

2. The author can use this initial patch to apply automated review comments or do self-review. When the author is satisfied with the diff of the change, they mail the change to one or more reviewers. This process notifies those reviewers asking them to view and comment on the snapshot.

3. *Reviewers* open the change in the code review tool and post comments on the diff. Some comments request explicit resolution. Some are merely informational.

4. The author modifies the change and uploads new snapshots based on the feedback and then replies back to the reviewers. Steps 3 and 4 may be repeated multiple times.

5. After the reviewers are happy with the latest state of the change, they agree to the change and accept it by marking it as "looks good to me." Only one "looks good to me" is required by default, although convention might request that all reviewers agree to the change.

6. After a change is marked "**looks good to me**," the author is allowed to commit the change to the codebase, provided they *resolve all comments* and that the change is *approved*.

The primary end goal of a code review is to get another engineer to consent to the change, which we denote by tagging the change as "looks good to me." **We use this "looks good to me" as a necessary permissions "bit" to allow the change to be committed**.

## Code Is a Liability

It's important to remember (and accept) that code itself is a liability. It might be a necessary liability, but by itself, code is simply a maintenance task to someone somewhere down the line. Much like the fuel that an airplane carries, it has weight, though it is, of course, necessary for that airplane to fly.

*New features are often necessary, of course, but care should be taken before developing code in the first place to ensure that any new feature is warranted. Duplicated code not only is a wasted effort, it can actually cost more in time than not having the code at all; changes that could be easily performed under one code pattern often require more effort when there is duplication in the codebase.* ***Writing entirely new code is so frowned upon, that some of us have a saying "If you're writing it from scratch, you're doing it wrong!"***

This is especially true of library or utility code. Chances are, if you are writing a utility, someone else somewhere in a codebase the size of Google's has probably done something similar.

## Code Ownership

**When a team is small, everyone usually has access to all the code and can make changes anywhere. This works because:**

*** Everyone knows each other**

*** The project is small**

*** Mistakes are easier to manage**

**But as a team grows, this approach does not scale well. If many people change everything freely, the codebase becomes messy and hard to manage. To solve this, Google uses a system called ownership.**

**What Is Ownership?**

**Ownership does not mean you own the code like property. It means you are a steward — you are responsible for looking after a part of the codebase.**

**Owners are people who:**

*** Understand the part of the code they own**

*** Review and approve changes to that code**

*** Make sure it stays healthy**

*** Know who else can help if needed**

**What Are OWNERS Files?**

**In each folder of the codebase, there can be a file called OWNERS.**

**This file lists the usernames of the people responsible for that folder and its subfolders.**

**Key points:**

*** Each directory can have its own OWNERS file**

*** Ownership is additive: owners of a folder also own its subfolders unless overridden**

*** You can include as many owners as you want, but fewer is usually better for clarity**

**Tools can then read these files to know who must approve changes.**

**How Ownership Helps**

**This system makes large-scale engineering possible by allowing:**

**Local decision-making: Owners review their own parts without involving everyone.**

**Global changes: A few people in the root OWNERS file can approve huge changes across the codebase.**

**Better documentation: Anyone can find who is responsible for any file simply by checking the OWNERS files up the directory tree.**

**Easy project creation: New teams just add their OWNERS file; no central authority needed.**

**Why It Works**

**The mechanism is:**

**Simple**

**Scalable**

**Efficient**

**It ensures that thousands of engineers can work safely and effectively on a huge, shared codebase (billions of lines of code).**