Lessons In Sustainability

How to Maintain a C++ Codebase for Decades

Disclaimer

This is based on my experiences. Your milage may vary.

The comments represented herein do not represent an official policy of Google as a company. (But they are good suggestions, and you should listen.)

Sustainability

What does sustainability **mean**? Should you care?

- What's the expected lifespan of your code?
- Is your code per-project or monolithic?

Monolithic Codebase

Pros:

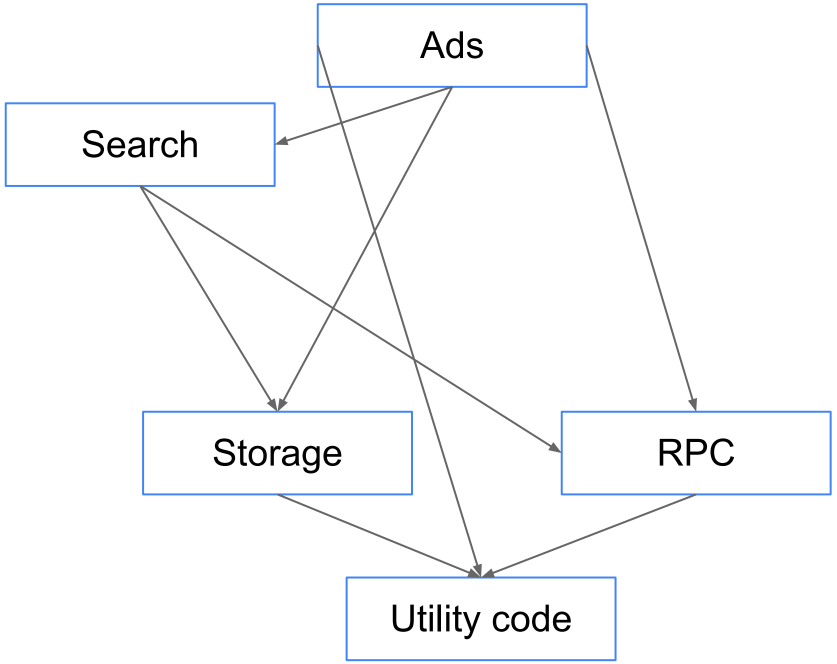
- Consistency
- Infrastructure scaling
- Infrastructure usage visibility

Cons:

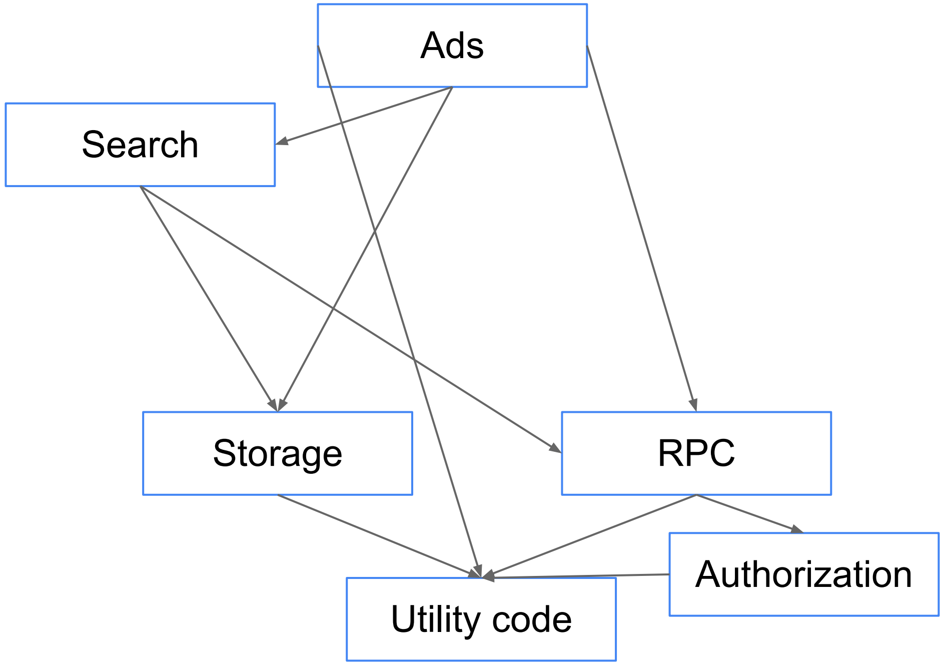
- Scaling (repo sync)
- Cannot just throw it away and start clean

No single right answer.

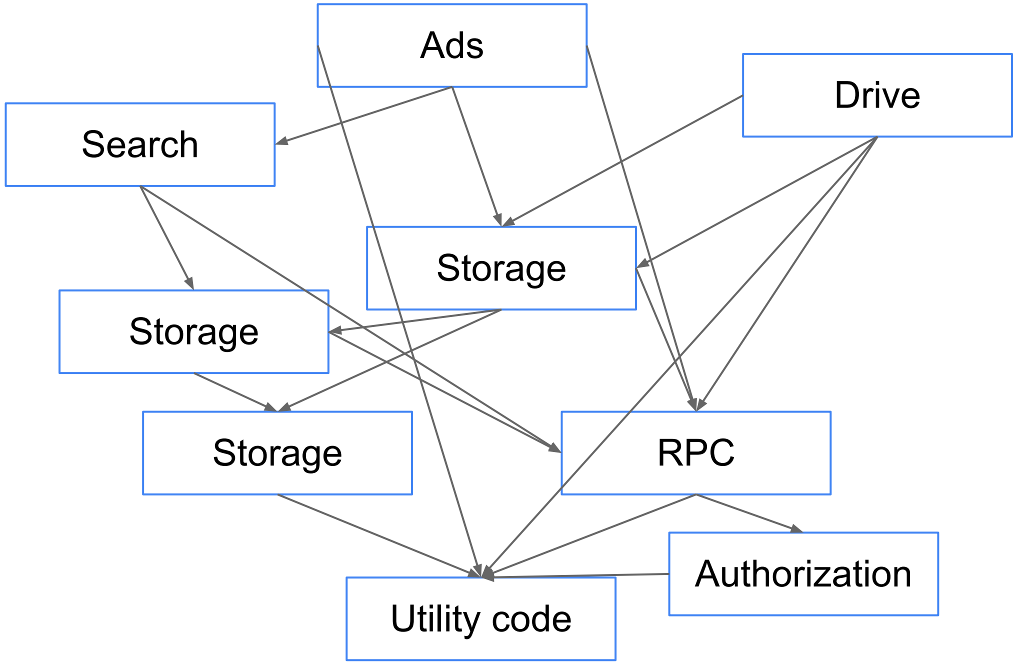
Monolithic Codebase



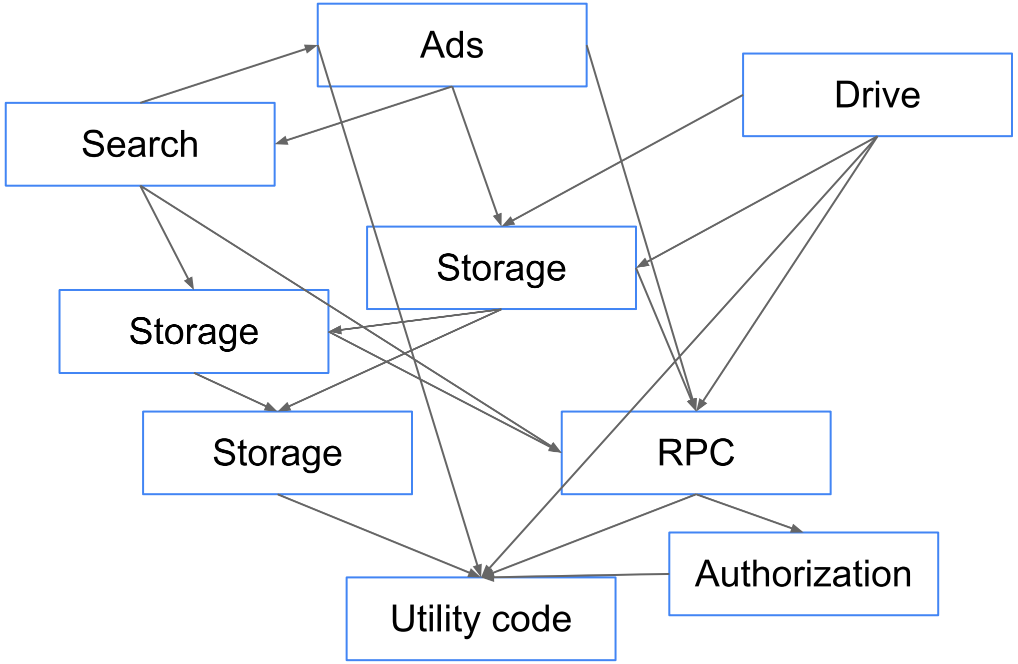
Monolithic Codebase



Monolithic Codebase



Monolithic Codebase



Sustainability

What does sustainability **mean**? Should you care?

- What’s the expected lifespan of your code?
- Is your code per-project or monolithic?

Per-project means you’ve got a big-red-button option. Monolithic means you **have** to get it right.

Change

Change is usually inevitable.

- Tooling changes
- Bugs will rise
- New attacks will emerge

Or, put another way: completely static infrastructure is usually unachievable in the long term.

Change is Inevitable

If a change is necessary in the lifespan of your project, do you have a plan to tackle it?

Is it **possible** to make changes of that sort safely?

Do you have **practice** making changes of that sort?

Sustainability

Your organization’s codebase is **sustainable** when you are **able** to change all of the things that you ought to change, safely, and can do so for the lifetime of your codebase.

(Whether you choose to do so is your business.)

Organizational Features

To be sustainable, you need some or all of:

- Testing (is this change safe)
- Policies (mitigate the crazy)
- Technology (automation and awareness)
- Practice, practice, practice

Google

We need to change hashing

(for example)

Testing

You need a way to be sure that your hashing change is safe. What could go wrong?

- **You need a culture of testing**

Testing Culture

Tests are your first line of defense in enforcing the contracts of your code and the code you depend upon.

This is not about “I’m good at coding, I don’t need to write tests.” This is about “The team who writes the code I depend upon sometimes makes mistakes.”

This web of trust-by-verification is **not** a proof that the codebase works.

Testing Culture

How to get there:

- No more changes without a test.
- No bugs can be marked as FIXED without a test that demonstrates the bug is indeed fixed (previously failed, now passes).

You have to hold this line. It is at most a short-term harm to velocity.

Testing

You need a way to be sure that your hashing change is safe. What could go wrong?

- You need a culture of testing
- **You need test infrastructure: If it’s hard to run tests, nobody will do it, and then nobody will write tests.**

Testing Infrastructure

Compare:

- How do I test this? Let me check the README. And install that thing. And run the test. Wait, it doesn't pass. Was that my change? Or did I install it wrong?
- `bazel test :all`

It has to be quick, reliable, and consistent across the organization.

Testing

You need a way to be sure that your hashing change is safe. What could go wrong?

- You need a culture of testing
- You need test infrastructure: If it’s hard to run tests, nobody will do it, and then nobody will write tests.
- **You need policies: If our change breaks you and you have no tests, not our fault**

Google

Policies

You need ways to guide the codebase. What if everyone writes their own hash?

- Style guides - strongly encourage consistency and safety
- Code review, and take it seriously - encourage sane code
- Best practices - lightly encouraged guidance
- Readability - Require responsible supervision, mentorship
- Churn policies - encourage responsible infrastructure change
- Large scale changes policies - Gate changes that touch many files

Technology

Where do you invest to make long-term sustainability possible?

- Distributed build/test (try bazel)
- Common test frameworks (try gtest)
- Tests are built into the build graph (bazel)
- How to generate a change at scale (clang-mr)

Technology

Where do you invest to make long-term sustainability possible?

- Codebase understanding
  - Indexing (kythe)
  - codesearch

Technology

Where do you invest to make long-term sustainability possible?

- Codebase understanding
  - Indexing (kythe)
  - codesearch
- Tooling for bug reduction
  - continuous tests (with the Sanitizers)
  - static annotations (turned on by default)
  - clang-tidy

Technology

Static annotations for thread safety are a big win.

Technology

Static annotations for thread safety are a big win.

```
class Foo {
 public:
  Foo();
  void Update(int n) LOCKS_EXCLUDED(lock_);

 private:
  Mutex lock_;
  int data_ GUARDED_BY(lock_);

  void UpdateInternal(int) EXCLUSIVE_LOCKS_REQUIRED(lock_);
};
```

Technology

Static annotations for thread safety are a big win.

```
class Foo {
 public:
  Foo();
  void Update(int n) LOCKS_EXCLUDED(lock_);

 private:
  Mutex lock_;
  int data_ GUARDED_BY(lock_);

  void UpdateInternal(int) EXCLUSIVE_LOCKS_REQUIRED(lock_);
};
```

Technology

Static annotations for thread safety are a big win.

```
class Foo {
 public:
  Foo();
  void Update(int n) LOCKS_EXCLUDED(lock_);

 private:
  Mutex lock_;
  int data_ GUARDED_BY(lock_);

  void UpdateInternal(int) EXCLUSIVE_LOCKS_REQUIRED(lock_);
};
```

Technology

Static annotations for thread safety are a big win.

```
class Foo {
 public:
  Foo();
  void Update(int n) LOCKS_EXCLUDED(lock_);

 private:
  Mutex lock_;
  int data_ GUARDED_BY(lock_);

  void UpdateInternal(int) EXCLUSIVE_LOCKS_REQUIRED(lock_);
};
```

Static annotations: Thread safety enforced by the compiler.

Technology

Raise your hand when you spot the bug.

```
void Foo::Update(int n) {
  lock_.Lock();
  if (n == data_) return;  // Skip it, already done.
  data_ = n;
  lock_.Unlock();
}
```

Technology

We use `clang-tidy` for static analysis during code review (see also: Tricorder).

- Automatic compiler-based code understanding
- Antipattern recognition **with suggested fixes**

Technology

We use `clang-tidy` for static analysis during code review.

```
std::vector<std::string> names = GetNames();
...
for (const std::string name : names) {
  Process(name);
}
```

Technology

We use `clang-tidy` for static analysis during code review.

```
std::vector<std::string> names = GetNames();
...
for (const std::string name : names) {
  Process(name);
}
```

ClangTidy: the loop variable’s type is const qualified but not a reference type. This creates a copy in each iteration. Consider making this a reference

Google

Practice

When change is needed, do you have practice making it happen? Or are you making it up while the crisis boils over?

Lessons will be learned and expertise will be gained.

Organizational Features, Revisited

To be sustainable, you need:

- Testing (is this change safe)
- Policies (mitigate the crazy)
- Technology (automation and awareness)
- Practice, practice, practice

Summary

Over a long enough period, change is inevitable.

If the set of important changes you are facing is only growing, you will eventually fail.

Technology, coding standards, organizational policies, and technology can make it easier (possible) to execute those changes safely.

Practice makes perfect.

Questions?

A brief history

Several iterations on a unified codebase in the early days.

This iteration really began in 2003.

Starting in mid 2000s, test culture.

These days: 5K programmers make a C++ change every week.

100M+ lines of C++ code.

Monolithic.