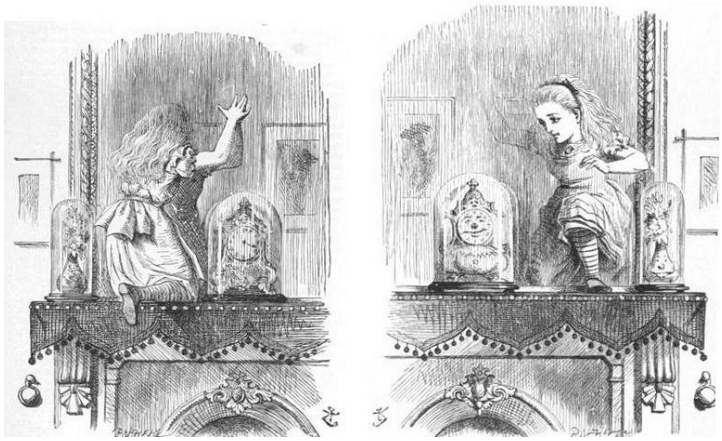


Eclipse in the Enterprise

Lessons from Google

Terry Parker
Robert Konigsberg
August 27, 2009

Google™

As a developer at Google, You Have a Lot of Choices.

At Google you are fairly unrestricted by which developer tool you can use. vi, emacs, eclipse, intellij, pico.

Some of our engineers use Eclipse.

Similarly, engineers can install any plug-ins they want. Eclemma? Findbugs. Fine. AnyEdit? Go ahead.

But Our Engineers Are ALSO REALLY BUSY.

So We Want Them To Be Efficient Quickly.

Once you have all these nice plug-ins, and a nice content-assistance-friendly UI, how do you plug that in to our HOME GROWN build system?

Our team is responsible for providing that integration.

Supporting Our Users

Google™

Since alot of our users are not power users, we have to be the front line to Eclipse, since users are not power users.

Our users rely on us to be the primary support framework. Suggesting people go to the IRC channel, join the Eclipse newsgroups or submit their own bugs is, for the average Google Eclipse user, not necessarily going to happen.

Our users are often not power users.

History

The first community tool we know of was written in 2004. It was a 95-line python script that generated a .classpath file from a precompiled build target.

Most everything else was done manually -- including regeneration.

No support for launching, tests, etcetera.

Support and upgrades to Eclipse was managed loosely a community of dedicated users on their 20% time.

What we do

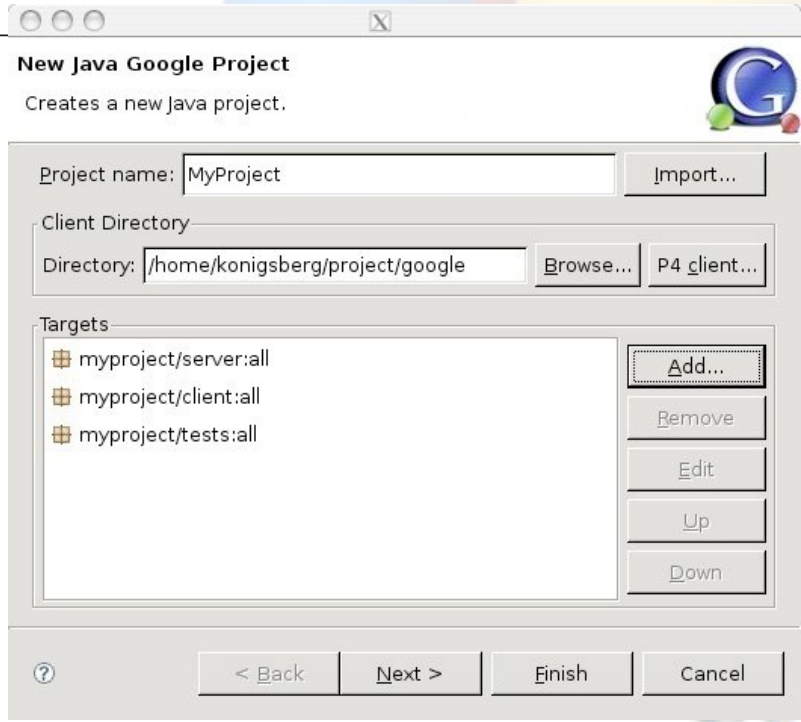
Google™

So what we build:

We build a set of plug-ins that ties Eclipse to our build system.

Here's a little bit about how it works.

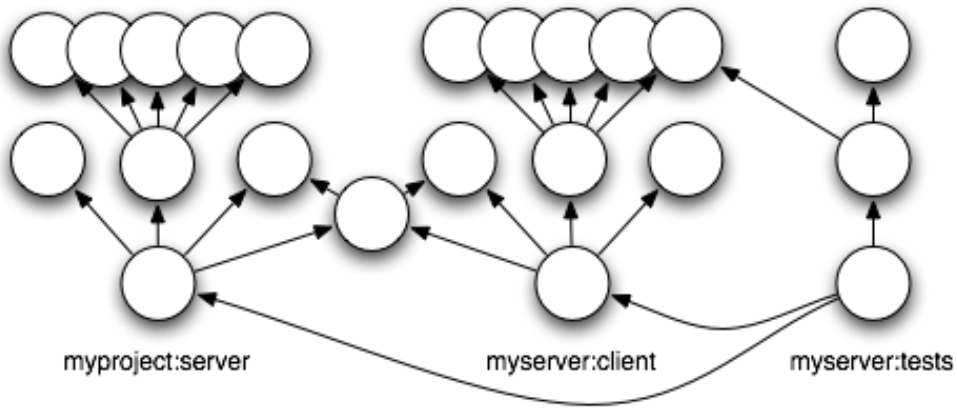
Creating a Guide Project



To Create A Guide Project

Specify The Top Level Component Targets You're Interested In Building.

Creating a Guide Project



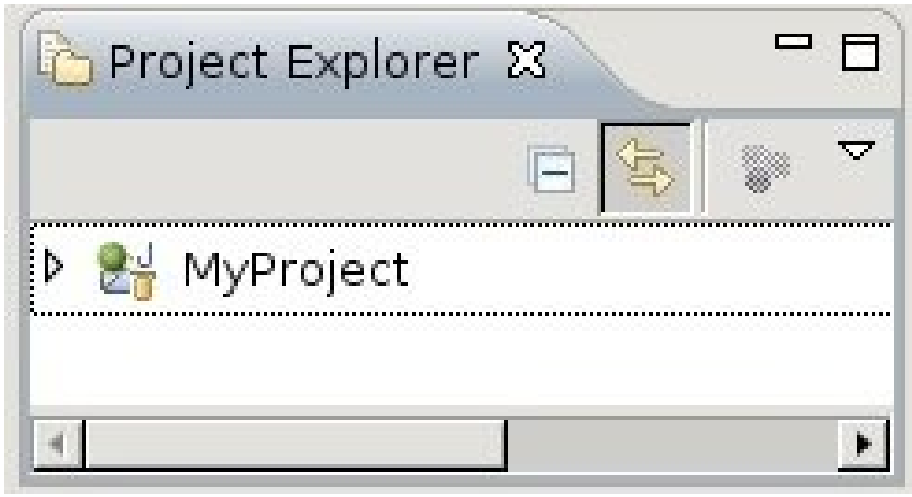
Google™

Guide Will Talk To Our

Build System And

Fetch All Required Subcomponents... <NEXT>

Creating a Guide Project



Google

And Create A New Project

With A Workspace Full Of Files

And Configured To Work.

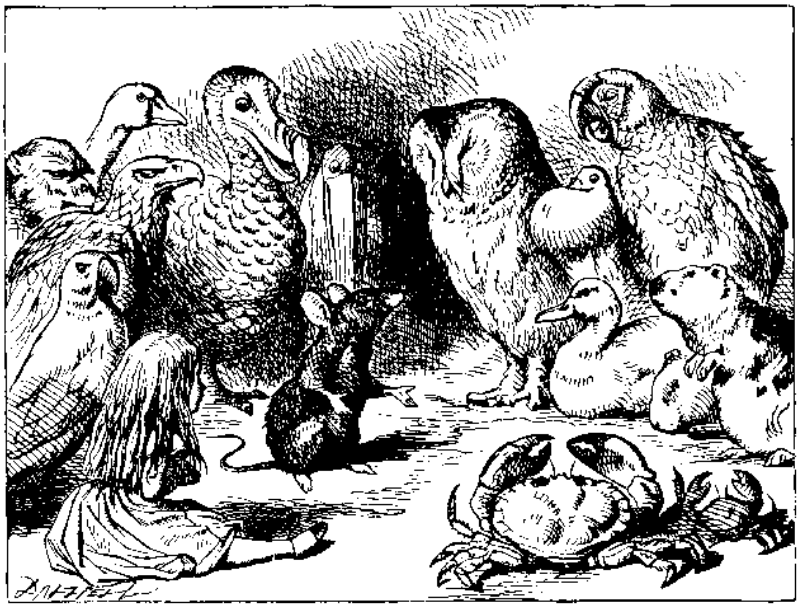
The End Result

- Optimized Workspace Settings
  - Configurable Workspace Settings
- Project support
- Builder integration
- Editors
- Launching

- A Workspace Initialized With Common Plug-ins, Editors Formatted For Coding Standards.
- Integration with JDT, CDT and PYDEV
-
- Eclipse Builder integration with our build system.
-
- Custom editors for several internal languages.
-

Custom Launch Configuration Types Optimized For Our Build System.

Custom Editors



Google™

A Quick Note About Editors

One Easy Way To Contribute To Eclipse At Google Is By Writing An Editor Most Of Our Editors Started As 20% Contributions.

(Who Here Is Familiar With Protocol Buffers?)

Protocol Buffers Is An Open Sourced Technology. We Have An Editor For That.

Developing at Google

The Google environment is

- Thousands of developers
- A massive code base (it's ginormous!)
- Lots of languages
- Hundreds of tools
- Constant change in every dimension

Google engineers build a _lot_ of applications.

They build Ad Words. They build Google Spreadsheets. They build Google Analytics. They build all sorts of infrastructure that's used by web search, maps, and so on.

How does Google do it?
Lots of engineers
Lots of code
Lots of tools

As a Google engineer you are dealing with a massive amount of information, and a high rate of change.

Google Engineers & Eclipse

Our unique environment is a challenge

- Refreshes hang
- Waves of unnecessary/blocking builds
- Mondo classpath management
- The Google test framework stricter than Eclipse/JUnit
- (Until recently) updates to plugins break your project

It requires constant tuning to bridge Eclipse to the Google environment

Google™

Due to our unique environment, Googlers can experience these types of problems when using Eclipse

Fortunately, Eclipse is an open source, extensible environment, and our team can fix these things.

How to Piss off Everyone (aka, Deployment)

It's *Easy*! Push a new version of the IDE!

Google

Deployment is *tough*

Engineers live in their IDEs.

And they are always up against deadlines or in the middle of a huge change.

Change is bad.

Except when they are blocked on that bugfix.

Change is good.

Deploying to a large audience is a challenge--need to balance:

1) the need for updates against need for stability
2) providing useful features vs. installation bloat
3) push vs. pull (forced updates vs. multi version support issues)

A Modular Distribution (2008)

- Debian package-based distribution
- A "shared" installation
- Added new features through the "links" folder

In June 2008 our Eclipse installation looked like this:

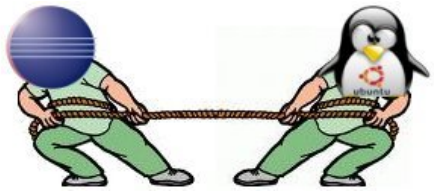
1) core package in read only partition
2) individual features through "llinks" folder

Worked pretty well
* provided a common installation for our team to support
* modular updates
* reduced user configuration problems (Mitigated that Eclipse left configuration management up to individual users.)

But
* debian package updates removed old plugins, causing running instances to crash
* single monolithic installation, so beta testing was tough
* harder to customize

And then...

Ganymede Broke our Distribution



Actually, it filled the void we had already filled

- With p2, Eclipse takes ownership of the plugins
  - Debian packages change bits out from under p2
  - p2's caching/metadata gets hopelessly corrupted

Google™

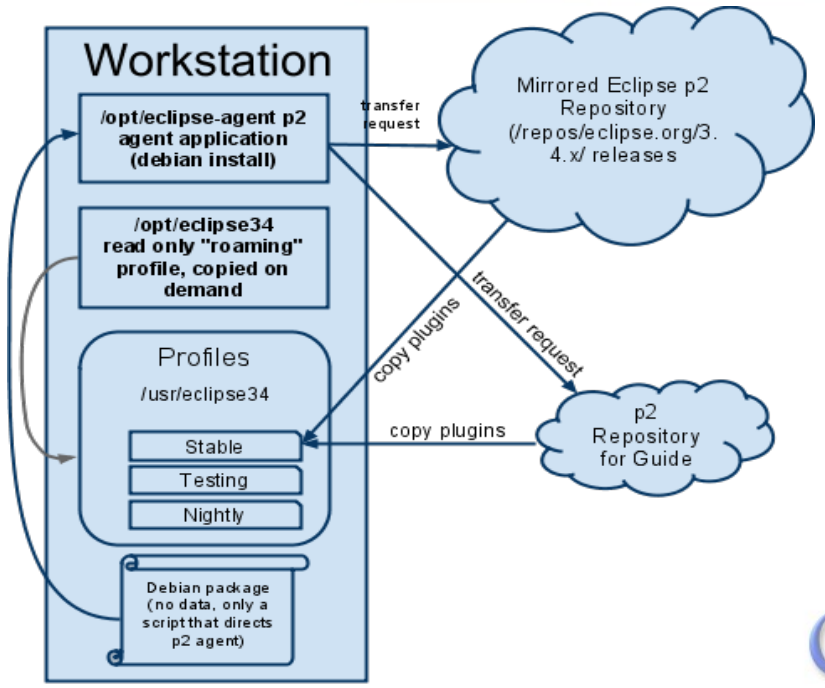
I _really_ like p2.
(DEFINE P2)

But this is what it felt like

p2 filled a void

p2 "if you can install it, it will run"

Solution: Delegate Installation to p2



Google™

The solution is to embrace p2

This works REALLY well!

1) Avoids corruption
2) Very easy for users to try new things

We install 2 RO components via debian packages, but delegate all customizations of user profiles to p2.

Benefits:
* Leaner/faster distribution
* Corruption issues gone
* Support for beta testers in a separate profile
* Much easier for users to customize/try new things

Scalability

From: Jane Googler
To: google-eclipse-users
Title: Eclipse freezes when...

Google projects involve a **lot** of code

- Refreshes that never return
- Indexing Java files
- Indexing C++ files
- (was an issue for Python, but Radim fixed it)
- IDE responsiveness

Google

Google projects are very large.

This is a problem of our own making.

Users expect to have all the source available to them, even if not modifying it.

Mental model is of a single, large project.

Problem: Workspace refreshes are _not_ lazy.

Point Eclipse at "/" and it keels over.

Map a Subset of Source

Not every file needs to be editable, but it should be debuggable

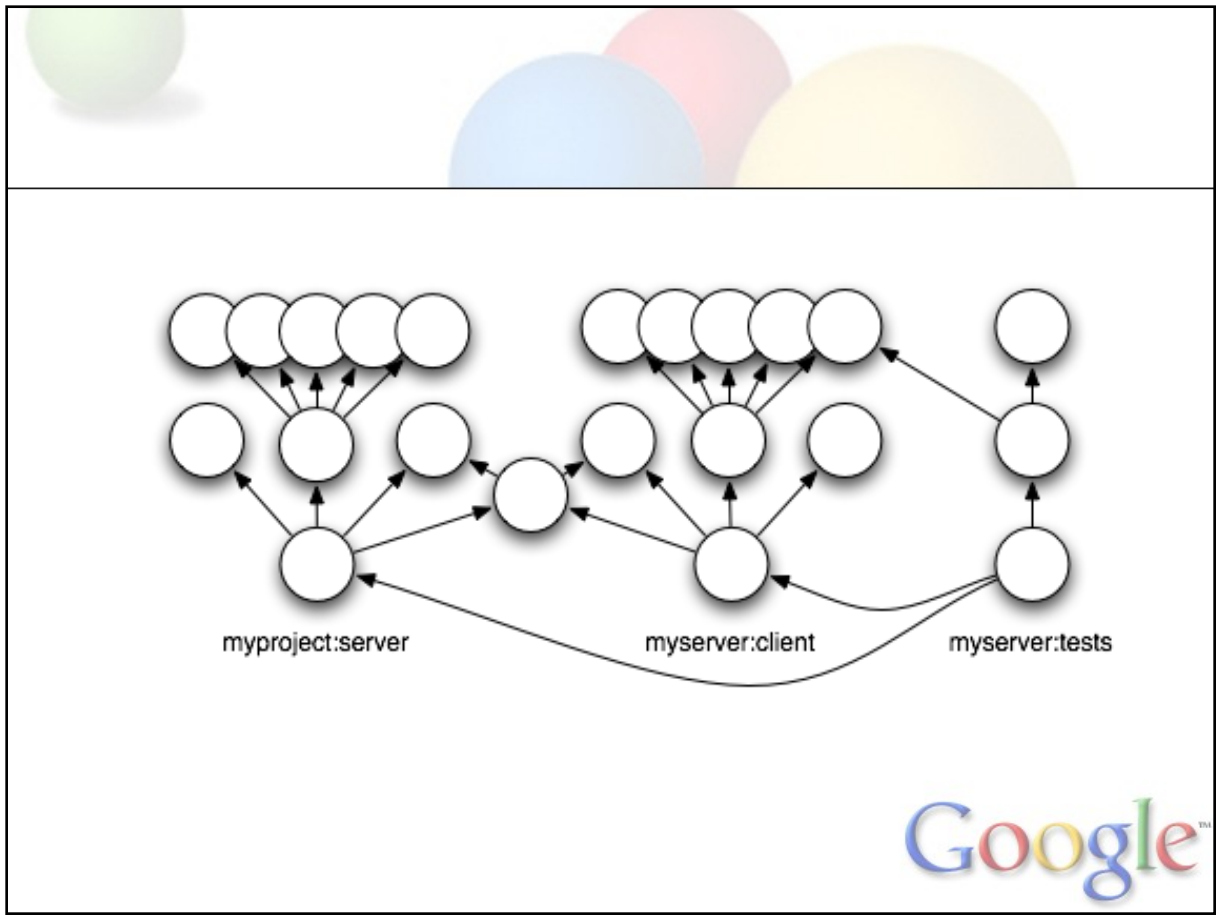
- For Java set up backing jars
- For C++
  - set up GDB to discover files not mapped in Eclipse
  - set up CDT indexer to discover external headers

Your project may depend on a hundreds of other libraries

You don't need to edit that code in Eclipse

You do want to navigate and debug it.

You want code completion.

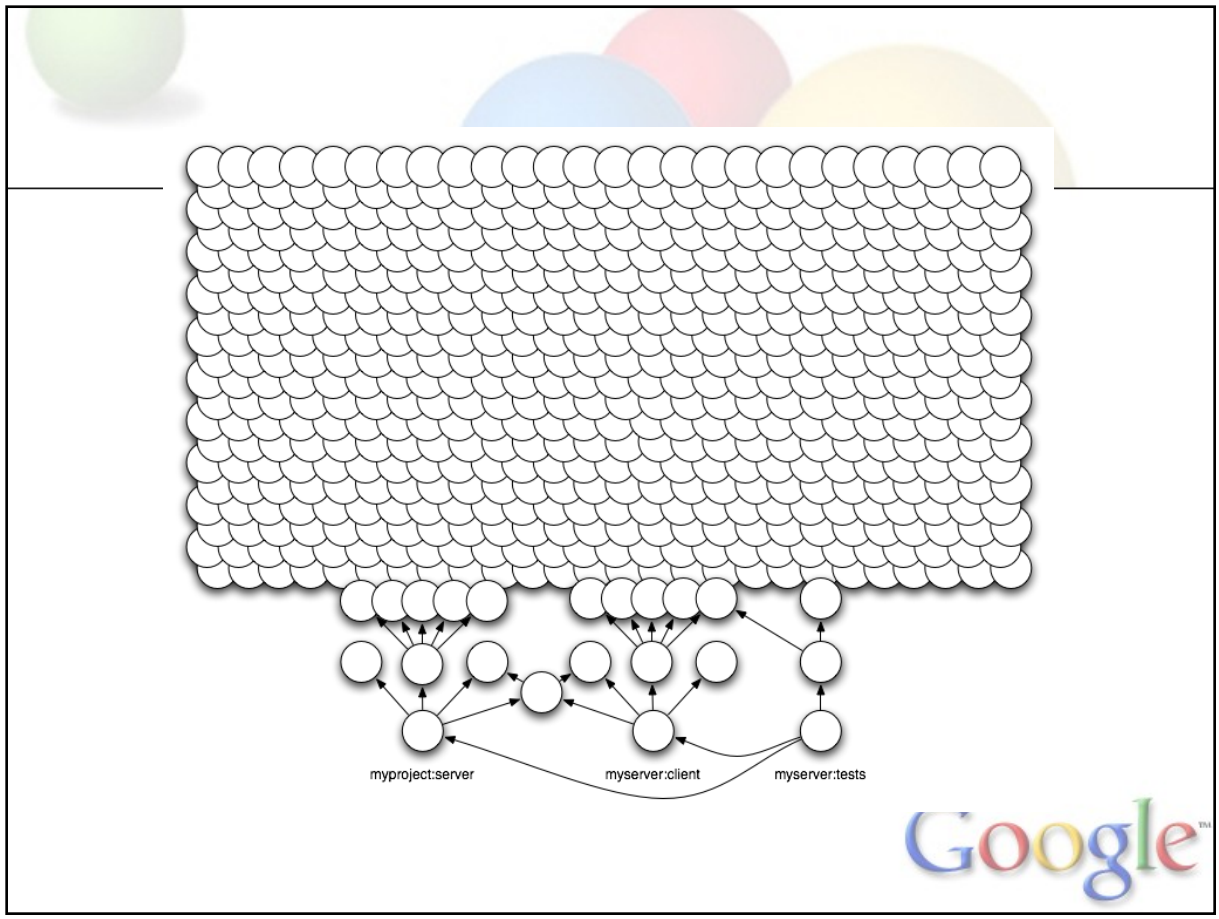


You May Recall

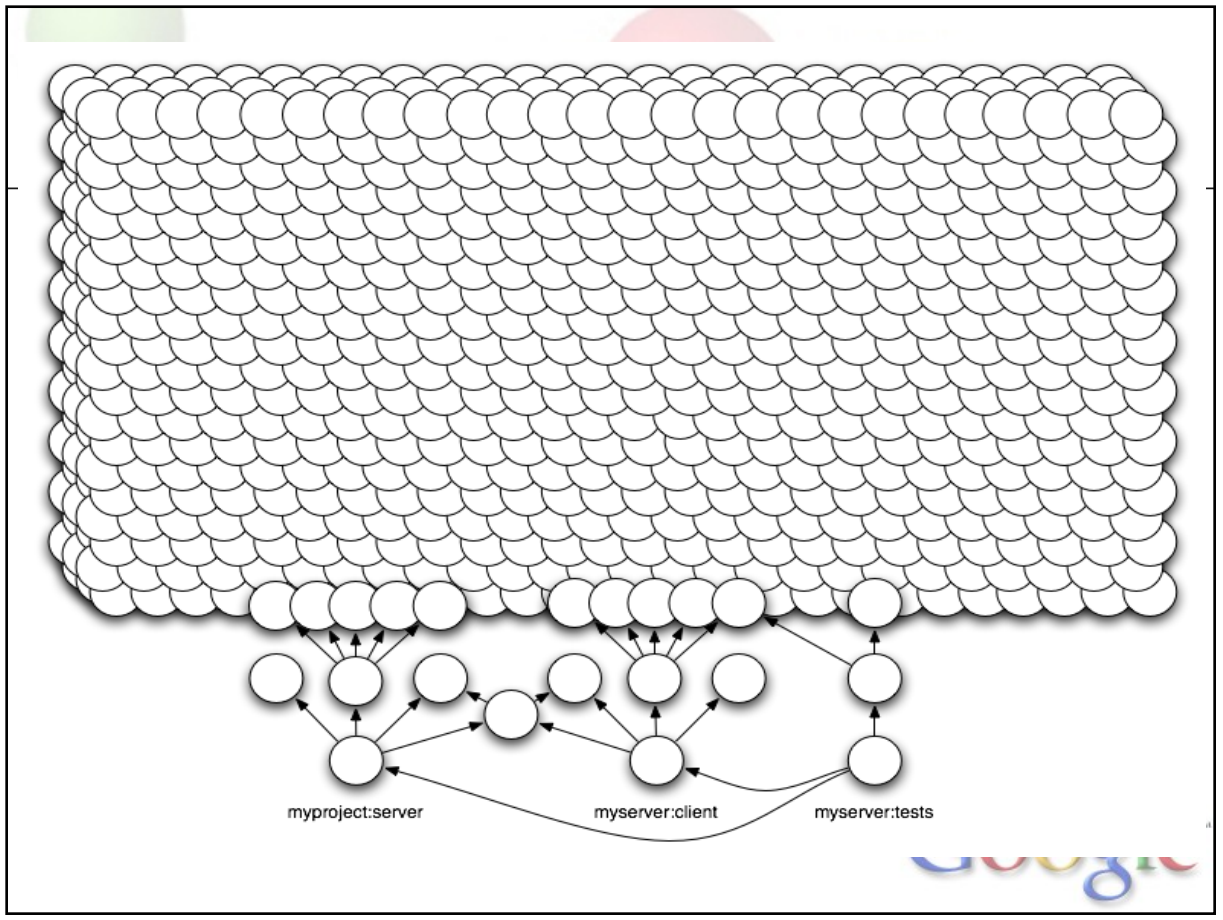
This Is An Example Set Of Components And Subcomponents

DEFINE BACKING JAR

But Really It Looks Like ...



This. Though Really It Is...



This!

If Each Of These Is A Jar, What Could Go Wrong?

Math!

- 2,200 jar files
- n^2 algorithm
- new Path()
- 500,000 per second
- n = 2,200, n^2 = 4,840,000
- Round to 5,000,000
- 10 seconds!

Google™

An Engineer Reported Big Delays Stepping Through Debugger

Analysis Showed The Majority Of Time In Associated Source Lookup.

Path is the Java.Io.File of Eclipse.

WE DID FIX THIS PROBLEM

Easy Solution

```
IPackageFragmentRoot[] allRoots = ...
for (int j = 0; j < allRoots.length; j++) {
  IPackageFragmentRoot root = allRoots[j];
  if (... root.getPath().equals(
      new Path(entry.getLocation()))) {
    ...
  }
}
```

Easy Solution

```
Path entryPath = new Path(entry.getLocation());
IPackageFragmentRoot[] allRoots = ...
for (int j = 0; j < allRoots.length; j++) {
  IPackageFragmentRoot root = allRoots[j];
  if (... root.getPath().equals(entryPath)) {
    ...
  }
```

Not Necessarily In Inner Loop.

DEPLOYMENT INTERNAL

INTERNAL GIT REPO

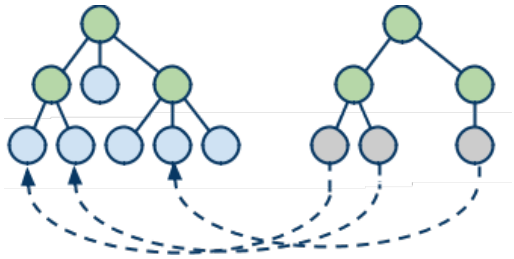
Scaling the Eclipse File System

Approaches with bad side effects:

Symlink forest (slow and inflexible)

Source Tree

Link Forest
(Eclipse points here)



Google

Talked about configuring Eclipse to deal with "external" files, but how make them external?

How do you limit what files Eclipse sees? The source is actually there.

Greenish nodes are directories

Symlink forest mirrors directory nodes and has symlinks @ leaves.

#1) creating new files puts them in the symlink forest, not the actual source!

CLEAR?

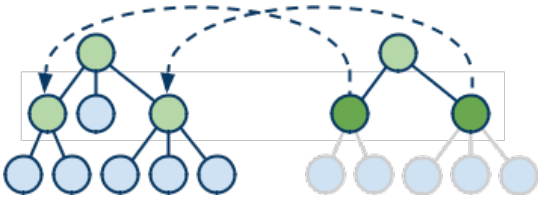
Scaling the Eclipse File System

Approaches with bad side effects:

Eclipse linked folders at leaf folders
(slow, buggy to add new folders)

Source Tree

Linked Resources
(Eclipse points here)



Google

#2) creating a new folder only locates it correctly for leaf nodes

Neither of these matches the user's mental model

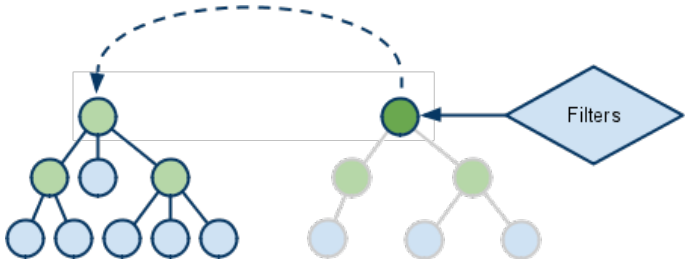
Both of these break new Java package creation and many refactorings

Scaling File System Solution

Implement an Eclipse File System (EFS) that filters files

Source Tree

Single Linked Resource at Root



Google™

DEFINE EFS
stateless

EFS solution is simple: implement 2 classes, delegate nearly all of the operations to the existing Local FS class, override 3 functions to handle filtering (childNames/fetchInfo/mkdir in IFileStore interface)

e4 will provide Flexible Resources (check performance and that filtering can be programatically set **pre-refresh**)
e4 solution will get backported to 3.6

Configuration

Eclipse is super-configurable!

This is great when you're a single user

Google™

Configuration

Eclipse is super-configurable!

This is "problematic" when you're supporting a corporate installation

Set up default preferences in config.ini
eclipse.pluginCustomization=<path>/plugin_customization.ini

Google™

I don't want Roberts's editor settings to completely reformat my file.

Support problems.
In past, Googlers imported preference files to get in sync.

Things you can sync through default preferences:
Code formatting
Debug settings
Java compiler Error/warnings settings
Visibility of markers
Web browser firewall/proxy settings
Auto update scheduling

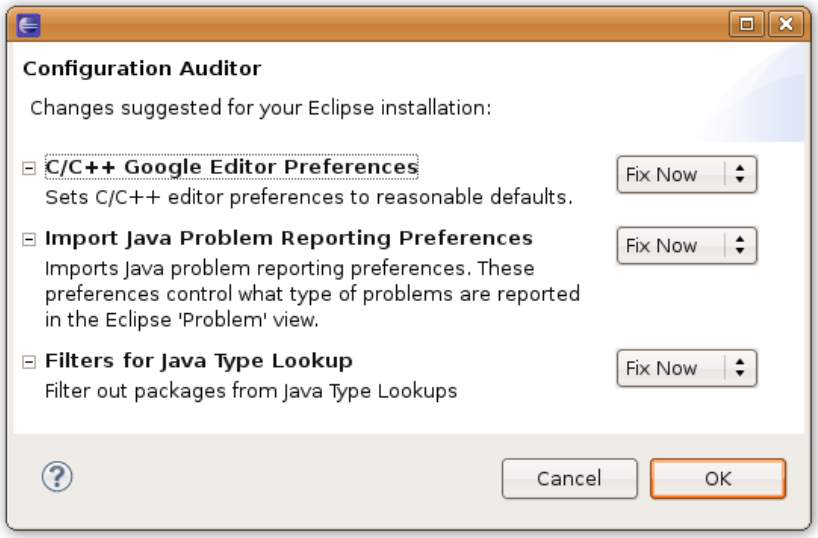
But what about:
Configure For Localized Team?
Configure Outside Of Preferences?
Have Multiple Workspaces Copies?

In fact I know we're not alone dealing with these kinds of problems.

Comes From Blog Post from June.

I think they're so desperate they'll take a dead one.

Google Configuration Auditor



Google™

Which sees your workspace and makes sure everything is in sync.

Auditor is an Eclipse plugin that

Runs As Startup And On Regular Intervals

- Gives the user a list of audits to run; user can delay or opt-out
- Applies the selected audits, updating workspace preferences

Uses for the Configuration Auditor:

- Team and personal preferences
- Complex changes, such as cleaning up files, injecting user's ldap, etc.
- Interim fix for default preferences

Audit

```
public interface Audit {
  String getId();
  String getTitle();
  String getDescription();
  Evaluator getEvaluator();
  RepairAction getRepairAction();
}
```

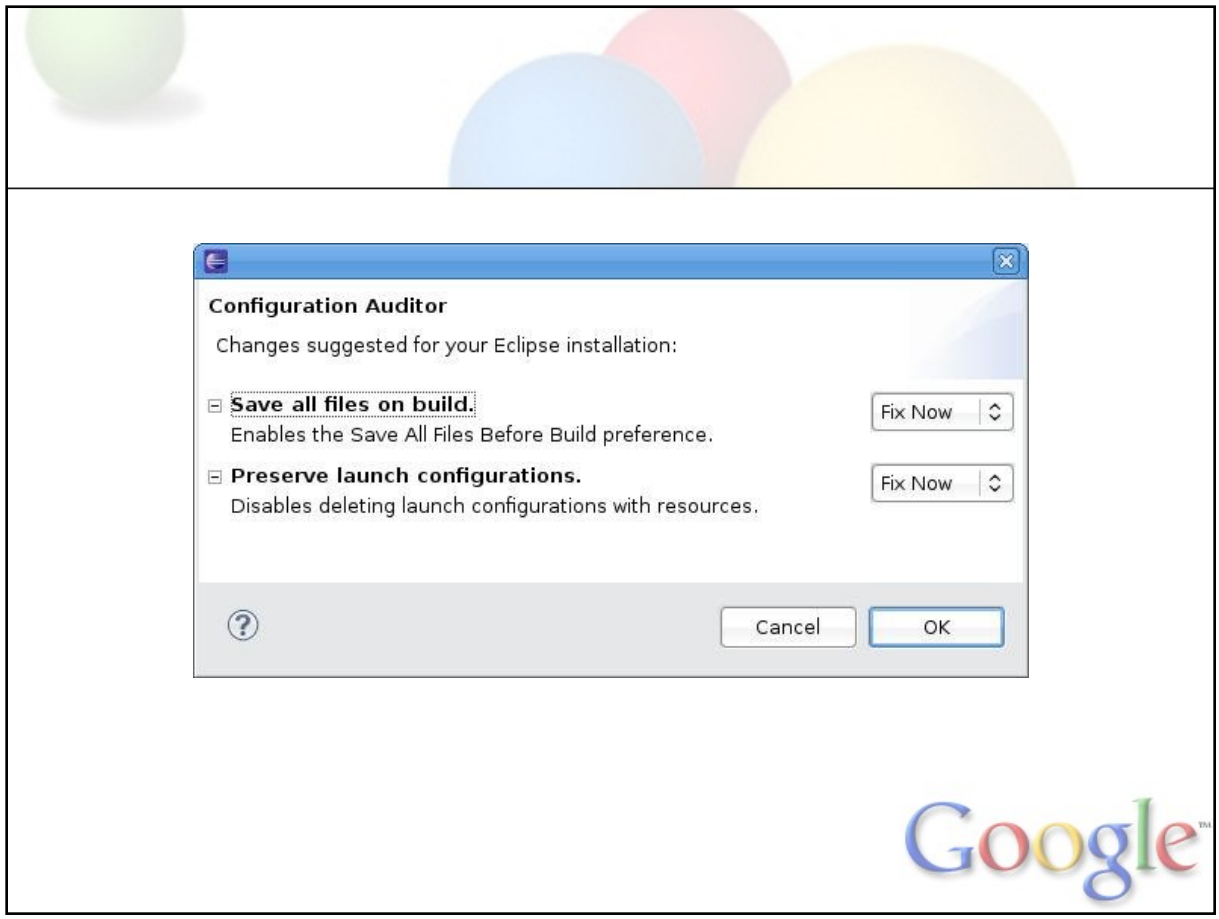
An audit contains two important operations: an evaluator and a repair action.

The evaluator looks at your environment and identifies if it's in sync with the audit's expectation.

The repair action changes the environment so the evaluator will pass.


Eclipse Notification
The Eclipse Configuration Auditor found audits that need your attention.
Repair failed audits
Disable this popup

Google



Discuss that audits can be declined. You don't have to accept them.

This behaves similarly to opt-in update mechanisms like Windows Update.

And audits don't need to just be system preference audits ; an audit can be implemented through glorified preference files, Java classes, Groovy scripts or extension points.

This Lets You Fundamentally Analyze And Repair Any Part Of Eclipse.

External Build Systems

For external build systems, Eclipse

- Triggers builds too frequently
- Assumes null builds are instantaneous (locks entire workspace)
- Loses information--was that build generated by a menu action?

Google™

Well known problem
Example, the CDT indexer indexes each file, it clears that file's markers, which triggers a refresh, which triggers an automatic build.

External Build Systems

Our approach

- Implement a custom builder
- Avoid duplicated work (e.g., Java compilation)
- Sense when builds are truly needed
- Provide an alternate "execute build" command and a setting that ignores Eclipse's internal triggering

Google™

The CDT has grappled with this problem for years. They add a project-level "disable build automatically" switch.

We haven't really solved it.

Summary

Google

1.Eclipse is a Good Choice for Integrating With Our Build System
2. We regularly hit some scalability and performance issues
3. Eclipse's extension mechanism is a key part of why we continue to use it.

Attend these talks if you are interested.

Questions.