A methology that allows for a single Java code base covering many different versions? [closed]

Asked 15 years, 4 months ago Modified 4 years, 8 months ago

Viewed 517 times

12

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 5 years ago.

Improve this question

I work in a small shop where we have a LOT of legacy Cobol code and where a methology has been adopted to allow us to minimize forking and branching as much as possible.

For a given release we have three levels:

- CORE - bottom layer, this code is common to all releases
- GROUP - optional code common to several customers.

- CUSTOMER - optional code specific for a single customer.

When a program is needed, it is first searched for in CUSTOMER, then in GROUP and finally in CORE. A given application for us invokes many programs which all are looked for in this sequence (think exe files and PATH under Windows).

We also have Java programs interacting with this legacy code, and as the core-group-customer lookup mehchanism does not lend it self easily to Java it has tended to grow in a CVS branch for each customer, requiring much too much maintainance. The Java part and the backend part tend to be developed in parallel.

I have been assigned to figure out a way to make the two worlds meet.

Essentially we want a Java enviornment which allows us to have a single code base with sources for each release, where we easily can select a group and a customer and work with the application as it goes for that customer, and then easily switch to another codeset and THAT customer.

I was thinking of perhaps a scenario with an Eclipse project for each core, customer, and group and then use Project Sets to select those we need for a given scenario. The problem I cannot get my head about, is how we would create robust code in the CORE projects which will work regardless of which group and customer is selected.

A Factory class which knows which sub class of a passed Class object to invoke instead of each and every new?

Others must have had similar code base management problems. Anybody with experiences to share?

EDIT: The conclusion to this problem above has been that CVS needs to be replaced with a source code management system better suited for dealing with many branches concurrently and the migration of source from one component to the other while keeping history. Inspired by the recent migration by slf4j and logback we are currently looking at git as it handles branches very well. We've considered subversion and mercurial too but git appears to be better for single location, multibranched projects. I've asked about Perforce in another question, but my personal inclination is towards open source solutions for something as crucial as this.

EDIT: After some more pondering, we've found that our actual pain point is that we use branches in CVS, and that branches in CVS are the easiest to work with if you branch ALL files! The revised conclusion is that we *can* do this with CVS alone, by switching to a forest of java projects, each corresponding to one of the levels above, and use the Eclipse build paths to tie them together so each CUSTOMER version pulls in the appropriate GROUP and CORE project. We still want to switch to a better versioning system but this is so important a decision so we want to delay it as much as possible.

EDIT: I now have a proof-of-concept implementation of the CORE-GROUP-CUSTOMER concept using Google Guice 2.0 - the @ImplementedBy tag is just what we need. I wonder what everybody else does? Using if's all over the place?

EDIT: Now I also need this functionality for web applications. Guice was until the JSR-330 is in place. Anybody with versioning experience?

EDIT: JSR-330/299 is now in place with the JEE6 reference implementation Weld based on JBoss Seam and I have reimplemented the proof-of-concept with Weld and can see that if we use @Alternative along with ... in beans.xml we can get the behaviour we desire. I.e. provide a new implementation for a given functionality in CORE without changing a bit in the CORE jars. Initial reading up on the Servlet 3.0 specification indicates that it may support the same functionality for web application *resources* (not code). We will now do initial testing on the real application.

java **project-management** **dependency-injection** **java-ee-6**

edited Apr 17, 2010 at 16:06

asked Aug 17, 2009 at 13:55

Thorbjørn Ravn Andersen

75.3k ● 34 ● 199 ● 352

Cletus, heh, so I just list all the jobs assigned to me, and instant 100k karma? – Thorbjørn Ravn Andersen Aug 17, 2009 at 19:39

If they're as icky and horrifying as this job, sure. :) – cletus Aug 18, 2009 at 9:10

Cletus, while editing I thought: How do you/your employer cope with the "each customer needs his version" of a product, with common code and customer specific code? – Thorbjørn Ravn Andersen Apr 17, 2010 at 16:09

2 Answers

Sorted by: Highest score (default)

1

I apologize in advance for not answering your question directly. You're asking "how can we engineer our software to avoid having to do a certain type of release engineering? I don't know the answer to that question, but I read your description, and I'm worried about it. At its root, I think it's a losing game to make your core software engineering needs subservient to your release engineering technologies.

In other words, if your customers really do need separate releases, then you're going to have to deal with that. What I read you saying is that you want to use Eclipse to

do the work that your version control software should be doing for you. That doesn't strike me as a viable solution.

I bet you can see where I'm going with this. If the problem is actually "branching in CVS is too painful," then I think the technical solution is "migrate away to CVS to something that allows more robust and easier branching and integration."

In the free software world, that usually means Subversion. A good commercial alternative is Perforce. I can say that I worked for a company which released multiple verisons of software to many customers, and often cross-integrated changes from one branch to another. We used Perforce, and branching (and integrating) was easy enough that any engineer could and did do it on a regular basis without disruption to their daily work.

Hope that helps.

edited Apr 9, 2020 at 21:11

answered Aug 17, 2009 at 14:42

peterb
1,368 ● 9 ● 14

You might be right in your observation. We branch whole projects which is too rigid for our needs, so we need a source repository suitable for us. CVS has been "good enough" for a

long time, so I was not thinking outside that box.
– Thorbjørn Ravn Andersen Aug 17, 2009 at 19:36

Chosen as answer as you accurately point out that the pain point is the source management system.
– Thorbjørn Ravn Andersen Sep 19, 2009 at 8:55

1

Have to agree with @peterb, Eclipse is not the way to manage this. Branching in your source control system is meant to serve this purpose, so choose one that makes this as easy as possible (Subversion should work fine).

You may also want to consider using Maven (or something similar) to then define your dependency management for your different customer scenarios. Then you can generate both your eclipse projects from maven to do your actual development, as well as the release builds.

Share Improve this answer

Follow

answered Aug 17, 2009 at 14:51

Robin
24.3k ● 5 ● 52 ● 58

I had been thinking of "mvn eclipse:eclipse" but we would very much like to stay away from maven, as we do not agree with it upon how things should be done :)
– Thorbjørn Ravn Andersen Aug 17, 2009 at 19:38