# How do you version your projects and manage releases?

Asked 16 years, 2 months ago Modified 16 years, 2 months ago

Viewed 4k times

8

Our situation is as follows, but I'm curious about this problem in any situation.

We have a framework consisting of 4 projects:

- beans
- util
- framework
- web

We also have modules that need a version and depend on a version of beans and util.

Finally we have a customer project that consists of a specific version of the core projects and one or more modules.

Is there a standard way to version these projects?

What seems simple to me is becoming really complicated as we try to deliver releases to QA and then manage our ongoing development with the maintenance of the release (release = tag and possible branch).

I kind of prefer the following:

1.2.0 - major and minor versions + release.

1.2.1 - next release

1.2.0_01 - bug fix in 1.2.0 release (branch)

etc.

Any ideas?

java language-agnostic maven-2 versioning

Share

Improve this question

Follow

edited Oct 10, 2008 at 14:03

asked Oct 10, 2008 at 13:56

ScArcher2
87.2k ● 47 ● 122 ● 160

## 9 Answers

Sorted by: Highest score (default)

10

We use major.minor.bugfix. A major release only happens for huge changes. A minor release is called for when there is an API change. All other releases are bugfix releases. There's definitely utility in having a build or revision number there too for troubleshooting, although if you've got really rigorous CM you might not need to include it.

Coordinating among the versions of all these projects can be done really well with help from tools like Apache Ivy or Maven. The build of one project, with its own version number, can involve the aggregation of specific versions of (the products of) other projects, and so your build files provide a strict mapping of versions from the bottom up. Save it all in [insert favorite version control tool here] and you have a nice history recorded.

Share Improve this answer Follow

answered Oct 10, 2008 at 17:09

bhavanki
1,527 ● 10 ● 16

One place I worked that did major-minor-bugfix was really serious about only incrementing the major for huge things. They had existed for 15 years and I was working on the latest and greatest version: 1.52.0 :) – tloach Oct 10, 2008 at 17:54

3

I use {major}.{minor}.{buildday}.{sequential}. For Windows, we use the utilities [stampver.exe]() and [UpdateVersion.exe]() for .NET projects that handle that mostly automatically.

Share Improve this answer Follow

edited Oct 14, 2008 at 19:45

Paul Dixon
300k ● 54 ● 314 ● 349

answered Oct 10, 2008 at 14:36

kenny
22.3k ● 8 ● 51 ● 87

2

There are no standard version number systems. Common themes are to have a major, minor and build number, and occasionally a point number as well (1.2.2.1 for example, for version 1.2 point release 2 build 1). The meaning of the version numbers is highly flexible. A frequent choice is to have backwards compatibility between minor versions or point releases though.

Releases are probably best done by labeling a set of source controlled files as long as your source control allows this. Recreating a release is then as simple as syncing to the label and building, which is very useful :)

Share Improve this answer Follow

answered Oct 10, 2008 at 14:05

workmad3
25.6k ● 4 ● 37 ● 56

2

In the automated build system i'm currently using I version with Major.Minor.Build.X, where Build is every time we hit system test, and X is the last Subversion revision number from the repo the code is being built from. Seems to work quite nicely for Subversion as we can easily get back to the codebase of a particular build if the need arises.

Share Improve this answer Follow

answered Oct 10, 2008 at 14:11

Steven Robbins
26.6k ● 7 ● 78 ● 90

2

I use a variation on the linux kernel version numbering system:

major.minor.bugfix

where even minor numbers indicate a somewhat stable release that may be distributed at least for testing, and odd minor numbers indicate an unstable/untested release that shouldn't be distributed beyond developers.

Share Improve this answer Follow

answered Oct 10, 2008 at 17:50

Draemon
34.7k ● 16 ● 79 ● 106

1

Where possible, I prefer to have projects versioned with the same build numbering, unless they are shared. It allows for more consistency between moving parts and it's easier to identify which components constitute a product release.

As workmad3 has stated, there's really no common rule for build numbers. My advice is to use something that makes sense for your team/company.

Some places I've worked at have aligned build numbering with project milestones and iterations,
e.g: Major = Release or Milestone, Minor = Iteration, Build = Build number (from the project start or from the start of

iteration), Revision = If the build has to be rebuilt (or branched).

Share Improve this answer Follow

answered Oct 10, 2008 at 14:15

RobS
9,422 ● 3 ● 37 ● 63

---

1

One of the most common conventions is major.minor.bugfix, with an additional suffix indicating a build number or pre-release designation (e.g. alpha, beta, etc.).

My team numbers builds according to project milestones - a build is handed over to our QA group at the end of a development iteration (every few weeks). Interim CI builds are not numbered; because we use Maven, those builds are numbered with a SNAPSHOT suffix.

Whatever you decide, be sure to document it and make sure that everyone understands it. I also suggest you document and consistently apply the release branching policy or it can quickly get confusing for everyone. Although with only 4 projects it should be pretty easy to keep track of what's going on.

Share Improve this answer Follow

answered Oct 16, 2008 at 3:21

Ken Liu
22.9k ● 19 ● 80 ● 98

---

Awesome, we use maven as well. It sounds like our plan is similar to what you said. I completely agree that making everyone else understand the process is key. – ScArcher2 Oct 16, 2008 at 14:48

0

You didn't mention if any of the projects access a database, but if any do, that might be another factor to consider. We use a major.minor.bugfix.buildnumber scheme similar to others described in answers to this question, with approximately the same logic, but with the added requirement that any database schema changes require at least a minor increment. This also provides a naming scheme for your database schemas. For example, versions 1.2.3 and 1.2.4 can both run against the "1.2" database schema, but version 1.3.0 requires the "1.3" database schema.

Share Improve this answer Follow

answered Oct 10, 2008 at 17:39

Eric R. Rath
1,979 ● 1 ● 14 ● 17

We're using autopatch. It will automatically keep the database up to date when new builds are deployed. (keeps track of them in a patches table) It's pretty nice. – ScArcher2 Nov 6, 2008 at 16:40

-1

Currently we have no real versioning. We use the svn build number and the release date. (tag name is like release_081010_microsoft e.g.)

Older Products use major.minor.sub version numbering

Major never changed Minor changes on every release/featurerelease every 6 months. Sub is everything which doesn't affect the feature set - mostly bugfixes.

Share Improve this answer Follow

answered Oct 10, 2008 at 17:47

Ronny Brendel
4,835 ● 5 ● 37 ● 55