

Apache Archive: Rules for Revolutionaries

by James Duncan Davidson

NOTICE: This document is a WIP (Work In Progress).

Archival copy of Duncan's 'Rules for Revolutionaries.'

1. rules for revolutionaries

Note:

As Sent on 13 Jan 2000. 1995-2002, James Duncan Davidson.

Note: This message was written at a specific place and time in response to a specific situation. It does not address several important issues like what happens when revolutions do not totally succeed. It also doesn't address the issues of responsibility. The Apache Jakarta PMC has responsibilities that aren't detailed here. With that said...

Date: Thu, 13 Jan 2000 15:46:41 -0800
Subject: RESET: Proposal for Revolutionaries and Evolutionaries
From: James Duncan Davidson <james.davidson@eng.sun.com>
To: <general@jakarta.apache.org>
CC: <tomcat-dev@jakarta.apache.org>

Ok, the logical place for this is general@jakarta, but I'm including tomcat-dev@jakarta so that the people who are there and not on general can see it. Please do not discuss on tomcat-dev, please only discuss on general.

In a closed source project where you've got a set team, you make decisions about where the entire team goes and somebody takes the lead of deciding what gets done when. In the discussions about Craig's long term plan, this metric was applied by several of us in thoughts about where to go next.

Apache Archive: Rules for Revolutionaries

After pondering this for a while, it's (re)become obvious to me that there's no way that anybody can expect an open source organization to work the same way that a team in a corporate setting can. Ok, so this is pretty freaking obvious, but I've been watching people that are not from Sun and who have been doing open source for a while talking and proposing things that come from this line of thought as well. Its not just people from Sun or people from any particular entity.

So -- in any software development project there is a natural tension between revolution and evolution. In a closed source environment, you make the call at any particular time on whether you are in revolutionary mode or evolutionary mode. For example, JSDK was in evolutionary mode for years. Then in Nov 98, We made a decision to go revolutionary. Of course, at the time the project team was composed of 1 person -- me, so it was an easy decision. After that revolution was over in Jan 99, Tomcat was in evolutionary mode getting JSP bolted in and working with J2EE. We (Sun folks) could do that because that was what suited the goals best at the time.

However, Open source is chaotic. With its special magic comes a different reality. This is:

- 1) People work on their own time (even people paid by a company can be considered to be working on their own time in this situation as each company is going to have different cycles and things they want)
- 2) People work on what they want to. If you are working on your own time, you are going to do what you want or you do something else.
- 3) Some people are evolutionaries, other are revolutionaries, and some are both at different times.
- 4) Both approaches are important and need to be cultured.
- 5) You really can't afford to alienate any part of your developer community. Innovation can come from anywhere.

To allow this to happen, to allow revolutionaries to co-exist with evolutionaries, I'm proposing the following as official Jakarta policy:

- 1) Any committer has the right to go start a revolution. They can establish a branch or seperate whiteboard directory in which to go experiment with new code seperate from the main trunk. The only responsibility a committer has when they do this is to

Apache Archive: Rules for Revolutionaries

inform the developer group what their intent is, to keep the group updated on their progress, and allowing others who want to help out to do so. The committer, and the group of people who he/she has attracted are free to take any approaches they want to free of interference.

2) When a revolution is ready for prime time, the committer proposes a merge to the -dev list. At that time, the overall community evaluates whether or not the code is ready to become part of, or to potentially replace the, trunk. Suggestions may be made, changes may be required. Once all issues have been taken care of and the merge is approved, the new code becomes the trunk.

3) A revolution branch is unversioned. It doesn't have any official version standing. This allows several parallel tracks of development to occur with the final authority of what eventually ends up on the trunk laying with the entire community of committers.

4) The trunk is the official versioned line of the project. All evolutionary minded people are welcome to work on it to improve it. Evolutionary work is important and should not stop as it is always unclear when any particular revolution will be ready for prime time or whether it will be officially accepted.

What does this mean?

In practice, this means that Craig and Hans and anybody else that wants to run with that revolution is welcome to do so. The only change is that it's not called Tomcat.next -- it's the RED branch or GOOGLE branch or whatever they want to call it.

Whenever Craig (or anybody else working on that codebase) wants to bring stuff into the trunk, they propose it here and we evaluate it on it's merits.

If somebody disagrees with Craig's approach (for the sake of argument here), they are free to create a BLUE whiteboard and work out what they think is a good solution. At that point, the community will have to evaluate both approaches. But since this is a populist society, with such a structure it is hoped that it becomes clear which is the preferred approach by the community by their participation and voting. Or maybe the best solution is something in the middle and the two parties work together to merge. Irregardless, the point is to allow solutions to happen without being stalled out in the formative stages.

An important point is that no one revolution is declared to be the official .next until it's ready to be accepted

Apache Archive: Rules for Revolutionaries

for that. There is the side effect that we could potentially end up with too many revolutions happening, but I'd rather rely upon the natural inclination of developers to gravitate towards one solution to control this than to try to control it through any policy statement.

When would this be official?

Well, if this is well recieved, we'd want to word it up and make it a bylaw (with approval by the PMC -- this is one of the areas in which the PMC has authority). Hopefully soon.

Comments? Suggestions?

James Davidson duncan@eng.sun.com
Java + XML / Portable Code + Portable Data
!try; do()