Overloading the development branch ? - a view of motivation and incremental development in FreeBSD

Niels Jørgensen
Roskilde University
nielsj@ruc.dk

Jesper Holck
Copenhagen Business School
jeh.inf@cbs.dk

Summary: Developer motivation in FreeBSD is enhanced by an incremental and unbyrocratic approach to integration. However, FreeBSD's community of developers is growing, and the approach may be difficult to scale. During times of intense development, the stream of contributed changes may "overload" the project's development branch, leading to build breakage and disruption of work.

The work organized by FreeBSD is maintenance-oriented: the operating system's 10th anniversary is up in 2003, and roots go back to the Berkeley Software Distribution of the late 70s. The project relies on voluntary contributions from 300 committers that have direct access to the source repository, and many external contributors. Each of the past few years, approximately 50 new committers have joined FreeBSD.

The project's approach to integration - the testing and other activities required to assemble parts into larger parts - is highly incremental and decentralized: Incremental in the sense that all changes to the software are made as modifications to the project's development branch and are required to preserve it in a working state. And decentralized in the sense that committers are allowed to add changes without asking for approval. Delegation of commit authority distinguishes FreeBSD from projects where a small group or an individual is in control of the repository, for example Linux.

Incremental and unbyrocratic integration motivates developers

Data indicates that the unbyrocratic and incremental process is highly motivating: As many as 81% responded that they were encouraged a lot by the delegation of commit authority (out of 72 FreeBSD committers replying to a survey I conducted in 2000). This includes committers whose work for FreeBSD is paid for by companies, for example a committer said *"I use FreeBSD at work. It is annoying to take a FreeBSD release and then apply local changes every time. When [..] my changes [..] are in the main release [..] I can install a standard FreeBSD release [..] at work and use it right away."* The potential for motivation in an incremental development model is recognized, for example, in Brooks' No Silver Bullet-paper ("Enthusiasm jumps when there is a running system") or McConnell in his work on rapid development (Rapid Development, Microsoft Press, 1996).

A complementary advantage of the delegation of commit responsibility is that the project is relieved of establishing a central integration team. Indeed, integrating other peoples changes may be viewed as less rewarding, and assignment to such tasks is used in some projects as a penalty (McConnell, op. cit., p 410).

Quality assurance by "parallel debugging"

FreeBSD's process for integration is decentralized to the point that the only requirement a committer must fulfill prior to commit is that he must test his code and subject it to reviewing by another committer. The FreeBSD project, similarly to many other open source projects, does not require changes to have their specification, analysis or design documented in writing and reviewed prior to implementation.

The community effort to test and debug the development branch (termed parallel debugging by Raymond) is perhaps the project's most important quality assuring mechanism, in a sense substituting the classical mechanisms based on written documentation. Other FreeBSD-processes related to quality assurance include the public nature of coding in the project, where the repository is web-browsable and each commit generates an automatic message to a developer mailing list, the granting of commit privileges to a person based on his merits in the project, and processes for revoking changes as well as commit privileges.

Parallel debugging of FreeBSD's development version is carried out, not by a dedicated testing team, but as a side-effect of other work: First, for mandatory testing of a change prior to commit, a developer must do trial integration of it into the most recent sources. Second, to benefit from the newest features and bugfixes, advanced users may wish to use the most recent version, possibly for purposes not related to FreeBSD development at all.

Scaling FreeBSD's process: overloading the development branch ?

FreeBSD's reliance on parallel debugging may have several disadvantages. One is that feedback generated may most often concern simple problems rather than subtle ones (challenging Raymond's thesis that "given enough eyeballs, all bugs are shallow"). Another is that the approach may be difficult to scale, because a single branch receives too many changes from too many committers.

FreeBSD's incremental approach bears resemblance with the XP methodology's approach to integration, mentioned by Beck as the main reason XP may not scale even to projects with 20 people: "The biggest bottleneck in scaling is the single-threaded integration process" (Kenn Beck, Extreme Programming Explained, 2000, p 157). There is even resemblance with code-and-fix practices suitable for toy programs developed by an individual.

Two issues related to "overloading" FreeBSD's development branch can be identified:

1. During ordinary development, bad commits is in inherent problem. Bad commits may break the build or just make the system unusable, and are a serious threat to the project: they halt parallel debugging and pre-commit testing. But bad commits cannot simply be forbidden. Ruling out errors in committed software defeats the purpose of parallel debugging. Preventing broken builds becomes more difficult as the number of commits increase and more hardware platforms are supported. Establishing a balanced interpretation of the "don't break the build"-principle is a major challenge.

2. During stabilization prior to production release, there is a conflict between using the development branch for stabilization vs. for new development. FreeBSD's stabilization period prior to release 5.0 of January 2003 lasted two months during which commits of new features were banned. The potential represented by developers with an "itch" to write new features is not used, they may even feel discouraged. To accommodate, the release engineering team may cut down on the stabilization period, that is, branch off at an earlier point of time a new production branch, where the stabilization effort is insulated from new development. Then commits of new features do not risk being production released prematurely, or introduce errors that disrupt stabilization. However, there is a trade-off, because splitting up into branches has a cost: First, bugfixes found during stabilization must be merged to the development branch. Second, and more importantly, the project wants everybody to focus on making an upcoming production release as stable as possible. Splitting up into distinct branches reduces load per branch, but splits the community's debugging effort, and so does not overcome what may be an inherent weakness in the incremental approach to integration.