Distributed Version Control Systems – Why and How

Ian Clatworthy, Canonical

Abstract

The Version Control space is undergoing a renaissance right now thanks to the increasing popularity of Distributed Version Control Systems (DVCS) such as Arch[1], Bazaar[2], BitKeeper[3], darcs[4], Git[5], Mercurial[6], Monotone[7] and SVK[8]. This paper explains why this technology is useful today and will be important in the medium to long term for most software development teams, whether open source or commercial. Guidelines are also suggested for selecting a tool and recommendations are presented on how to use the technology effectively.

The Challenges of Software Development

While there are important differences between open source and commercial development models, the core challenges are largely the same and timeless. As Fred Brooks observed many years ago, there will never be a silver bullet[9] to slay the software development beast. Most interesting software is inherently complex, users want it as soon as possible and typically have low tolerance for poor quality.

In the open source world and many commercial environments, a program is never finished while it has users with new ideas for what it can do, changing requirements, new environments or bugs. Software development is therefore not just about static design but very much about managing the evolution of design, communicating among the contributors and helping people understand what was done before.

To generalise further, Software Engineering is ultimately a communications challenge[10]. Version after version, the path from ideas to released code is often a long one involving many players . In commercial organisations for example, these include customers/users, sponsors, business analysts, architects, team leaders, software engineers, technical writers, quality engineers, support engineers, etc. The history of computer science is arguably one long procession of technologies designed to make that communication chain/cloud a more reliable one: higher level programming languages, OO, Use Cases, UML, design patterns, Agile development methodologies and beyond. The core challenge of software development is ultimately this:

How do we collaborate more effectively, both instantaneously and over time?

The Role of Configuration Management

Configuration management is a foundation best practice of every popular software engineering methodology regardless of the methodology type. Whether a waterfall, spiral, iterative, agile or open source approach is used, CM is fundamental because version control tools have proven themselves to be the most effective way, in terms of signal to noise ratio, of technical people communicating changes across the room to each other today, let alone across the globe over time. As a consequence, improvements to VCS technology have a direct impact on collaboration effectiveness. It is therefore no surprise that open source leaders such as Mark Shuttleworth and Linus Torvalds are among the driving forces behind this technology. Mark has stated that[11]:

> "Merging is the key to software developer collaboration".

Linus has strong views on how VCS tools support development or otherwise[12].

Changing the Game

There are numerous pros and cons to using a distributed VCS tool. As with most technologies, these vary depending on one's perspective: Developer vs Release Manager vs Community. Collectively, the benefits do add up and most early adopters of the technology are using it today for one or more of the reasons presented. For the vast majority of teams though, important questions remain:

1. Could some of the benefits be realised by waiting for later versions of existing central VCS tools?

2. Are there compelling reasons why distributed VCS technology will become mainstream regardless?

The answer to both questions is "yes". I fully expect better central VCS tools (or the promise thereof!) to delay mass adoption of distributed VCS tools in the short term, particularly while the DVCS tools are maturing. In the medium to long term though, the compelling benefits of DVCS technology will shine though.

Ultimately, a tool is just a tool and the real benefit comes from the processes it enables. There are two reasons I believe distributed VCS will ultimately replace central VCS technology across the industry:

- Better adaptability
- Used wisely, it delivers higher quality software

These benefits are explored further below.

Distributed VCS in a Nutshell

Wikipedia offers the following definition[13]:

> "Distributed revision control takes a peer-to-peer approach, as opposed to the client-server approach of centralized systems. Rather than a single, central repository on which clients synchronize, each peer's working copy of the codebase is a *bona-fide* repository. Synchronization is conducted by exchanging patches (change-sets) from peer to peer. This results in some striking differences from a centralized system:
>
> - No canonical, reference copy of the codebase exists by default; only working copies.
> - Common operations such as commits, viewing history, and reverting changes are fast, because there is no need to communicate with a central server.
> - Each working copy is effectively a remoted backup of the codebase and change history, providing natural security against data loss."

While this definition is technically correct, practically there is always a single copy that is sanctioned as the main development branch in teams using distributed VCS tools. Over and above the potential for disconnected operation, the primary differences between distributed and central VCS are these:

- Developers can collaborate directly without needing central authority or incurring central administration overhead
- The acts of **snapshotting** changes and **publishing** changes can be decoupled.

These differences mean that the integrity of the main trunk remains higher over time. This has a positive impact all round:

- developers are continuously working from a more stable base (so much less time is needed tracking down why their last `update` command has broken their sandbox)
- better quality is delivered when the time comes to ship reducing stress on the QA and release management roles/teams
- the VCS server set-up and managed by the system administration team does not need as much disk space, CPU, bandwidth, etc. to accommodate the concurrent load needs of the team/community[14].

The Developer View

One of the reasons why DVCS technology is gaining popularity is because developers seem to prefer them. Some notable benefits are:

1. **Disconnected operation** - developers can still be productive when the umbilical cord to their central VCS repository is broken, e.g. when travelling.
2. **Experimental branches** – creating and destroying branches are simple operations. This is particularly useful when experimenting with new ideas, e.g. a 'spike' when using eXtreme

Programming.

3. **Easier ad-hoc collaboration with peers** – intelligent merge tracking means merging early and merging often is both possible and surprising unpainful. It is difficult to explain just how much of an impact this can make on how co-developers can work together more easily, e.g. when Pair Programming.
4. **Staying out of the way** – collectively, the above features mean that developers spend less time on mechanical chores and more time on tasks that add value.

The Release Manager View

Good Release Management is a complex art, balancing the numerous trade-offs implied by the classic management triangle: Scope/Resources/Time. A great deal of the success enjoyed by teams using Agile Development Methodologies comes from using smaller iterations more frequently and the adaptive planning[15] that occurs as a consequence. Even with the best adaptive planning and continuous integration practices though, centralised VCS trunk quality "dips" during the course of each iteration and a scramble often occurs towards the end of the iteration to restore quality to the previous level or better.

In an ideal world, a Release Manager would built each new version by picking and choosing "lego brick" size blocks of functionality - features, improvements and bug fixes. If a feature doesn't meet user requirements or quality standards during testing, it should be possible to truly drop it as an atomic change (not just leave the code largely in place and hide access to it). Following the Defer Commitment practice of Lean Software Development[16], the Release Manager could defer decisions to the last possible moment, not just the start of each iteration.

Given the inherent complexity of software, this nirvana will sadly never arrive. However, distributed VCS brings us closer to it than ever before *provided* developers work in **feature branches**[17] either as individuals or in small groups. While this is technically possible using massive amounts of branches in a centralised VCS tool, distributed VCS tools make it practical thanks to their intelligent merging and their support for pulling changes just as easily as pushing them.

Viewed from this perspective, distributed VCS tools become a natural progression of the state of the practice. Subversion improved on CVS by making tree-wide commits atomic while Bazaar (for example) improves on Subversion by making "feature-wide commits" (i.e. feature branch merges) atomic.

The Community View

Every innovation starts with a problem that existing technology doesn't successfully solve. Mark Shuttleworth has explained that his driving motivation[18] for developing DVCS technology is because of its positive impact on open source communities:

> "Distributed version control is all about empowering your community, and the people who might join your community. You want newcomers to get stuck in and make the changes they think make sense. It's the difference between having blessed editors for an encyclopedia (in the source code sense we call them "committers") and the wiki approach".

Beyond lowering the barrier to entry, there are other notable community benefits:

- an easier migration path from non-core to core contributor
- as the community grows from 1 to 10s to 100s to 1000s, different workflows can be adopted without needing to change the VCS toolset
- branch management scales better than patch tracking.

The last point is important for teams or people who want or need to keep a list of patches and reapply them when a new upstream version is released. While tools like Patchwork Quilt[19] are useful, it is typically much easier to do that particular dance using a DVCS and either:

- having your own branch with patches applied and merging the new work, or
- reapplying bundles, i.e. patches with the full intelligent metadata available.

A more detailed analysis of this issue has been provided by John Arbash Meinel[20].

The communities we join, whether off-line or on-line, say a lot about the sort of people we are. The flip side to this is that the tools we offer, when starting a community, implicitly say a lot about the sort of people we want to sign up.

The Senior Management View

There are many complex issues and few easy answers facing senior IT managers in most large corporations. Many IT best practices (outsourcing, offshoring, agile) are inherently in conflict and the risks – wasting money solving the wrong problems, quality, unpredictable exchange rates, skill shortages in development centre locations, etc. - can be high. As the open source movement has shown with stunning results, effective distributed collaboration *is* possible given the right people, the Internet and the right toolset. New ways of building companies and teams are now a reality and distributed VCS is one of the important technologies required. To quote Brian Aker, Director of Architecture at MySQL AB[21]:

> "We have been using a distributed source control system since 2000. Our development process, which allows us to span multiple countries since all developers work from home, wouldn't work without it."

Of course, employing the best people possible regardless of their location is nothing new. The change is that the areas where that is viewed as a best practice will expand from consulting to engineering and beyond.

Problems to be Aware Of

At this point in time (Q4 2007), assuming every feature you need is there and rock solid in a given DVCS tool would be unwise. The maturity of all DVCS tools has a way to go in comparison to industry standards such as Subversion. As a rule, quirks and rough edges remain in both the tools and their documentation. 3rd party support in terms of integration into other tools is often beta quality. Text books and training courses are generally not yet available.

It must be stressed though that the leading tools are clearly good enough for a huge number of projects to adopt today. Interestingly, many of the tools are indeed much better in many regards than more established tools in ways users may not initially expect. For example, all the leading DVCS tools are incredibly good at efficient project history storage.

Maturity and 3rd party support will undoubtedly remain a tactical adoption issue for a fair number of projects in the short term. Given their rapid rate of development, I expect this issue to largely disappear over the course of 2008. The strategic debate will then most likely pick up steam.

The Arguments Against Distributed VCS

Greg Hudson has argued that the distributed VCS approach and the pyramid development model used by the Linux kernel team has numerous limitations and that a central repository with multiple committers would work better[22]. Linus has rejected this arguing that the approach implements the practical reality of developers using a "network of trust" to get things done. I agree that the centralised workflow model has far more merit than Linus gives it credit for but his network of trust argument is rock solid. Indeed, most businesses and all teams I've ever seen *unofficially* operate on trust, regardless of the official management and technical hierarchies.

Ian Bicking has written about the potential downside of the distributed approach[23]. To summarise his concern, sharing early and often (as the central approach effectively mandates) encourages a better dynamic, particularly in the open source world. A rebuttal of this has been given by Bryan O'Sullivan[24]. I would further argue that (team-wide) publishing of changes once complete actually increases the quality of communication within a team. For example, receiving a diff about a completed logical unit of work is far more valuable than lots of smaller diffs (that may not make sense in isolation and a higher percentage of which are likely to be going down an incorrect path). Free services such as Launchpad[25] also help by making it easy to publish new branches to a central registry.

Havoc Pennington[26] has argued against the broader applicability of the distributed approach:

> "I think what I don't get yet is why you'd *want* to maintain a bunch of local changesets for very long. The

Linux-kernel-style fork-fest seems just nuts for anything I'm working on."

He has also raised the issue of generally poor usability:

> "The distributed systems seem pretty wild from a user experience standpoint. In the sense of "jeez, I can tell a (kernel, Haskell, shell) programmer wrote this." Subversion may be less flexible but it's also less *confusing*."

These concerns should not be dismissed lightly. For distributed VCS to gain industry wide adoption, it needs to embrace and extend the central approach the vast majority of teams are comfortable with today. As I have argued previously, the future of version control is neither central nor distributed - it's adaptive[27]. There *are* compelling advantages to distributed VCS technology but many users and teams will adopt it for incremental improvements initially, looking for flexibility down the track - not a whole new way of working on day zero.

Selecting a Distributed VCS Tool

Of the numerous tools available, a small number will reach a critical mass of acceptance. In my opinion, the most likely candidates are Bazaar, Git and Mercurial. A sample of the projects adopting these three tools are Ubuntu, the Linux kernel and OpenJDK respectively. Popularity aside, when selecting a tool, the criteria to consider ought to include reliability[28], adaptability, usability[29], extensibility[30], integration[31] and low administration. I have previously published a series of articles covering these criteria in more depth.

The Case for Bazaar

While recognising Git and Mercurial for the amazing tools that they are, I believe Bazaar will be the right strategic choice for a majority of teams in the future, if not already. (Disclaimer: I work for Canonical, the company sponsoring Bazaar, and joined the development team in early 2007 accordingly.) Bazaar has a high focus on:

- **usability** – being a tool developers love to use (I.e. having an interface that exposes just enough detail, having a straightforward command set, trying hard not to unpleasantly surprise users)
- **flexibility** - enabling whatever workflow makes sense across the collaboration spectrum: from personal productivity tool to communities with thousands of users
- **correctness** - doing the right thing (e.g. comprehensive rename support so that merging just works as frequently as possible)
- **quality –** Bazaar has a test suite with around 8500 tests (and growing)
- **cross platform** support (though Mercurial is also good in this regard).

For these reasons among others, I believe Bazaar is, and will most likely remain, easier for many teams to migrate to than the alternatives. In the open source world, Bazaar has the added benefit of free hosting on Launchpad and strong integration with the collaboration tools it provides. For example, Launchpad makes it easy to register branches, find branches and associate branches with product releases and bug fixes. In the commercial world, Bazaar provides the easiest upgrade path for teams largely happy with centralised workflow and commercial support is available if required.

To date, the primary issue with Bazaar has been performance on large code bases, particularly across high latency networks. This issue has been largely addressed by the introduction in 0.92 of a new format known as *packs*. As well as greatly improving performance on some key use cases, packs are important to Bazaar's evolution because they enable some interesting new features under development such as *history horizons*. With a strong and growing community actively developing it using numerous best practices and a productive programming language (Python), it is common for each (monthly) release of Bazaar to include 50-100 enhancements. It's an exciting community to be a part of and we are always looking for new members. (If Bazaar sounds exciting to you, please consider helping us to change the world.)

Using Distributed VCS Effectively

Distributed VCS is a new field and, as a rule, best practices are still evolving. Existing Software Engineering texts rarely mention DVCS and tool-independent books on the topic may well take another year or two to

surface. With these facts in mind, the best source of information of how best to use the technology is arguably the open source communities: the IRC channels and mailing lists for the various tools.

Some general recommendations are:

- Make sure the location of the primary development trunk is well publicised
- Use a program such as Robert Collins' PQM[32] together with an automated regression suite to ensure its quality is sacred
- Use a program such as Aaron Bentley's BB[33] to track submitted patches and their review status.

Distribution increases flexibility but a sensible amount of central data management can greatly aid effective collaboration. For example, Launchpad provides a central nexus where people can find information about open source projects, distributed team members, find branches, browse the code and so on. Alternatively, tracking important branches can be done using whatever technology makes sense, e.g. Wiki pages, a CMS, database or shared drive in a commercial environment.

PQM (Patch Queue Manager) automates commits to a branch by enforcing a test suite on the result of automatic merges. This is important because, as a project grows, the time to run a thorough automatic test suite grows as well. Development either slows down or developers stop running the tests on every single commit. The latter option is the road back to poor quality where regressions are only found after they've been in the main branch for some time.

BB (Bundle Buggy) retains an action queue of patches still needing review. It is an excellent aid to the typical open source process of people posting patches to a mailing list. Too often these patches are dropped, or become an interruption to the reviewers' work if they must be handled immediately. Working with Bazaar, BB keeps track of patches that have been merged or superseded, and it accumulates comments and decisions from reviewers. To put the importance of this in perspective, here's a simplistic summary of the key difference between commercial and open source project management:

- Commercial: *How fast can we plant?*
- Open source: *How fast can we harvest?*

BB is Bazaar's harvesting dashboard of choice.

Conclusions

The new breed of distributed version control tools are exciting because they change the software engineering game: they enable new ways of collaborating and that in turn enables new ways of thinking about and executing software development. Given its inherent distributed and loosely coupled nature, the open source community has much to gain from DVCS technology. As distributed development and open source practices gain popularity in commercial teams, the technology will become increasingly valuable there as well.

Adoption of DVCS technology will be piecemeal in the short term as the tools and 3rd party support matures. Nevertheless, DVCS is here to stay as the process flexibility it offers and higher quality software development it enables are compelling advantages. In particular, DVCS makes development in feature branches practical, maximising the rate at which teams can deliver value to users – a key tenet of Lean Software Development. By 2011-2012, I predict this technology will be widely adopted and many teams will wonder how they once managed without it.

Of the large number of available DVCS tools, three stand out as likely to gain a critical mass of acceptance: Bazaar, Git and Mercurial. With a high focus on doing the right thing, usability, workflow flexibility and cross platform support, Bazaar is a safe choice for many teams looking to make the most of this technology.

Acknowledgements

Thanks to Elliot Murphy and Martin Pool for reviewing this paper.

[1] Arch home page, http://www.gnu.org/software/gnu-arch/
[2] Bazaar home page, http://bazaar-vcs.org/
[3] BitKeeper home page, http://www.bitkeeper.com/
[4] Darcs home page, http://darcs.net/
[5] Git home page, http://git.or.cz/
[6] Mercurial home page, http://www.selenic.com/mercurial/wiki/
[7] Monotone home page, http://monotone.ca/
[8] SVK home page, http://svk.bestpractical.com/
[9] Fred Brooks, No Silver Bullet: Essence and Accidents of Software Engineering, IEEE Computer, Apr 87, http://en.wikipedia.org/wiki/No_Silver_Bullet
[10]Ian Clatworthy, Collaboration Redefined, http://ianclatworthy.wordpress.com/2007/06/08/collaboration-redefined/
[11]Mark Shuttleworth, Merging is the key to software developer collaboration, http://www.markshuttleworth.com/archives/126
[12]Linus Torvalds, Google tech talk: Linus Torvalds on git, http://www.youtube.com/watch?v=4XpnKHJAok8
[13]Wikipedia, Distributed revision control, http://en.wikipedia.org/wiki/Revision_control#Distributed_revision_control
[14]Mark Reinhold, Source-code management for an open JDK, http://blogs.sun.com/mr/entry/openjdk_scm
[15]Craig Larman, Agile and Iterative Development: A Manager's Guide, Chapter 2 – Iterative & Evolutionary, http://safari.oreilly.com/0131111558/ch02
[16]Lean Software Development home page, http://www.poppendieck.com/
[17]Steve Berczuk & Brad Appleton, Software Configuration Management Patterns: Effective Teamwork, Practical Integration, Chapter 19 – Task Branch, http://www.scmpatterns.com/book/
[18]Mark Shuttleworth, Renaming is the killer app of distributed version control, http://www.markshuttleworth.com/archives/123
[19]Patchwork Quilt home page, http://savannah.nongnu.org/projects/quilt/
[20]John Arbash Meinel, Thoughts on branch tracking vs patch tracking, https://lists.ubuntu.com/archives/bazaar/2007q4/032519.html
[21]Brian Akers, response to “Distributed Source Code Management – Niche or Trend?: The Q&A” by Stephen O'Grady, http://redmonk.com/sogrady/2007/06/26/dscm/
[22]Greg Hudson, Why BitKeeper Isn't Right For Free Software, http://web.mit.edu/ghudson/thoughts/bitkeeper.whynot
[23]Ian Bicking, Distributed vs Centralized Version Control, http://blog.ianbicking.org/distributed-vs-centralized-scm.html
[24]Bryan O'Sullivan, On distributed and centralised revision control, http://www.serpentine.com/blog/2005/08/10/on-distributed-and-centralised-revision-control/
[25]Launchpad home page, https://launchpad.net/
[26]Havoc Pennington, Source control, http://log.ometer.com/2006-10.html
[27]Ian Clatworthy, Version Control: The Future is Adaptive, http://ianclatworthy.wordpress.com/2007/06/21/version-control-the-future-is-adaptive/
[28]Ian Clatworthy, Wanted: Rock Solid Version Control, http://ianclatworthy.wordpress.com/2007/07/11/wanted-rock-solid-version-control/
[29]Ian Clatworthy, It Takes a Community to Raise Great Software, http://ianclatworthy.wordpress.com/2007/07/02/it-takes-a-community-to-raise-great-software/
[30]Ian Clatworthy, Version Control: Plugins vs Toolkits, http://ianclatworthy.wordpress.com/2007/07/18/version-control-plug-ins-vs-toolkits/
[31]Ian Clatworthy, Version Control: Design for Integration, http://ianclatworthy.wordpress.com/2007/07/30/version-control-design-for-integration/
[32]PQM home page, https://launchpad.net/pqm
[33]Bundle Buggy home page, http://bundlebuggy.aaronbentley.com/