LUCAS WARD'S BLOG

WEDNESDAY, FEBRUARY 3, 2010

A Maturity Model for Source Control (SCMM)

Most enterprise developers are already familiar with the Capability Maturity Model. Those of you with Agile tendencies may have also heard of the Agile Maturity Model. The purpose of these models is to objectively assess an organization's maturity in a particular methodology. Despite any feelings you may have on CMM or waterfall in general, having an agreed upon way to assess the basic principles of a methodology can be quite useful. One area where this type of assessment typically falls under the radar is SCM. Many companies look at a particular SCM solution, and believe it covers every need. However, when objectively analyzing the various uses and features of SCM systems, a type of maturity model begins to emerge.

As with other maturity models, we can use a similar numbering system to delineate how 'mature' an organization or tool is, with a higher number being better. As you can imagine, zero corresponds to a complete lack of source control at all, or perhaps a shared file system and a roofing shingle. The highest level is reserved for the most advanced tools such as Git or Mercurial, that allow for advanced SCM capabilities such as zero-cost branching and merge through rename.

In doing this analysis, some common feature-sets emerge:

- Atomicity: If you check in 5 files, and there is an issue with one, none of the five should be committed.
- Revision Tagging: The ability to take a snapshot of the state of your codebase at a particular moment in time, and refer back to it later.
- Branching: The ability to have more than one version of the codebase in active development, seperate from each other.
- Merging: Taking different versions of the same codebase and combining those changes. This may mean changes from a local copy or between two branches, with a complete history of how these various pieces have been merged.
- Merge through Rename: A more advanced SCM feature for dealing with merging files that have been renamed. Consider file A in branch 1 and 2 that is then renamed to B in branch 2, but not in 1. Then consider what would happen when A's contents are changed in branch 1. SCM systems that support merge through rename allow for this and more, all others blow up.
- Repository Navigation: The means of accessing the contents of the repository. Subversion, for example, has a web interface. (among many other third party tools) While Clearcase has thick client access.

Lack of SCM Bravery

ABOUT ME

LUCAS WARD

I'm a Consultant at ThoughtWorks. I code primarily in Java. I co-led the Spring Batch project.

VIEW MY COMPLETE PROFILE

OPEN SOURCE PROJECTS I AM A COMMITTER ON

Spring Batch

SOCIABLE

Not Found

Error 404

BLOG ARCHIVE

one. They had taken that lesson, and were going to make damn sure they were never going to use the lame one again. Sadly that meant there were risk averse in respect of a third or fourth choice. Perhaps also as all the tools chains and workflows are different, there was caution based on that too.

Much has changed since then with experience of half a dozen much more common place now. However, if you walk into most enterprises today, you're likely to see only one of two SCMs: Clearcase or Subversion. For those of us who have used many of the SCM systems below, (especially us consultants) it is hard to understand why someone would continue to use some of the SCM systems that are lower on the maturity model below. It can be easy to think that it is imposed upon them by the powers that be, however, in many cases they will tell you at length why their SCM is better, or at least why they don't feel the need to switch. This stems from two fundamental issues:

1. They're pervasive. Like an IDE, they're there every second of every day, everything we do as programmers short of documentation needs to be checked in. It has a huge impact upon our day to day lives.
2. Steep learning curve. Regardless of the solution, and any third party tools helping to make it easier, source control in and of itself is complex. Switching requires that you accept a drop in productivity, and no one wants to become a newb again.

Five maturity Levels

Shadowing the CMM, we're aiming at five levels above zero.

Level Zero - 'No SCM'

No source control solution at all, or a shared file system with periodic backups. One developer, or a few at most share source without tools, and as such run a number of risks:

- Source may not be compilable at any moment
- Source may be lost because of developer error
- Extremely easy for developers to overwrite each others changes.

There is nothing to redeem in this level, or build on for future ones.

Level One - 'First Foray'

- Developers have workspace on network and cannot work offline
- Run a build means go to a long lunch
- Refactoring - if it works at all, is deathly slow
- Checkouts slow enough to be done over night
- Checkins slow
- Non-atomic commits
- Branching and tagging expensive
- Personal/local branching means second checkout
- Centralized rather than distributed
- Unusable or slow merge point tracking
- Merge through rename - merge, rename not understood by tool
- Repository can corrupt on occasion, high administrator/expert to developer ratio (1:10)

work well over long distances. Renaming of resources may be hard to impossible. Branching and tagging may require permissions on triplicate stationary, and with a slaughtered lamb or incense stick or two (they take a while and eat disk).

Examples:

Visual Source Safe

The core of VSS's problem is that it is not client/server and has not moved forward much in a number of years. Developers can hurt each other with locking files and even marking files as shared (an esoteric feature).

Clearcase's Dynamic mode of operation

Though an enterprise tool that sold widely into enterprises, it is like wading through molasses (treacle for Brits) to use. It mounts a network share or two for you, and thats what you edit on, and compile against. Both IDEs and builds are slow as a consequence. Dynamic checkouts are also precarious as you may have more than one developer active whithin it, and for periods of time, you may observe the codeline to be non-compilable. Furthermore, it does not support atomic commits. Though three way merge is actually advanced, nothing else about Clearcase in dynamic mode (including UCM mode) makes you want to particularly recommend it to anyone for any purpose. Merges from one active branch to another can actually take longer than the time spent making the original commits to the donor branch. There is a inverse square law at work with these installs of Clearcase, in that capacity drops off exponentially as you add more staff to it, and attempt to get busier (more throughput) with it. Clearcase sales folks and consultants recommend this mode of operation. Anomalous for Clearcase (dynamic mode) when rated against the bullets above, is that it can merge through rename, which is the preserve of more advanced tools (see later).

Level Two - 'Clunky'

- Developers have local copies and can work offline
- Local file system means fast(er) builds
- Refactoring - will go through; make a cup of tea
- Checkouts will trickle past as if in bullet time
- Checkins potentially still slow
- Non-atomic commits
- Branching and tagging potentially expensive
- Personal/local branching means second checkout
- Centralized rather than distributed
- Unusable or slow merge point tracking
- Merge through rename not working, requires extensive follow up conflict resolution before commit
- Repository can corrupt on occasion, administrator to developer ratio sub optimal (1:20)

Examples:

CVS

optimistic locking (it kinda introduced this), and developers find working offline a breeze. Commits back can still be like pulling teeth, in that every directory in a source tree is analyzed for differences with the server version via a wire call (just a hash but even so). Sadly, branching and tagging are still costly. Some folks still prefer CVS over others listed below.

TFS

Microsoft have tried to make something in Perforce's image (they historically ran a private fork of Perforce from the 90's called Source Depot), but do not have all of the features and performance of Perforce. They rammed in too many non-source control features and it is falling short of the mark in terms of installation and administration costs. Its pretty much wedded to Windows developers, and correspondingly leverages a ton of other MS server side pieces. Day to day integrated operations in Visual Studio are where you feel forcibly slowed down versus not using SCM. Agile folks approach checkout/checkin/conflict resolution with dread. This tool should be higher ranked based on features, but is a let down in implementation.

Clearcase static mode of operation

This is where Clearcase checks out a branch to your C:\ drive. Much like CVS, but a little faster. Builds run as fast as is possible for a local IDE drive (a gazillion times faster than a overloaded 10-base-T ethernet network with 50 devs on it). You're not going to get hosed by some other developer rendering the branch non-compilable (phew!). At least not unless they checked in that broken state. You don't automatically keep abreast of communal efforts - you have to sync/update periodically. Clearcase sales folks and consultants **do not** recommend this mode of operation, for some strange reason. Anomalous for Clearcase (static mode) when rated against the bullets above, is that it can merge through rename, which is the preserve of more advanced tools (see later).

Level Three - 'Basic'

- Developers have local copies and can work offline
- Local file system means fast builds
- Refactoring - speedy; smile at your neighbors for a second
- Checkouts might finish before you die of old age
- Checkins very fast
- Atomic Commits
- Lightweight (cost free) tagging and branches
- Rudimentary branching and merging
- Personal/local branching means second checkout
- Centralized rather than distributed
- Rudimentary merge point tracking
- Merge through rename not working, requires extensive follow up conflict resolution before commit
- Repository can corrupt on occasion, low administrator to developer ratio (1:100)

Examples:

Subversion

Compared to CVS its a definite advance. There is much improved speed. That is for normal checkin checkout operations and branching and tagging. It is a comfortable place for Agile developers to feel that their love for throughput is requited. Just don't try to do frantic parallel development on more than one branch.

Level Four - 'Effective and Reliable'

- Developers have local copies and can work offline
- Local file system means fast builds.
- Refactoring - speedy; smile at your neighbors for a second
- Checkouts reasonably fast.Checkins reasonably fast
- No-op sync/update very fast
- Atomic Commits
- Lightweight (cost free) tagging and branches
- Advanced branching and merging
- Personal/local branching means second checkoutCentralized rather than distributed
- Sophisticated merge point tracking
- Merge through rename only possible with configured branch mappings, otherwise a fix-up before commit is required
- Repository corruptions very rare, very low administrator to developer ratio (1:1000)

Examples:

Perforce
Fast is what this ten year old tool was built to be. In some operations nothing beats it. Back in the day, Perforce heralded a number of firsts in terms of capability (atom commits, lightweight branching/tagging, three-way merges).

This is bound to be controversial at 'level 4' as P4 uses read-only locks extensively. Some Agileists are going to violently object as this mandates good tool support (Intellij juggles the read-only flags via perforce, but Vim does not). This severely limits the ability to work offline (making it anomalous versus the definition for level 4). You can still do it, you have to blast away the read only flags, and do a revert-unchanged when you reconnect later, but don't try to do renames while offline - it'll end in tears. Also refactorings (when there is tool support) will work, but will be slightly slower than for the likes of Subversion as network IO is happening per changed file.

Level Five - 'Speedy, Invisible, and Highly Capable'

- Developers have local copies and can work offline
- Local file system means fast builds
- Refactoring - speedy; crick your knuckles momentarily
- Checkouts reasonably fast
- Checkins reasonably fast
- No-op sync/update very fast
- Atomic Commits
- Lightweight (cost free) tagging and branches
- Advanced branching and merging

- Seamless merge through rename - no configuration needed
- Sophisticated merge point tracking
- Repository corruptions very rare, almost invisible administrator to developer ratio (1:10000)

Examples:

GIT and Mercurial

These two are very similar, and both have their fans. It is difficult to tell which will win out over the other in time. For both, the killer capability for Agile folks who treat previously committed code like wet-paint, is merge through rename. It works so well that you feel this is how Fowler intended refactoring to feel and that all other SCMs fall short on that vision. In short Git and Mercurial have the speed of Perforce, plus easy local branching, plus distributed operation, plus merge through rename.

Level Minus One - 'Death Wish'

There is a special place in hell for PVCS Dimensions. We don't know about the latest version which is rumored to have atomic commits, but we'e pretty sure that the turn of the millenium version was the worst of Clearcase but with a bucket full of suck thrown into the mix. For example, person A doing a sync/update in the morning would take 45 mins, but if person B started their sync/update after person A, then 1.5 hours would be the reality (even if nothing changed). Sometimes things were so bad, an ad-hoc distributed mode would leap into being (devs putting sets of changed files on network mounts, floppy disks or emailing). As far as the authors can recall, there is 100% correlation between PVCS-Dimensions use and failed projects.

Anomalies

Subversion with Git as a front end. Git has built in support for Subversion servers. Given some time, Git can clone a whole Subversion repository to a local workspace. From there you get the quick branch juggling capability, as well as local commits that can be sent back to Subversion later via 'dcommit'. Speedy branch local juggling (a level 5 gain) is possible just like for Git proper. However, Git as a front end for Subversion cannot participate in Subversion merge point tracking (yet), thus we cannot back-implement another level 5 feature - merge through rename. Lastly, Git fronting Subversion is still mostly tied to a single server, so cannot claim to be distributed in the text book sense of the word. Where should it be placed ? 3.5 or 4 perhaps in terms of maturity, but merge lets it down.

Seamless merge through rename is the high bar

You're most likely to be using Clearcase in the enterprise today, and may have heard that a shift to Subvesion will be more productive (and a lot cheaper). Though Subversion is enterprise approved now and taking over from Clearcase as #1 soon enough, it is time to look for new tools. Git and Mercurial mark the high bar now in many ways. However one feature stands out for Agile teams chasing high throughput - seamless merge through

This blog post was the result of much discussion and pair blogging with Paul Hammant.

POSTED BY LUCAS WARD AT 8:52 PM
LABELS: SCM

24 COMMENTS:

ketchup said...

Nice job.

But is looks a bit "biased" against dynamic ClearCase: This can be quite responsive, even for large installations. It can be done! :)

Further, "Branching and tagging expensive" is a bit broad, since branching and tagging are quite different use cases: Having expensive branches might not be as big as a problem as having expensive tags. Depends on your development model, of course.

Then again, I agree that Git and probably Mercurial (I dont use hg, so I cant really tell) are clearly marking the top of the line, if only technology is considered, and I clearly prefer git over any other SCM I worked with, including SVN, ClearCase, Synergy and, of course, CVS/RCS.

But Enterprise decisions are made at least partial in laws department, which calls, as a minimum, for controlling (read) access and enforced certified workflows. The nature of DSCM is against that, and evolution will take time. So I guess well see Subversion around for quite some while.

FEBRUARY 5, 2010 5:59 AM

Cosmin Stejerean said...

There is nothing about Git that prevents controlled read/write access or implementation of enterprise workflows.

Between the distributed nature of Git and the available hooks one can implement even the most complicated enterprise workflows.

Perhaps we need more commercial support options for Git in order to allow companies to feel comfortable.

FEBRUARY 5, 2010 9:47 AM

Mike Mason said...

Could you articulate why Subversion doesnt make it into the next level? By my reckoning it hits most of the criteria.

Whilst the best developers in the world undoubtedly will make use of the new features in advanced tools like Git and Mercurial, those features are likely to be dangerous to the average corporate developer.

where such merge swapping is even easier, would make this problem worse and make people even less likely to actually fix their schedules and engineering practices.

FEBRUARY 5, 2010 2:07 PM

Paul Hammant said...

Mike,

The Svn, P4, Git/Mercurial rankings above are ultimately about merging as you surmise.

Subversion is weaker than P4 presently because its merge point tracking cannot work through a rename. Indeed for Subversion 1.6.3 through 1.6.5 the merge would abort in particular situations because of conflicts concerning renames. 1.6.6 sailed past the abort, but left us with conflicts that wed need to resolve later. That conflict would not have existed in P4 with the appropriate branch-spec (see below). It would not have existed in Git/Mercurial at all.

Perforce can do merge through rename if you setup branch mappings - though its pretty second class. It cant be tooled because branchspecs are in themselves committed. They transcend versions. If they were committed as part of a change list, then the likes of Intellij, Eclipse and Studio could update specs when refactorings happen.

While I agree that trunk is best, merge is also used for updating working copy (tis often forgotten). To that end, Subversion leaves you abruptly, Perforce cant help because a branch-spec is about branches not working copy, and only Git and Mercurial update your working copy with renames/moves regardless of whether you had local commits that have not been pushed to the backend.

Now there is nothing here that the Subversion teams (or Perforce the company) cannot fix.

Meanwhile, like you, true trunk based development is what I recommend as teams mature through experience with SCM tools.

FEBRUARY 5, 2010 2:35 PM

juancn said...

I use a clearcase derivative at work (which added a form of pseudo atomic commits). Its fugly, slow, and extremely unreliable!

We actually built a bridge using git as a frontend to make it palatable (with a lot of script-fu to keep histories in sync).

With network latencies > 200ms, the unnamed-CC derivative is completely unusable (Im in Argentina and the data center is in the US).

ketchup said...

Cosmin,

hooks are nice. Problem is: they are not mandatory. You cannot force developers to use them, so you cannot enforce. Hence, you cannot implement an Enterprise workflow (let alone certificate it), because a workflow you cannot enforce is ... not a workflow.

If you are Enterprise, you dont assume loyal developers. They are human resources, as loyal as you pay them if youre lucky (but Enterprise is not about being lucky). Hence they might work for your competition tomorrow. You dont want to give your competition (or whoever might be after your busines vital information) a headstart by handing over all your code. A, sorry, heres the right button: *all* your code, ever.

In other words: Enterprise is driven by accounting and legal, not by geeks.

Next, Enterprise loves Windows as workstation platform. Theres no Windows git available (*dont* tell me theres cygwin unless cygwin git can clone a repository at the first try again).

You see, its not only git that has a hard time in Enterprise. Its primarily that DSCM is not yet trusted, and this distrust is not entirely unreasonable.

FEBRUARY 6, 2010 2:16 AM

Cosmin Stejerean said...

You might not be able to enforce hooks that run on the developers workstation, and I dont think one ever should try to enforce hooks at that level.

You can however enforce the hooks that run on the canonical source repository, which is what really matters for enforcing controlled read/write access. You can prevent people from pushing changes to a certain branch, from pushing changes that touch a certain file, from pushing changes with commit messages that are not to your specification, etc.

If you are curious about how specific workflows you have encountered can be implemented, I will gladly attempt to give you examples of how I would implement them in Git.

The whole argument about non-loyal developers stealing source code is a red-herring as it has nothing o do with Git or any other source control being used.

Regarding Windows support, it is true that traditionally Git has been a second class citizen on Windows, requiring cygwin to run. That said, support for Git on Windows has become a lot better recently.

Git under cygwin should run just fine, but if that does not work for you, or you do

FEBRUARY 6, 2010 9:32 AM

Paul Hammant said...

IRO Red-herring or not, lets expand that a little ..

With Svn (and Git-Svn proves it) you can pull out all commits ever to a repo (trunk + all branches) and end up with a zip that is quite manageable in terms of flash drive. Thus Svn can also fail the "competitor could receive all code ever too.

That said, paths can be hidden behind Apache DAV permissions. Perforce can similarly have restricted paths/branches. Clercase breathes this sort of restraint.

Thus we need to work out configurations whereby the canonical git repo has variable restrictions per cloner.

FEBRUARY 6, 2010 9:46 AM

ketchup said...

Cosimin,

with hooks you can implement *write* policies, not *read* policies. Thats basically impossible in a DSCM, since all you need is find someone who has read access, and clone his repo.

Red-herring or not (I disagree with you: Ive seen co-workers sending out sensible data because they where not aware of the problem, and Ive seen co-workers sending out code because they where unaware of any problem), this is an issue Enterprise has. Try it: Suggest to your project leader you want to copy all the source of your project onto a stick and walk it out the door, destiny unknown. Its a great exit strategy. From your project.

Second, cygwin git does not work as expected, since cloning is broken. Google for cygwin 1.7.1 git. As soon as cygwin has an issue tracker (besides subscribing to an email list), I might consider cygwin as a viable alternative to a Unix environment. (Hint: If you have a cygwin prior to 1.7.1, and you need itjust for git, dont upgrade until someone blurrs out its working again. Trouble is to find out when its fixed.)

@Paul,
yes, with the git-svn "bridge" you can suck an SVN dry. But neither does svn make that especially easy for you, nor is this the "fault" of svn. Plus your admin might implement policies here, and it would work for any CSCM you create a bridge for git for.

Dont get me wrong: Just because I knwo these arguments against git/hg does not imply I share the objections - at least not completely (face it: developers are as fallable as every human). I really dont want to defend the Enterprise position, especially not against git, which is, dunno how often I had stated this, my avourite SCM ever (and I worked with quite some SCMs). Its just that DSCM *inherently*

Paul Hammant said...

Git on Windows ..

Im interested by AndLinux.org. As soon as I can make some space on my MBP, Im going to do Win7 + AndLinux.org.

It might be more of a first class place for Gitters on Windows.

on Svn not making it easy

In Summary.

Ive thought some more and I think the summary of the permissions issue for Agile teams is that while all folks in a dev team should see all of trunk, some enterprises want to make release branches read restricted because theyre putting settings into bash scripts or properties files (or alike).

Going back to trunk, Ive only seen once in thirteen years of using SCM a company put read restrictions on items in trunk. That company had co-mingled multiple separate projects in one big source tree (kinda like this)

If you were using Git (or Mercurial) in such a strict enterprise, most likely youd be ruling out the clone/pull from each other aspect of Git. Of course that would be a rule, as anyone could try to mount a share or open up SSH on their dev machines to subvert rules. Thats true for Svn, CVS, Static-Clearcase (and more) too.

FEBRUARY 6, 2010 1:25 PM

Cosmin Stejerean said...

Preventing developers from leaving the building with source code once it is checked out to their machines, or preventing them from accessing source code another developer checked out are issues that affect all source control systems equally.

Regarding local checkouts, the only difference I see between a source control system like Git and something like Subversion is that by default Git gives you a copy of the entire history locally, as opposed to only the latest revision. Lets ignore for a moment the fact that it is possible to get a copy of every revision from any version control system. What about those historical revisions makes stealing a Git repository more concerning than stealing the latest revision from any other repository?

Any enterprise concerned about people stealing source code or any other IP can take the same kind of measures to prevent theft, from severely restricting network communications to superglueing USB ports. Thats why I mentioned the issue of stealing source code is a red herring. It has nothing to do with Git.

restrict read access to release branches, although the easiest way of doing so is to separate release branches with sensitive information into a second repository that only select people can access.

FEBRUARY 6, 2010 8:25 PM

Lucas Ward said...

@Ketchup

But is looks a bit "biased" against dynamic ClearCase: This can be quite responsive, even for large installations. It can be done! :)

Since its centralized, having good infrastructure can help with some of the pain, but it will never be as fast as something on a local hard drive. Furthermore, its not atomic, which leads to all kinds of other errors. If you rename a file in your IDE, that rename is immediately committed, even if the change in files referencing this class arent, which leads to a broken build. Check-ins arent atomic either, so if there is an issue with one file, for whatever reason, you just broke the build. Theres also the file locking. Two developers cant work on the same file at the same time without branching and a huge messy merge after the fact. Theres also the slowness youll encounter when you need to do anything more trivial than simple check-in and check-outs, which is why the admin to developer ratio is so high. Even if your network is responsive, your productivity will be lower.

So, I wouldnt say that the article was biased against ClearCase dynamic mode, but rather the things we thought were important in an SCM naturally push ClearCase dynamic lower, although, Im not sure what features others would find more important that would push dynamic mode up. I would be interested in hearing them though.

FEBRUARY 8, 2010 2:40 PM

ketchup said...

@Lucas,

sorry, my "wording" was not correct: "Biased" might be too harsh, and I dont want to be rude on your blog. Please contribute that lapse to my imperfect and non-native English language skill. And yes, to begin with, ClearCase, as *any* CSCM, is no longer cutting edge. I totally agree. But let me correct a thing or two.

- CC has atomic check-ins, but it has no change sets. You might be able to emulate change sets with labels, but thats hardly the same. Atomic check-ins just make sure a failed check-in wont ruin the repository (as it was possible in CVS), and does not help code integrity.

- Renaming a file in CC checks out the parent folder element *and leaves in checked out*. If your IDE checks it in immediately, your IDE is broken: Usually you have to check in the folder element yourself. (Actually, thats the coolest feature CC has, if you employ it correctly.)

merge if two developers concurrently worked on the same object.

- Calling an admin for more complicated tasks than checking in or out is ... a myth. Whats more complicated than check-in/check-out? Merge?!? A developer is far better eqipped to solve merge conflicts than a CC admin ever could (which might be the guy who knows how to handle an Oracle, but not your code). You just need to read the manual. I know, thats hardly done nowadays, but then again CC is a dinosaur. Be brave! Hit F1 once in a while!

So, while Im still with you in any practical term, I find the objections against CC mostly grown out of buggy tools and misinformation (for lack of a better word - please refer to the first paragraph). Just one favour? Please, dont tell me the speed of your SCM is lowering your productivity, lest Im compelled to ask how you measure productivity. ;)

Anyway, I should be glad as long as git is coming out of that race of yours first place. You know, in git, the developers *are* the SCM admins, so they all must learn how SCM works. Otherwise they simply cant work.

FEBRUARY 9, 2010 2:12 PM

James Sears said...

Your enthusiasm doesnt mask your inexperience in using some of these tools.

FEBRUARY 10, 2010 8:11 AM

Paul Hammant said...

@ketchup. Lots to respond to.

ClearCase admin:dev ratio (IBM has at one time or another recommended 1:20). AFAIK admins are not for merging exclusively, though they are known to get involved. Youre right - thats a developer duty in a good team. Admins are for tricky stuff like (but not exclusively) repo repair, and branch creation.

Youre point about working copy (SCMs other than dynamic-CC) differ greatly in merge capability. With Git and Mercurial, their merge-through-rename works incredibly well. An Agile team thats doing lots of refactoring need not instigate as much communication around such checkins. For P4/Svn and anything else, much conversation is needed - "hey everyone, I want to move ShoppingCart to an new package, any objections?". Someone with WIP on ShoppingCart is going to ask you to wait until theyve checked in. Even Dynamic CC has downstream consequences for merge pain, after Agile style refactorings.

In terms of performance. Lets take the high bar for one operation. Say you want to sync/update from some canonical repo. Say there are actually no changes to come since your last sync/update. Perforce will take one second to tell you "already up to date". Static ClearCase (lets not mention PVCS Dimensions) can take 30 mins. Dynamic ClearCase makes you pay by in build and IDE minutes to make that sync/update look cost-free. Its a lie though. There is no Agile team on earth that is at anything even close to 2/3 effectiveness using Dynamic ClearCase, and no Agile

FEBRUARY 10, 2010 12:15 PM

Lucas Ward said...

@ketchup

"sorry, my "wording" was not correct: "Biased" might be too harsh,..."

No worries. Im currently learning French, and fully understand how confusing the small differences in meaning between words can be.

"- CC has atomic check-ins, but it has no change sets. You might be able to emulate change sets with labels, but thats hardly the same. Atomic check-ins just make sure a failed check-in wont ruin the repository (as it was possible in CVS), and does not help code integrity."

This is somewhat of a semantics argument. What I meant by atomic, was that if you checked in 10 files, and have some kind of issue with one, none of them should be checked in. I suppose we could call this atomic changelists or something, but its beside the point.

"- Renaming a file in CC checks out the parent folder element and leaves in checked out. If your IDE checks it in immediately, your IDE is broken:..."

As far as Ive ever known, there has only really been one clearcase IDE, RAD. At least thats where I used it. I know theres plugins for other IDEs, but I would be a bit scared to use them. I agree that the file will still be checked out. However, the file will still be considered renamed regardless. If someone else does an update, and you havent checked in the files that required renaming as part of the refactoring, which you definitely wont, then the build is broken until you check in the result of the refactoring. Whenever I was working with CC I would always do renames first and check in immediately to avoid this issue. Its not the end of the world, but still annoying.

"-... CC has private branches. And: You will always have to merge if two developers concurrently worked on the same object..."

Personally, thats a problem for me. This ultimately leads into proliferation of branching and away from trunk based development. Although, the benefits of TBD are a completely separate discussion.

"- Calling an admin for more complicated tasks than checking in or out is ... a myth. ..."

On the previous CC projects Ive worked, I always seemed to have some kind of scenario where I had to call in an admin. I suppose it wasnt always simple merges, but something seemed to always get funky, leading to me getting out of the chair and hearing faint mumbling about config specs. :)

"... I find the objections against CC mostly grown out of buggy tools and

I dont want to make offense, but this is the argument that always comes up when discussing ClearCase that bothers me. It basically boils down to: "You must be using it wrong!" Youre right, RAD is buggy, but considering who creates it, I think we can consider it part of ClearCase, its all the same suite after-all.

I also completely disagree with your productivity statement. Everytime a build is accidentally broken because of the tool (even the IDE you need to work with to use it), productivity is lost. Everytime you have to call an admin over to look at your config spec, productivity is lost. Its not necessarily the speed of the SCM, since if thats a problem, even with clearcase, its probably network related. Even SVN or Git can have the same issues. I will say though, I have never been on a CC project, even one where day to day usage was relatively zippy, where rebaselining wasnt measured in hours, and dont get me started on ClearQuest :)

Thanks for the well thought out comments. Having people poke holes in arguments is the only way to improve them, and improve your thinking on them.

FEBRUARY 10, 2010 3:24 PM

ketchup said...

@Paul,
yes, my point: Admins should not merge. Merging code is essentially coding.

The merge-throu-rename-capabiblity of CC made it stand out in its time, as well as its support for massive parallel branches. But I suppose Agile and massive parallel works only with short-lived branches. Maybe the "content tracking" of modern DSCM helps, but, honestly, I dont know.

(The SCM problems are my main, if not only, objections against Agile. Glad to hear it works well with git and hg, have to try that in my next project.)

Finally, I agree dynamic CC probably spent even more network time than traditional working copy models. It is supposed to hide that from user by updating when the system is idle, which is probably why it never bothered me. (Never? No, thats a lie: I remember a project where I went ballistic because I had to wait for a compile for 45min, just because our project site was connected with 64kbit with the CC system. *Thats* pain.)

But Id like to think Linus was right as he said having a very very fast SCM changes the way we use the SCM. So dont take my comments on CC speed as an glorious aprisal, its a simple statement that dynamic CC can be setup in a way which does not interfere too much in traditional software development. It cant support the way Id like to work today anymore.

(Actually, thinking back, the highest risk in using dynamic CC is that nobody can work anymore if the CC server is down, with all the dire business implications.)

@Lucas,

please dont start on ClearQuest. Id rather forget about that. :)

I used the CC plugin fo Eclipse some years ago. I cant remember exactly, but it might have had that strange behaviour already. If so, its probably a weakness in the relevant Eclipse API, which might not support checking in and out folders, so they have to checkin the modified folder immediately (just my guess). That would render an otherwise great CC feature useless.

If it really matters, you can verify the correct behaviour in the CC Excplorer. Then again, its purely academic, since you need to use the refactoring tools in you IDE, which obviously behaves wrong.

So I must accept the rename behavior is broken, at least for Java developers (which are possibly the majority of CC users nowadays?). I already lamented the missing change sets, so theres nothing to discuss left here. :)

Im not sure about the branching argument. I like branches, I think CC and git do it right, while CVS and SVN are wrongly pretend theres no branching involved, even if every working copy effectively is a branch. However, I agree that you need short-lived branches for this to work well. Maybe you need to check on the private branches; I have no experience with them.

Finally about config-specs: Ah, I completely forgot about them! Fine stuff, esotheric, perfectly useable to confuse co-workers and drive developers crazy! ;)

In fact, I think config-specs are build- and configuration management responsibility, since they are part of the CC branching scheme. If a developer has to handle them, he does a job hes not paid for, hence he looses productivity. I stand corrected.

P.S.: Sometime you do something wrong, because you dont know better, sometime, because you must. With CC, at some point, its usually because you must. And Im not even telling you youre doing something wrong. Except, of course, if you use a 64kbit line and developers have to maintain config-specs, that would be wrong. :)

FEBRUARY 10, 2010 8:55 PM

Ran said...

One thing missing in this maturity model is how these models support continuous integration. I have found out that using SVN is good step in practising CI because it has only one common repository and merging takes effort so people tend to move towards small commits.

FEBRUARY 17, 2010 6:59 AM

Michele said...

A problem with Git?

MARCH 15, 2010 7:32 AM

ketchup said...

@Michele

I just tried to provoke the problem you described and failed: Git either correctly auto-merged the file (when no conflict araised), or flagged a conflict and did not merge. Then again, Im using git a lot, and this problem never occured to me.

If you feel you experienced a bug, you should:
0. make sure its a bug, not a feature, just to save you lots of time and trouble
1. write down how to reproduce the behaviour in detail
2. visit http://git-scm.com/ and make sure its gits fault, either intentionally or not
3. contact the developers, see http://git-scm.com/

Point 0. is very very important, especially when you plan point fingers at git (as your statement seems to imply). ;)

You might be able to find a support shortcut with some heavy git users (e.g. Github), but beware to assume they provide universal git support for free. :)

MARCH 15, 2010 8:15 AM

Michele said...

Absolutely no fingerpointing. I really want to understand how this works. After I held a presentation at my work about Git - since many developers here are not happy with our current system (MKS) - a developer challenged me with this problem. MKS just crashes. I tried it several times - both on OSX and on Windows XP (mysysgit) - and always with the same result. I am curious to know exactly what you did. I have asked Scott Chacon directly, because I didnt find a good place to present the problem.

Ill describe the three scenarios Ive used. I always start with a clean directory.

#Initialize
git init
touch test
"edit test"
git add test
git commit -m "first commit"
git checkout -b nbr #new branch

#Scenario 1 - rename file in one branch
git mv test test_nbr
"edit test" # Doesnt really make a difference
git commit -am"renamed to test_nbr"
git checkout master
git merge nbr

#Scenario 2 - rename file in both branches
git mv test test_nbr
git commit -am"renamed to test_nbr"
git checkout master
git mv test test_master
git commit -am"renamed to test_master"
git merge nbr
#Result in master: Conflict. I dont understand why, but at least Git indicates that a file has been renamed.

#Scenario 3 - rename and edit file in both branches
git mv test test_nbr
"edit test_nbr"
git commit -am"renamed to test_nbr and edited"
git checkout master
git mv test test_master
"edit test_master"
git commit -am"renamed to test_master and edited"
git merge nbr
#Result in master: Two files, test_master and test_nbr, and no conflict.

MARCH 16, 2010 1:52 AM

ketchup said...

@Michele

Sorry if I appeared unfriendly, it was completely unintentionally. I just feel the right place for git support is not exactly the comment section of Lucas blog (but thanks to Lucas to endure us!).

In short: Git does not track files, it tracks content. Imagine all you files punched into a flat file with a line indicating the original filename, then diff it. This way git can find a given text blob and track the changes to the text across file boundaries. It should explain the conflict in scenario 2.

As for scenario 3, Im not really sure why it works. Im not really sure if it should work or not, but it emphasizes my opinion that no technical solution can ever help a team of developers if they dont talk to each other (one way or other): If you start a refactoring, tell it to your mates. :)

But maybe you really should dig into http://git-scm.com for a authorative answer. These guys not only built git, they *know* git. ;)

MARCH 16, 2010 3:02 AM

Michele said...

Its ok. I will pursue this, since I have to "defend" Git where I work. Because of the content vs file, I also guessed this is the reason of the conflict in scenario 2, but I

ketchup said...

@Michele

Scenario 2 raises a conflict since the "line" holding the filename was changed to "test_master" on branch master and to "test_nbr" on branch test_nbr. Same line, concurrent changes, so its a conflict. At least thats how I would explain it. :)

I tested your scenarios 2 and 3 on Cygwin 1.7.1 with git 1.6.6.1. I could run the scenarios on some more OS including Solaris and AIX, but as I understand it, the OS should not have any influence on the way changes and conflicts are detected (except maybe for POSIX file attributes), since only the *content* of the files are considered.

MARCH 16, 2010 6:22 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

LUCAS WARD'S BLOG

FRIDAY, FEBRUARY 19, 2010

SCM Continued

In my last blog post I talked about a maturity model for source control systems. Martin Fowler has recently posted a blog entry that covers the same topic:

http://martinfowler.com/bliki/VersionControlTools.html

One of the reasons that Paul and I thought a maturity model for SCM was interesting, was that it brought objectivity to the debate.(or at least tried to) It seems far too subjective (and easy) to break SCM systems down into two categories: usable and unusable. For me personally, I would list ClearCase in the 'unusable' category. However, despite it's shortcomings, thousands of programmers use it on a daily basis. Therefore, it can't be 'unusable'. What it boils down to though, is that for me personally, given what I find important when developing, ClearCase is a tool that I feel prevents me from coding effectively. If someone else doesn't hold these same values to be important, perhaps it changes what they view to be usable.

Martin takes an interesting twist with his article, in that while he still uses two groups, his dividing line is on 'recommendability'. This is still a somewhat subjective measurement, especially since his focus group was us ThoughtWorkers. But after reading the responses to my last blog post, a common theme seemed to come from those using ClearCase. They were willing to defend it as 'usable', but not a single one that I read would recommend it. This mirrors some of my project experience as well. I've even known some ClearCase admins that would argue with me at length about how ClearCase isn't that bad, as long as you 'used it right', and yet even they wouldn't recommend it as a source control product. Perhaps it's much more useful to ask a client if they would recommend their SCM to others, rather than arguing about why you don't think it's the best solution. Afterall, if they wouldn't recommend it to others, why are they still using it?

POSTED BY LUCAS WARD AT 4:27 PM

0 COMMENTS:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

ABOUT ME

LUCAS WARD

I'm a Consultant at ThoughtWorks. I code primarily in Java. I co-led the Spring Batch project.

VIEW MY COMPLETE PROFILE

OPEN SOURCE PROJECTS I AM A COMMITTER ON

Spring Batch

SOCIABLE

Not Found

Error 404

BLOG ARCHIVE