What's next in Version Control for SAS programming environments?

David Garbutt

NNIT AG, Zurich, Switzerland

Since 2004 Version Control systems have become recognised as essential for SAS® programmers working in medium- to large-scale environments and have been widely implemented as well as being included in environments such as Macumba, EntimICE® (R) and SAS® Drug Development. But what next? What is new in the computing world and the world of version control? What are the essential drivers of change? Where can, and will, we be in ten years time? This paper presents a view of the future that is nearly here.

In this paper I explore some achievements and trends in the area of SAS programming environments, or IDEs. In particular those created for use in the Pharmaceutical industry.

1 Version control in Statistical Programming

The industry came late to the benefits of Version Control and this partly due to the special needs of SAS programming Garbutt (2005, 2006): SAS programming in Pharma is a little different from most of the software industry, our product is not the same as the software industry's, and the fact that we need to control binary objects (e.g. SAS datasets) as well as directory structures, programs, and outputs also makes a difference. Indeed, most available systems did not support binary objects until recently.

A key issue is dependency tracking because of the FDA's re-creation rule. Dependency tracking allows re-creation because all the components of an output can be traced, and labelled, and therefore later re-assembled. However dependencies encompass a wider range of file types than is normal when programming. Other critical differences and the distributed nature of programming teams meant that many VC tools were not suitable for Statistical Programming. Because we have the 'press red button, once' model for deliveries Garbutt (2006) we must invest a lot in methods for running large batches of jobs (ca. 400 outputs per study is not unusual) efficiently and using only validated sources. And, typically, the tools developed long ago by the software engineer-

ing community are ignored Garbutt (2005).

1.1 Custom built systems 1998-present

1.1.1 GPS II and build control

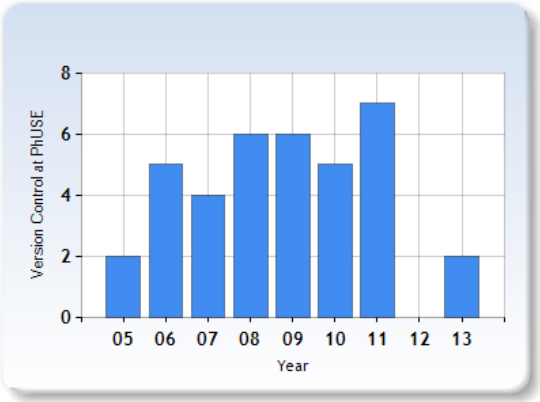
Ten years ago I was working long hours on rolling out a world-wide SAS Programming environment with version control, and what would be called build control, built with IBM Rational ClearCase®. The system is still in use with over 900 users worldwide, typically 100–200 online, and running 3,500 SAS batch jobs a day. As of 2010 it held about 2.5 TB of SAS data and was growing at about 10% a year.

GPS II was designed not only to add VC to our armoury[1], but also to accomplish better build control for outputs and to support business processes and constraints. For example, only outputs created from validated components[2] could be sent to the Documentum system. This approach I called the walled garden because there is easy development inside but strict supply-chain checks at the borders.

This was possible to do with ClearCase because it supports dependency detection[3](DD) and event-driven hooks for customisation that can implement business rules[4] to police check-ins. GPS II left the GUI for the user to choose, several editors[5] were supported and access could be via Windows or via X.

1.2 Mainstream Commercial systems 2006-present

Since around 2006 commercial systems for SAS development with version control and with IDE-like GUIs have become commercially available. The two leading systems are SAS® DD which is now a hosted environment, and ®EntimICE® from Entimo AG. There is also Macumba, developed at Bayer for internal use Weiss (2012) which has perhaps the best GUI for SAS programming ever created. Most of these systems are now using SVN instead of CVS to provide versioning. But all are still centralised systems of the client/server generation.



'Version Control' in PhUSE papers.

[1] The previous system emulated version control by moving files between three identical file trees (with all projects) under devl/... test/... and prod/... and the system relied on copying files if older versions were to be kept.

[2] 'Components' include programs, the macros they call, and the data they use. For final delivery data must originate from a locked database.

[3] see Garbutt (2005) for an screenshot of a Configuration Record. ClearCase does this by implementing on the OS level a version aware file system. This is one of three ways to implement DD. The second way to is add code to the software itself like SAS DD which uses the SAS option RTrace to monitor file opens executed by SAS code. The third is to emulate the file structure within the application itself like entimICE does.

[4] See Diebolt (2003) for some examples of how coding triggers can change the behaviour of the system. One of the GPS II triggers prevent a user checking in a version of a program if that user was in the history of the QC twin program.

[5] Including ESS `http://ess.r-project.org/` and SlickEdit `http://www.slickedit.com/`

1.3 Limitations

It seems strange to talk about limitations, SDD has hundreds of customers, entimICE is growing rapidly, and GPS II is still running, but there are problems, and they are structural in nature.

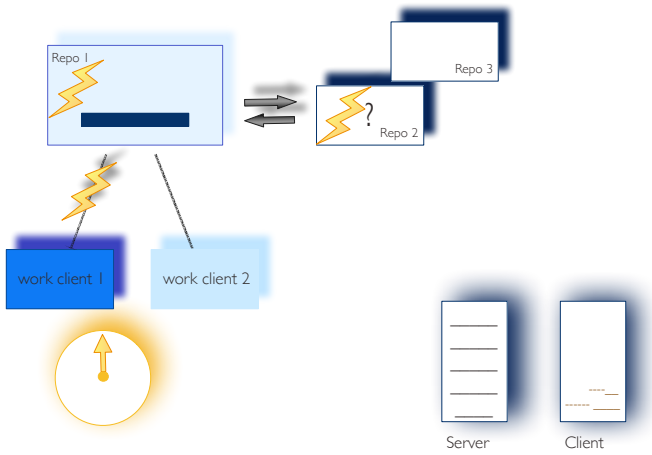


Figure 1: Centralised Version control schematic

Cost and scope Centralised systems require big resources an all the costs and testing that goes with it. Achieving acceptable response times is a constant effort. Centralised systems are viable for groups of 100 programmers or more. Below that their value is a harder sell, although still justified from the programmer's viewpoint. The result is that although there are free VC tools the adoption rate for small companies is not high and many shops still rely on file shares to get through the red-button days.

Final run durations on large servers are *not* dramatically faster than tricked-out PCs. Consequently a small team can create a final run on their own PC's faster. Using grids to automatically distribute jobs to clusters of similar machines is an active area of development right now.

Pharma teams are very globalised and the centralised model of VC that has been implemented does not handle synchronisation across sites cleanly[6]. Consequently systems rely on Citrix to deliver the experience to users not on-site. This adds another layer of configuration optimisation that is needed.

Bug Tracking systems are under-utilised and could be very productive if used to cover report deviations from specifications in addition to true bugs. Since such issues could be results of data

[6] Not as cleanly as Dropbox does, for example

problems it should be possible to integrate such reports into data management procedures as well.

Documentation is often not in the same system, let alone not version controlled, it is often not machine readable and changes cannot easily be linked to the code changes needed or the correct output version.

What developments might address these problems?

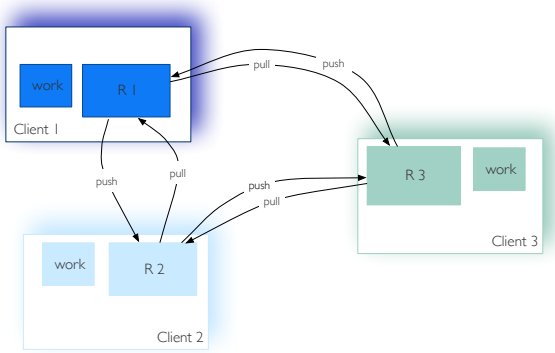


Figure 2: Distributed (symmetric) Version Control schematic

2 Drivers of change

The IT field is perceived by many outsiders as changing rapidly but, as we know, the real story is different. The design problems faced when building applications now are the same as faced in the 70's — only the cut-off points have changed. There was, in other words, always big data, that had to be handled differently but now the boundary is petabytes, in the 80's it was Megabytes. The cut-off is determined by a combination of the available RAM, the available disk space, and to lesser extent the CPU power.

2.1 Power and the balance of power

The key characteristics are first, the balance of power (and cost) between the desktop computer and the remote one, whether in the cloud or in the server room that used to be the broom cupboard. Second is the speed and cost of transferring bits between the two.

But this is not the whole story because increasing power for any given application brings diminishing returns.

An example is word processing which is based on processing bytes This was something which even early 16-bit PCs in 1981 like the Sirius[7] had enough power for. Systems at that time could keep up with an average touch-typist (100 w.p.m.) and this was the

[7] or Victor 9000 http://en.wikipedia.org/wiki/Sirius_Systems_Technology

[8] believe it or not there were mainframe programs for word-processing, or, more properly, text processing. See http://en.wikipedia.org/wiki/SCRIPT_(markup) for an example related to SGML an ancestor of HTML.

point they became mainstream. The next borders were crossed in the 90's for audio and the 2000's for video.

As the local CPU power becomes sufficient[8] and the network can transfer the data within seconds the balance swings in favour of using the local computer.

How is the arrow pointing now?

CPU The infamous rise of CPU clock speed that drove so many PC purchases has now halted with the limit around the 3GHz mark. Since 2005 changes have been focussed on reducing power consumption and adding processors. The current phase is characterised by a new pattern the desktop *is* as powerful as an average server CPU. Under these conditions power is increased by adding processors, not by buying larger and more powerful ones.

Network speeds have increased in the last ten years, with many offices running Gigabit ethernet and Internet connections at 100 Mbps or even higher. This trend plus the creation of new screen-oriented protocols make a remote desktop a real possibility. A single user experience but delivered from a machine across the network.

Disk Accesses 'Fast' disk access times have gone from 17 msec to around 8 msec, in 30 years; in other words they have not changed much especially compared to CPU speed. Technology *has* added speed by exploiting caching and multiplicity. However the rise of the SSD large enough to replace an HDD has made a leap in performance possible. This will be fully realised as drivers are adapted for the different characteristics of SSDs and computers are designed with faster data buses.

Distributed systems for version control or databases are made practical by increasing network speeds and the wider availability of high-speed internet access combined with the high-enough power of today's desktops and the low prices of commodity hardware.

Automation and Process Integration our work is made up of processes we do, and once they get repetitive and frequently needed we start to automate them using programs or other methods. This automation drives better integration between different applications and different spheres of work. Recent products such as Kiln `http://www.fogcreek.com/kiln/features/team-up/` show companies are addressing these issues.

2.2 Distributed Version Control systems

DVC systems were developed in the early 2000's as people started looking for systems that could work offline. Git and Mercurial have become the best known although they were not the first. Both are now supported by good communities, free hosting sites for repositories (github, bitlocker), GUI tools and lots of tutorials[9]. A DVCS has no central repository[10], there is no server you *must* connect to, all repositories have a complete copy and therefore act as back-ups for

[9] for example on Git here From Subversion to Git

[10] SVN and Perforce support federated servers, ClearCase supports replicated servers but this is not the same, some nodes are privileged over others.

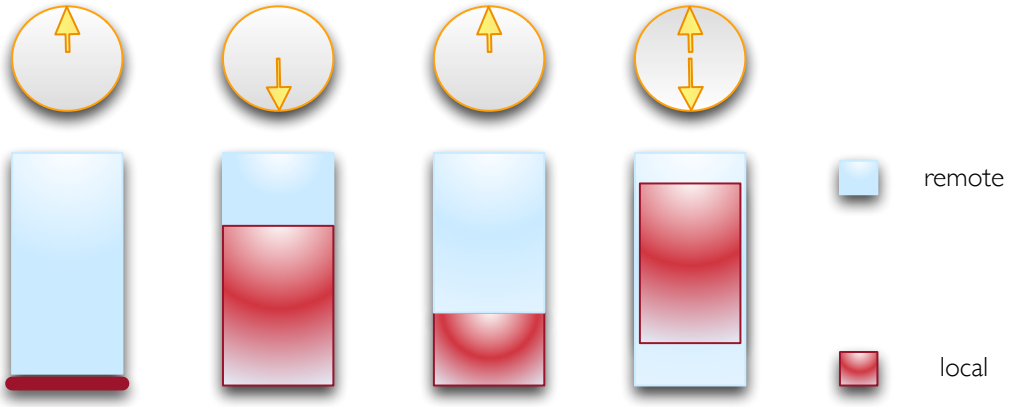


Figure 3: Balance of power/cost between remote and local computers. rome left to right: the 70's, 80's, 90's & '00s Web/Citrix, and on the right the next step: all local power delivered remotely.

each other. DVCS are designed around the branch model and the unit they work with is a commit of a group of files (or a changeset). This fits very well with SAS programming systems because changes to a report for example, rarely involve only one file. In the earlier designs changes were to a file and groups of changes something that needed work to keep together. Later design reverses this concept and makes a set of file changes the central object.

As can be seen in Figure 4 on the following page DVC systems use more local power both for storage and for processing (note the different locations for the branch, merge and commit commands). The local repository also solves the problem of many centralised systems —a check-out is needed to be able to read a file— i.e. to have a local copy. ClearCase does not have that problem[11] — its problem is getting fast access to a client across a network. It works best on the same subnet or with high-capacity, low latency network in place.

It is increasingly the norm to use bug tracking integrated with the Version Control system. These system are not distributed but are still centralised (Atlassian Jira for example). The next wave of VC systems will be better integrated with supporting systems. An example is Veracity (2011) which includes bug tracking, a distributed documentation wiki and support for sprints, time recording and Burn-down charts[12].

[11] in ClearCase all of a repository is visible in a dynamic view, but is read-only until a checkout is made, in SVN and other systems a checkout must be made to get a local copy of repository files. This is a significant disadvantage for statistical programming where a program my need read access to a large number of files. Expecting the users to manage these dependencies manually themselves is a non-starter.

[12] For a feature comparison table see http://veracity-scm.com/compare/

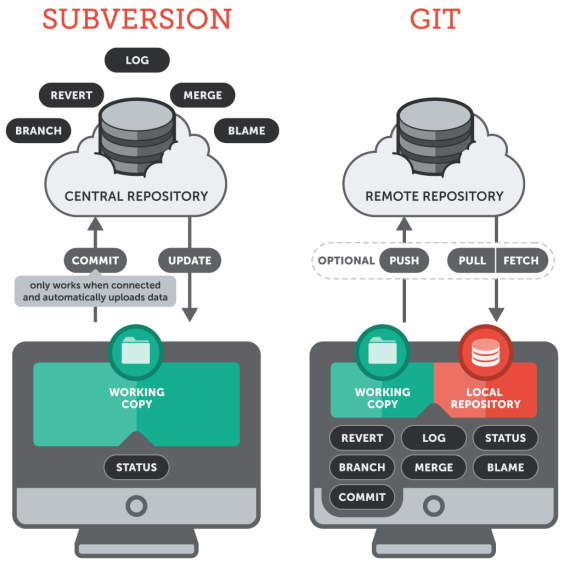


Figure 4: Diagram showing design differences of Subversion and Git

2.3 Distributed Databases

The final piece of this puzzle is the rise of NoSQL databases. These systems have different design goals from relational systems, some do not even have ACID consistency guarantees. But they do have scalability (to billions of operations a day) and several aim to provide seamless, configuration-free replication[13]. This latter property makes them ideal for our purposes because they can easily fit on to our node-based model. This makes them an interesting option for working with clinical data both and metadata. One reason is their schema-free nature which makes them suited for storing similar data that may have differing structures, such as can be found when pooling studies.

The use envisaged here is perhaps with metadata, summary information, build status indicators, or other datasets that need to be written by all team members. Text files of titles and listings, might be another example.

2.4 Virtualisation

The ability to run whole machines (operating systems with installed applications and data) inside another has been around since the 70's[14] becoming widely available in the 80's on IBM 370 architecture mainframes. VMware and Parallels have now made this a useful technology on desktops. A new variation called 'containerisation' is now becoming available which runs individual applications inside the Virtual Box and thus makes the host OS also inside a virtual environment. This has obvious benefits for maintainability.

[13] Redis and Cassandra are two
[14] VM/CMS Wikipedia: VM/CMS and at IBM

3 And now…

3.1 The All-in-One client-grid

If we put together the trends then there is a way ahead to solve some longstanding problems with current systems. We exploit the power available on today's cheap desktops by building everything locally. All shared storage is versioned, so there are fewer problems with lost or divergent files.

First, we can build a peer-to-peer system where all the components are on one client computer but resources are still linked and shared.

- a DVCS there will be a shared repository per project member. A server installation will host a copy of every repository, thus keeping a backup.
- A NoSQL auto-replicating DB for metadata and other shared information.
- SAS. JMP and/or R installation .
- auto-build system.

Installing distributed components on a single client is a simpler way to proceed because the complexities of client server installations are not needed. Also the costs are incremental scaling linearly with the number of programmers. Because no large servers are needed costs are low to start. This architecture can also support a grid for parallel execution.

On each grid machine the DVCS repository is made available, and before a job is received for execution the appropriate branch and tagset is set. At the end of the job the results are pushed if successful and become available on each client.

For the leading IDEs available this is also a viable underlying architecture.

The advantage of building an all-in-one client in this way is that it is simpler and cheaper for small to medium sized clients. It should also allow direct client level access speeds to files and performance. As we have seen this gives comparable performance to large hosted servers but with the immediacy (and flexibility) of local access for every team member.

3.2 Virtualised

This working system can also be virtualised making it cheaper on the scale on hundreds of users. It could possibly be implemented on a cloud computing platform. The same changes also mean the system also becomes easier to virtualise. This is interesting because of the economies of scale and the possibility to host whole systems as services. In principle it would also be possible to use the NNIT GxP cloud services[15] to host instances.

These new services bring proper SAS with Version Control to smaller companies and can also really change how people work in and CROs and larger Pharma companies by reversing the trend away from the desktop but without the costs and other disadvantages of previous devolved systems.

[15] NNIT GxP Cloud services

4 Overview and looking forward

Integrated, distributed, systems are becoming the norm and the challenge is now to provide that for small and/or geographically distributed teams cost-effectively. We have outlined a way to make that work, so let us hope that is what is next in Version Control for SAS systems.

Further Reading

The paper Fowler & Sadalage (2012) outlines how NoSQL DBs are different from relational ones and gives example implementations with links.

A good survey of Version Control which mentions most systems (although not DVCS) is Shannon (2009) and Mengelbier (2012) discusses Subversion and includes useful code for extracting metadata from SVN to use directly in SAS programs.

There are excellent guides to DVCS http://betterexplained.com/articles/intro-to-distributed-versio with lots of diagrams illustrating the processes and links to further reading. Eric Sink's book Sink (2011) examines many work scenarios and then shows how they would be handled with each of the major tools and thus covering both concepts and practical details.

The papers by Rajagopal (2007) and Fang (et al) discusses why dependency detection is important and useful, some approaches via code, and implicitly, why it should be automated. Alex (2008) discusses the SAS RTRACE feature and how to extract information from its logs. Note that this feature has limitations mentioned in the documentation. It only can tell the file name and access time, meaning the version number must be looked up outside the SAS system to be able to tag or label used objects from a particular run. This is a non-trivial amount of processing.

References

Alex, J, *The History of How, Who, Which, What, Where – Utilizing the SAS ® Trace Module!*, 2008, NESUG Proceedings Paper CC15 http://www.lexjansen.com/nesug/nesug08/cc/cc15.pdf 9

Diebolt, D, *ClearCase: The ten best triggers*, 2003, http://www.ibm.com/developerworks/rational/library/4311.html 2

Fang, J, Sharma, A, Narayanan, V, Dunleavy, P, *How is your I/O Logic? A primer on understanding program dependencies in large SAS ® implementations* , 2006, NESUG Proceedings www.lexjansen.com/nesug/nesug06/po/po15.pdf 9

Fowler, M, Sadalage, P, The future is Polyglot Persistence, http://martinfowler.com/articles/nosql-intro-original.pdf 9

Garbutt, DJ, *Implementing CFR 21 part 11 for SAS without tears or joins* , 2005, PhUSE Proceedings http://www.lexjansen.com/phuse/2005/as/as09.pdf 1, 2

Garbutt, DJ, *Understanding what 'make' can do for you*, 2005, Phuse Proceedings http://www.lexjansen.com/phuse/2005/ps/ps27.pdf 2

Garbutt, DJ, *Why Safety Reporting IS like Rocket Science*, 2006, PhUSE Proceedings http://www.lexjansen.com/phuse/2006/ad/AD13.pdf 1

Mengelbier, M, 2012, *Simple Version Control of SAS® Programs and SAS Data Sets*, SAS Global Forum Proceedings http://support.sas.com/resources/papers/proceedings12/365-2012.pdf 9

Rajagopal, G, *Link the Unlinked : Documenting Dependencies Among SAS Programs in a Project*, 2007, PharmaSUG proceedings Paper CC02: http://www.lexjansen.com/pharmasug/2007/cc/CC02.pdf 9

Shannon, D, *Solving the Version Control Issue*, 2009, PhUSE Proceedings Paper http://www.lexjansen.com/phuse/2009/ts/TS04.pdf 9

Sink, E, *DVCS and DAGs,* 2009, http://ericsink.com/entries/dvcs_dag_1.html. Printed book also available.

Sink, E, *Version Control by Example*, 2011, web version: http://ericsink.com/vcbe/ 9

Weiss, M, *MACUMBA – A Modern SAS GUI – Debugging Made Easy,* 2012, PhUSE Proceedings http://www.phusewiki.org/docs/2012/PAPERS/AD/AD01.pdf 2

Nomenclature

ACID	ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. (wikipedia)
DD.	Dependency Detection. A feature that allows the system when running a job or program to capture and record all items accessed fro reading or writing.
DVCS	Distributed Version Control System. Examples are git, Mercurial, bazaaar and Fossil
entimICE	The Entimo Integrated Clinical Environment from Entimo AG. http://www.entimo.com/solutions/statistical-computing-environment-sce/115-sce
HDD	Hard Disk Drive
NoSQL	Currently defined as Not only SQL is a relatively new class ofdatabase technology not using the relational model. The products falling under this sobriquet are diverse a list is available at http://nosql-database.org/. Google's Bigtable DB and Amazon's Dynamo were pioneering efforts in the area.
SDD	SAS / Drug Development. A product from the SAS Institute. http://www.sas.com/en_us/industry/life-sciences/drug-development.html

SSD solid state memory chip(s) used as an HDD

VC Version Control also called Source Control. Tools for assisting programmers to manage large code bases have existed now for 40 years and are nearly universally applied.

Contact information

I would value your comments and questions on this paper. Please contact me at:

David J. Garbutt, Senior Consultant, NNIT AG,
Bändliweg 20,
CH – 8048 Zürich
Switzerland
Work Phone: +41 79 774 28 50
Fax: +41 44 390 37 22
davg@nnit.com
LinkedIn profile

Acknowledgements

I would like to thank my colleague Leif Jordas at NNIT Switzerland for useful comments and for noticing my, now corrected, inadvertent promotion. I also wish to thank my family for their tolerance, support and ability to play.