Paper AD07

SAS in clinical trials – A relook at project management, tools and software engineering

Sameera Nandigama, inVentive Health Clinical, Maidenhead, UK

ABSTRACT

Traditionally, SAS® has been used as a standalone application for data analysis and projects are managed using spreadsheets and e-mails. Recent technological and software developments have now moved from a standalone application to an integrated statistical computing environment fitting into the workflow of a developer. However, there is a wide gap between the available tools for project management and those that are currently used. This gap is also visible in the design, development, testing and documentation of code in statistical programming. Hence, with the shift of the industry from a local application to a highly integrated and regulated software environment, the need for better use of available tools and refined processes is called for. In this paper, the author discusses tools such as JIRA and Doxygen and also processes such as Agile and Waterfall. All of which can improve control, efficiency and focused visibility at all levels.

INTRODUCTION

SAS programming in clinical trials largely has been taking a traditional approach to project management, the use of certain tools and the design process. Although the approach has served the industry well, this paper examines possible alternatives and the benefits they may bring.

PROJECT MANAGEMENT AND TOOLS

"Project management is the discipline of planning, organizing, motivating, and controlling resources to achieve specific goals"[1]. There are a number of approaches to project management such as the traditional approach using stages of initiation, planning, execution, monitoring and completion; process based management and agile project management amongst others.

SAS Project management in clinical trials is largely based on traditional methods. However, many new tools and approaches are available for project managers and developers that in spite of being popular in the traditional software development industry are not being widely used in SAS clinical programming.

Here we examine some concerns managers and developers have and tools that are available to solve these problems and any other benefits they could potentially bring.

TECHNICAL DEBTS AND WORKLOAD MANAGEMENT

Technical debt is the accumulation of small low priority tasks that needs to be completed for the work to be considered complete but is always pushed back for higher priority issues, eventually leading unbalanced workloads and pressures with the intent of reducing the technical debt.

Technical debt represents "things that are an effective and appropriate tactical and short-term choice but which we should put right in the longer-term in order to avoid specific risks or increasing costs (the interest on the debt)."[2]. Some examples of technical debt include but are not limited to:

- Incomplete error handling or not logging errors, leaving the possibility of bugs remaining in a delivered code.
- Temporarily hard-coding values.
- Using temporary or incomplete data to make progress with coding.
- Making early releases that does not fully meet all requirements.

Technical debts are a reality of life and help people move further into projects in face of temporary constraints. Generally, it is a temporary fix or shortcut that is taken immediately to deal with an issue that is blocking the project from moving further. Replacing the shortcut with a proper solution itself becomes a low-priority task for the future.

A project, over a period of time has a potential to accumulate huge technical debt resulting in a large volume of low priority work that as a whole becomes a risk to the project itself.

The key to manage tech-debts is:

- To ensure that the accumulated debt is effectively tracked
- To ensure that some amount of small low priority tasks are continually disposed of.
- To ensure that the backlog of work, irrespective of priorities, remains manageable.

There is a natural tendency in project management to focus on high-priority tasks all the time leading to the build-up of technical debt. High priority tasks do not need much effort to track, it's the technical debt that generally ends up being difficult to manage.

And in this context, it is useful to know and understand some popular issue management tools widely in use.

ISSUE MANAGEMENT TOOLS – JIRA

"An issue tracking system (or ITS, trouble ticket system, support ticket, request management or incident ticket system) is a computer software package that manages and maintains lists of issues, as needed by an organization"[3]. Specifically from a SAS project manager's point of view, we are talking about the tracking of tasks within a project. There are many tools available for this purpose starting with good old spreadsheets.

Spreadsheets are still widely in use today but may not be best suited for the purpose for many reasons:

- Accidental changes can happen while using spreadsheets and in general can be error-prone.
- Spreadsheets do not scale well. They work well if entries range from tens to early hundreds. The use of spreadsheets becomes increasingly complex with more and more entries.
- It is difficult to search for particular issues in a large spreadsheet.
- The history of changes is not easily preserved on a spreadsheet.
- Report generation with spreadsheets can often be clumsy.
- Visibility management is not possible with a spreadsheet

There are many dedicated tools available for issue management such as Redmine, Bugzilla, IBM Rational ClearQuest and JIRA. JIRA is one of the most popular tools available and we will use this as an example. JIRA not only supports issue management but also has features to aid project management.

In an issue tracking system like JIRA, users are arranged into groups. Privileges are assigned based on the user group an individual belongs to. The groups could be Project manager, Team lead, Developers, Scrum master, Testers or Statisticians.

The issues are generally categorized as:

- Tasks
- Bugs
- Support

JIRA supports custom categories based on the specific project management approach and can be adapted for the company's needs. Each issue has a priority attached to it, low, medium or high. Again there could be custom priorities. For the sake of visibility, each issue belongs to one component or more. Issues can also have one or more labels attached to it. All these features make searching for issues very easy. There is also support for raw text based searches.

Workflow could be built into the system, for example:

1. The project manager creates a JIRA ticket and assigns it to a SAS developer as high-priority.
2. The developer immediately gets a notification of new issue or he sees an update in his JIRA login.
3. The developer marks his current low priority task as "blocked" for new high priority task and marks the new task as "in-progress". See *figure 1.*
4. The developer understands the new issue and updates the comments and time estimates for the tasks.
5. The project manager, executives and the client who all are watching this high priority ticket are all getting updates about developer's initial assessment and time estimates.
6. The client also makes comments on the ticket to help developer understand the issue better.
7. The developer fixes the issue and marks it as fixed and the ticket is automatically assigned to a reviewer.
8. The reviewer notices that the new ticket is high priority and reviews the code immediately and makes his comments.
9. The developer addresses the comments and the ticket goes back to reviewer. The reviewer accepts the ticket which then is automatically assigned to a tester.
10. The tester quickly tests the report and is happy about the result and closes the ticket.
11. Everyone from the executives to testers have visibility that the high priority issue has been resolved.

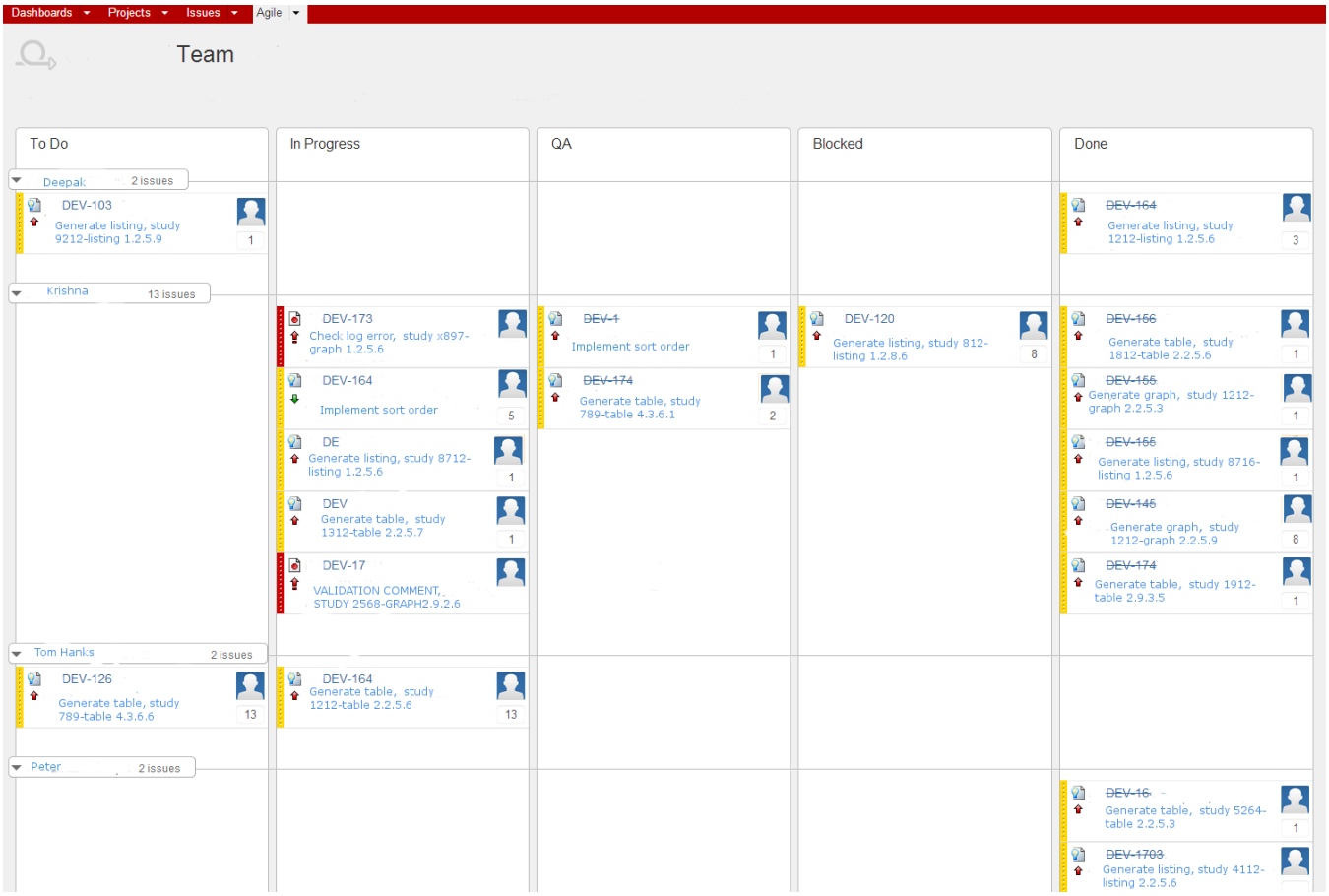

Figure 1: Work flow in JIRA

A complete history of comments and discussions are preserved on all tickets for future use. For example please see *figure 2.*

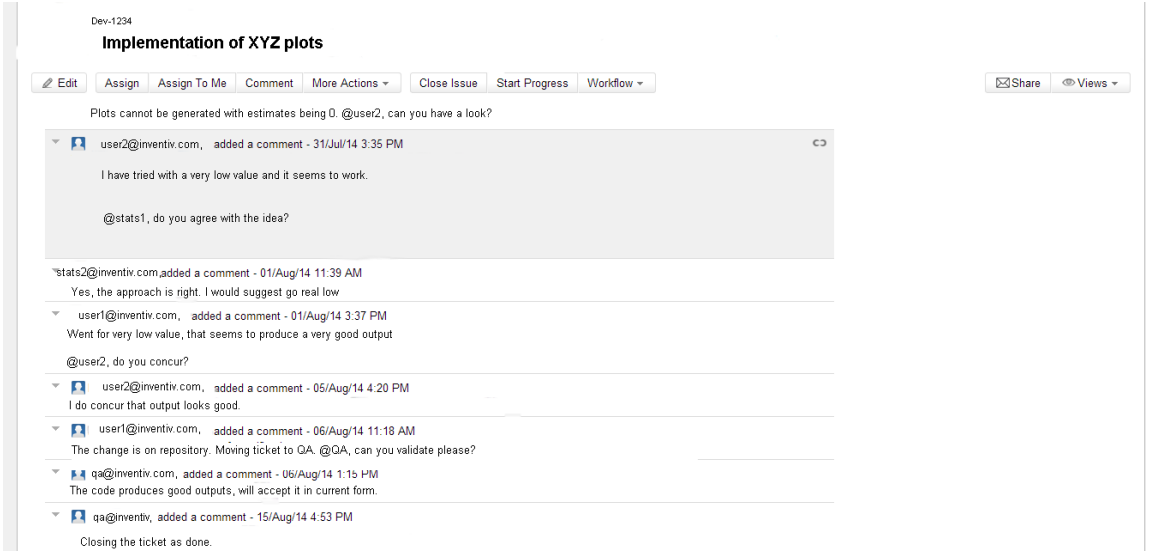

Figure 2: Comments in JIRA used for discussions

JIRA also solves the problem of visibility for all and the visibility is automatic.

1. Executives will be able to see a graph showing the issues raised against issues resolved for the whole company and if they spot a divergence, they will be able to see which project is letting the company down and can focus on those to bring them back under control. Please see *figure 3* and *figure 5* for project level views.
2. A director can view reports for his division. A project manager can view reports for his project.
3. A developer sees reports for his team and his testers. *Figure 6* and *figure 7* demonstrate team level view of progress.

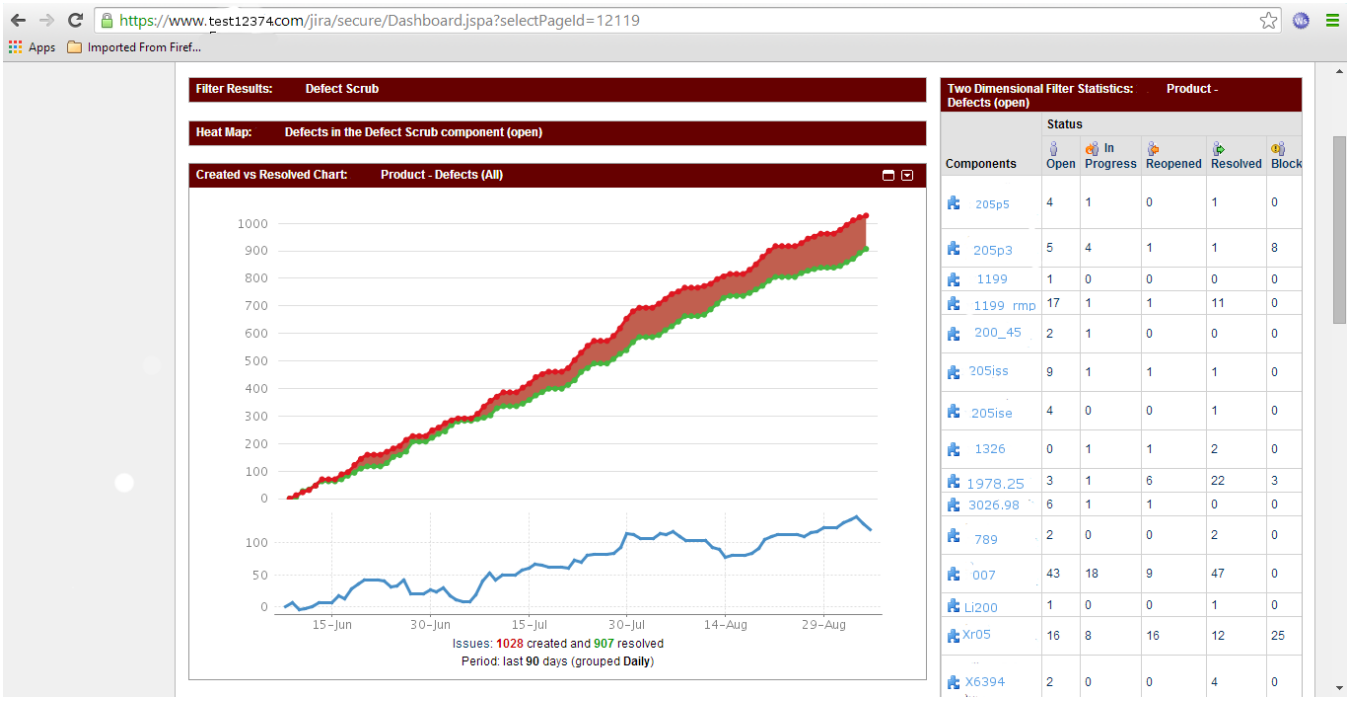

Figure 3: Project level view of all teams and cumulative status

Figure 4 demonstrates planning in JIRA. Issues are added to the backlog as and when the issue becomes known and then based on the priority, dragged into the current, next or any one of the sprints.

Figure 4: Project planning in JIRA.

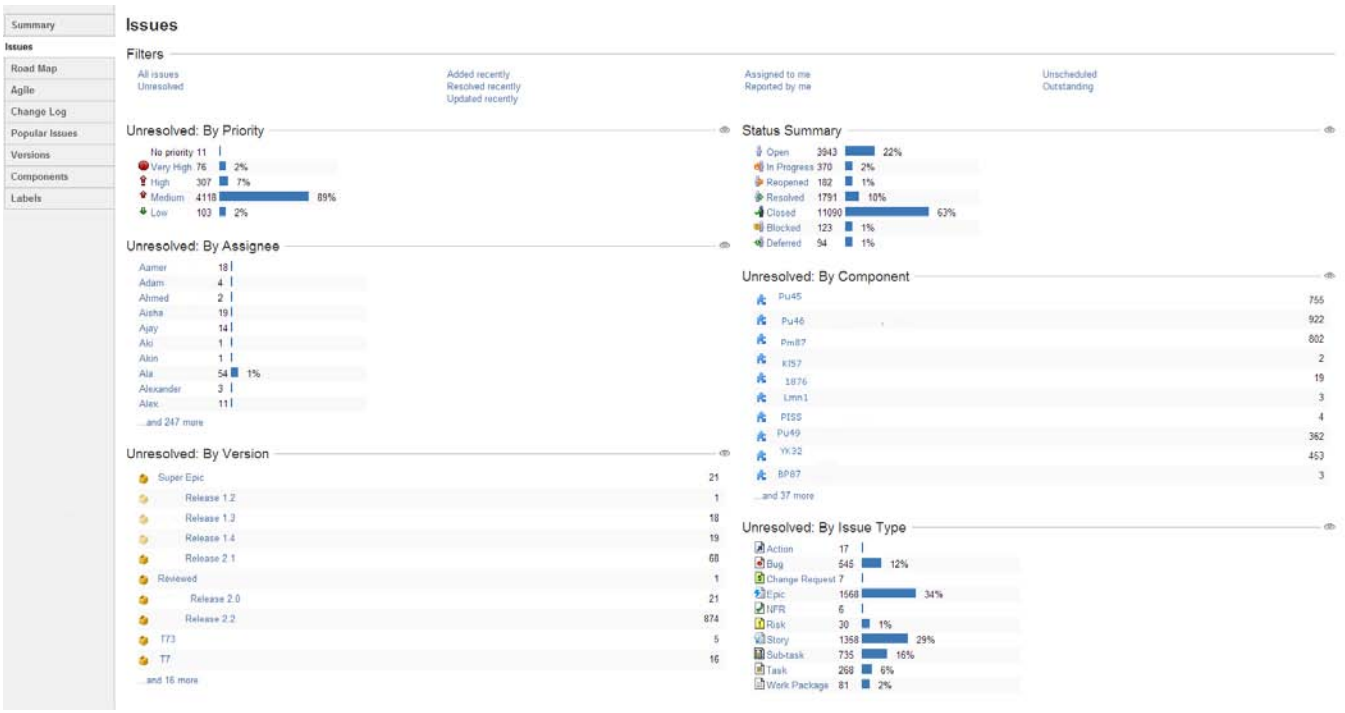

Figure 5: Project level view by task type.

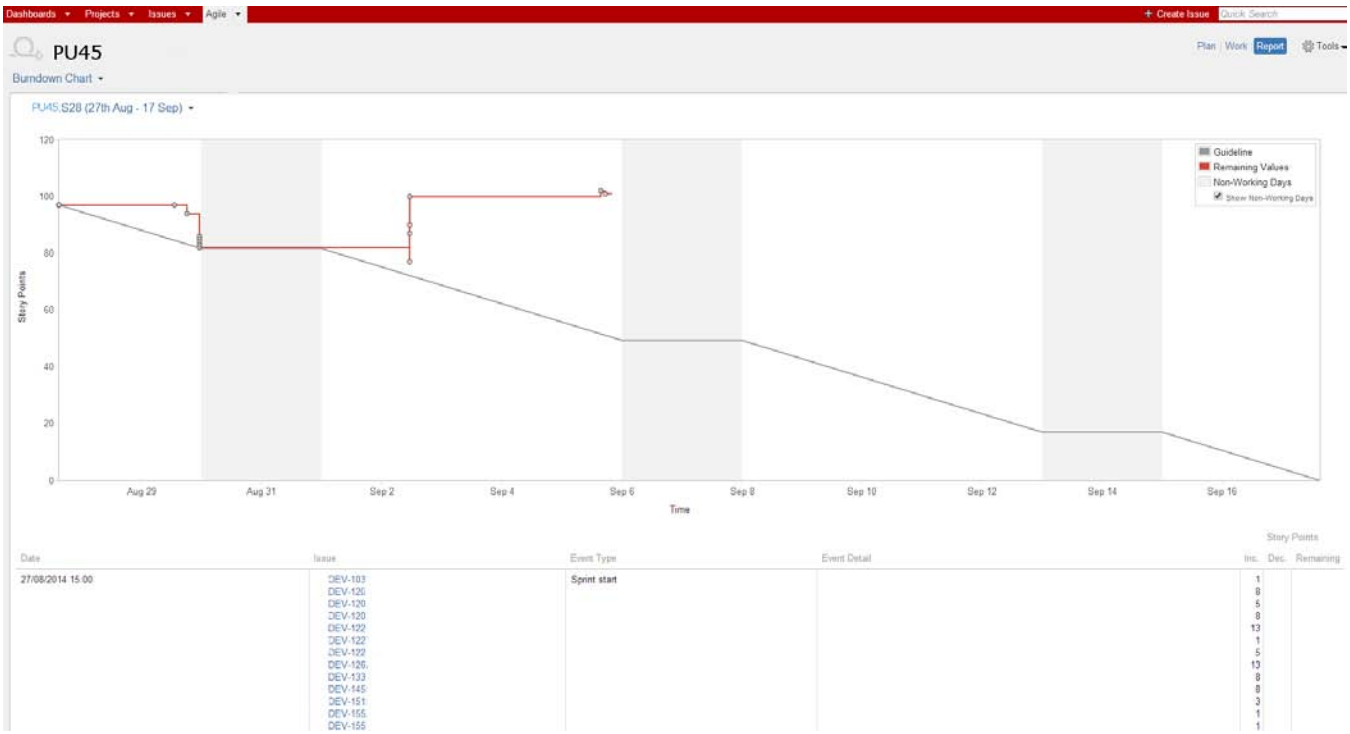

Figure 6: Team level view of progress

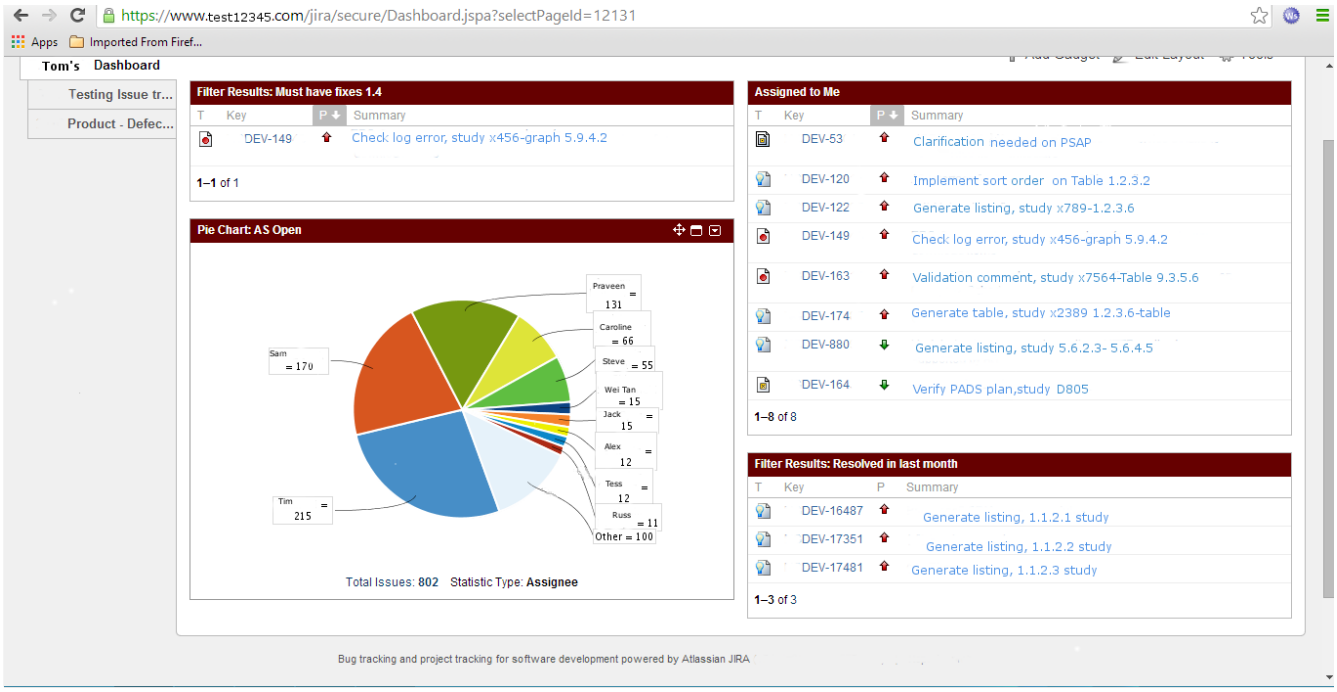

Figure 7: Team level view of work distribution.

Time spent by managers in preparing status reports and update meetings for clients and high-level management can thus be reduced and used more effectively elsewhere. Tools like JIRA bring efficiency across all levels, information flows freely with the right levels of control applied. This combined with tools like version management systems can contribute to higher productivity and efficiency in the company.

SOURCE CONTROL SYSTEMS - DISTRIBUTED (GIT) OR CENTRALIZED (SVN)

A version control system manages changes to code and document that allows a programmer to work without fear of losing his work at any stage by accidental deletion or modification that breaks a working code.

"Revision control, also known as version control and source control (and an aspect of software configuration management), is the management of changes to documents, computer programs, large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged."[4]

A version control system has its use in the SAS clinical trials programming industry to not only version control programs but also documents like specifications and templates.

Version control takes risk out of systems and adds improvement by:

1. Providing protection against accidental or deliberate deletion of important files or code.
2. Managing code/report releases become simpler.
3. Helps developer store his transient code that partially works at every stage of code development and removes fear in the developer of breaking something that was already working in order to fix something else.
4. Full history of changes to any file is preserved.

There are two types of version control systems:

1. Centralized version control systems: There is a central repository and all the clients synchronize with it, for example, CVS, SVN etc.
2. Distributed version control systems: Follows a peer-to-peer model, every copy is a repository and pulls and pushes can happen between peers, for example, GIT, Mercurial etc.

Every small change to SAS code or a document can be committed. Each commit has a unique commit ID. A label or a tag is used to mark a commit that in its current form has some special significance. For example, releases are tagged "Release-1.0" and "Release-2.0" and so on.

Project management tools like JIRA work well with version control systems like SVN. A commit ID mentioned on a ticket is enough for JIRA to pick up the patch from SVN/GIT directly so the users looking at the ticket can also directly see the changes made to a document or SAS code as a part of fixing the ticket. SVN also works well with build systems/automated testing frameworks like Jenkins.

JOB AUTOMATION WITH JENKINS AND AUTOMATIC TEST REPORTS

Jenkins is a tool that can run code automatically and produce summary reports. This tool can be aligned with a source control system like SVN to ensure a run on a separate session after every change is committed and the result is emailed after the run is completed. A SAS programmer need not manually start a run that lasts for hours and blocks the programmer doing any further productive work. Although, tools like Jenkins are commonly used as continuous integration tools, they can also be used in numerous other ways.

Jenkins runs a script at a particular time, interval or upon an event like a commit to SVN. What the script does is completely up to the writer of the script.

Jenkins could be integrated into the workflow of a SAS programmer, for example:

1. A Jenkins job is created to perform a 'Good Programming Practice' check on the programs of a study or a trial. This job is triggered every time a developer commits his code to the SVN repository and is scheduled to run at 20:00 Hrs every night. The developer commits his code and goes home in the evening. The next morning when the developer arrives at work, He/she finds the GPP check status reports last night's run in the inbox along with logs attached. If status is green, there were no GPP violations, if not, they look at the logs and fix the GPP violations.
2. A Jenkins job is designed to run when both the primary program and the validation program are committed. The job runs and compares the outputs of both the programs and any mismatches are reported to both the developers immediately via an auto generated e-mail.
3. A Jenkins job is written to count number of outputs a program is expected to produce. This runs on every commit and the developer is immediately notified if the count does not match expectations.
4. A Jenkins job performs log checks on every commit by a developer and if it finds errors in the log, the developer is notified via email with errors from the logs highlighted. *Figure 8* demonstrates the status of jobs.
5. A Jenkins job is created to verify that outputs of all the programs of a trial match the expected templates. This job runs every night and reports are sent out to the leads and the managers once the run is complete. As the project moves on and more and more programs are implemented correctly, there is a general movement of the result trend towards green as shown in the *figure 9.*

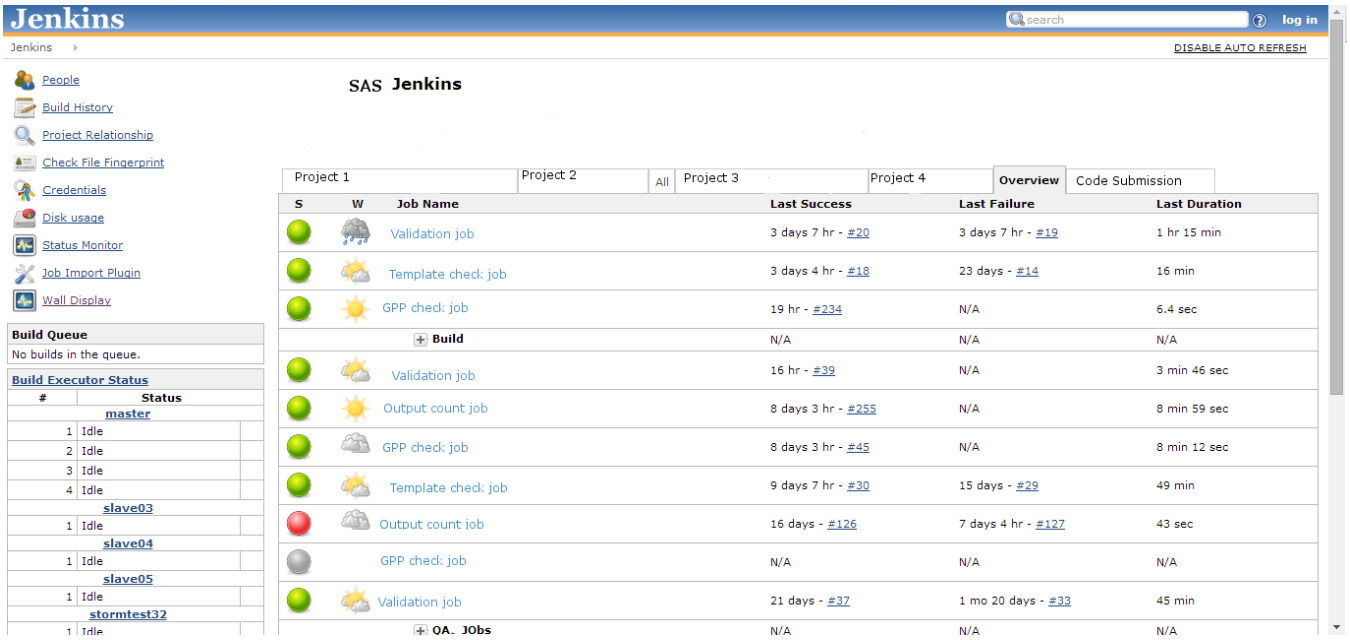

Figure 8: Status of last few runs of Jenkins jobs

The results are always stored in the data base. It is easy to compare results to previous runs to keep a tab on updates and progress. *Figure 8* demonstrates results of last few runs being maintained.

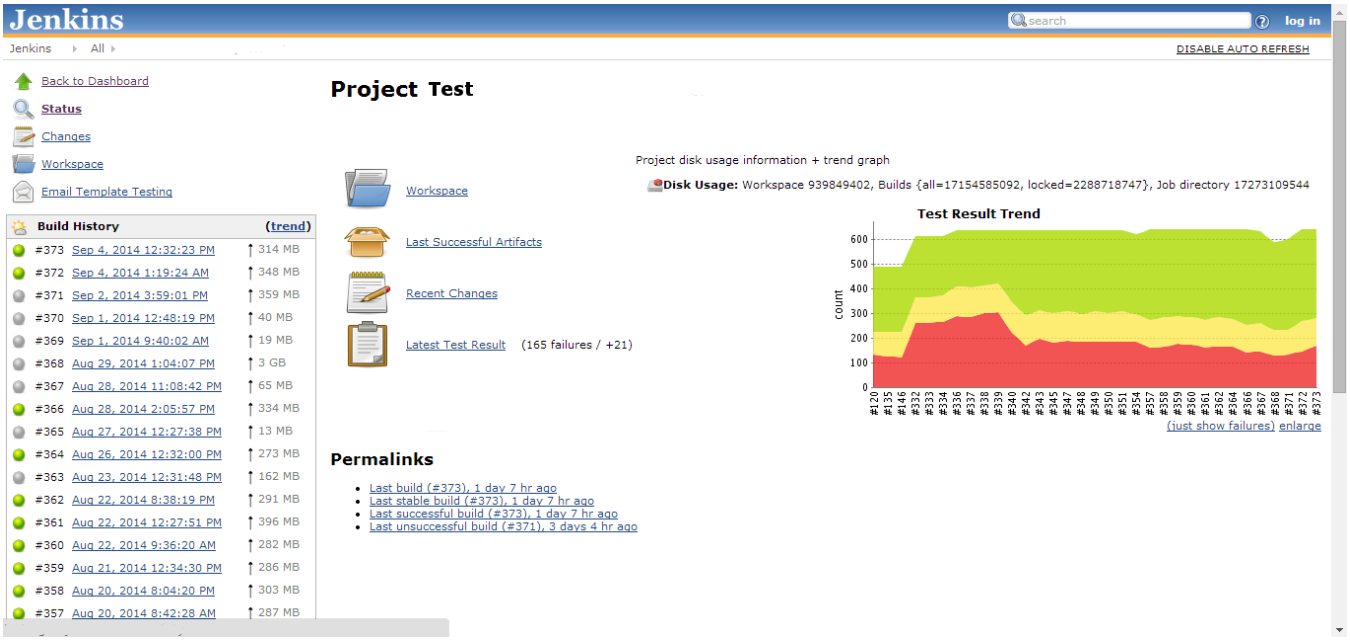

Figure 9: A Jenkins report for last few runs of a job

Jenkins works well with most version control systems and can form a good ecosystem for efficient development.

DOCUMENTING CODE AND DOXYGEN

Doxygen is a tool for used for writing software reference documents. The Doxygen documentation is written within code, and is thus relatively easy to keep up to date. Doxygen can cross reference documentation and code, so that the reader of a document can easily refer to the actual code.

"Briefly, Doxygen can generate an on-line documentation browser (in HTML) and/or an off-line reference manual from a set of documented source files. Documentation is extracted directly from the sources, making it is easier to keep documentation consistent with the source code. It also supports generating output in RTF (MS-Word), Postscript, hyperlinked PDF, compressed HTML, and Unix man pages. It can also be configured to extract the code structure from undocumented source files, enabling you to quickly find your way in large source distributions. The ability to automatically generate and include dependency graphs, inheritance diagrams, and collaboration diagrams enable relations between various elements to be visualized. It can also be used to create normal documentation."[5]

Doxygen is used by programmers of many languages other than SAS including C, C++, C#, Java, and Python. Therefore documenting to Doxygen style is a transferable skill across many programming languages.

Also, since Doxygen comments are written along with the code, any changes to code can be reflected in documentation.

How is Doxygen used?

- SAS programmers write their code using Doxygen standards.
- Programs reside in a directory called "Project" and its sub-directories on the "C" drive.
- For personal use, the developer opens a command prompt and changes directory "cd c:/Project" and runs two simple commands "doxygen" followed by "make". A PDF file is generated for quick read.
- The team lead wants to host an internal website that references the code. He again changes the directory to the project and runs the command "doxygen". A full html website is generated with comments from all the SAS programs extracted into the website. The home page is called "index.html". Through that page, the documentation for the whole project can be accessed. *Figure 9* demonstrates a website generated using Doxygen.

Benefits:

- Documentation writing happens alongside SAS code development with small changes to commenting style and no additional cost. The document then changes with the SAS comments and hence is always up to date.
- A well documented project results in less effort in knowledge sharing and improved efficiency.

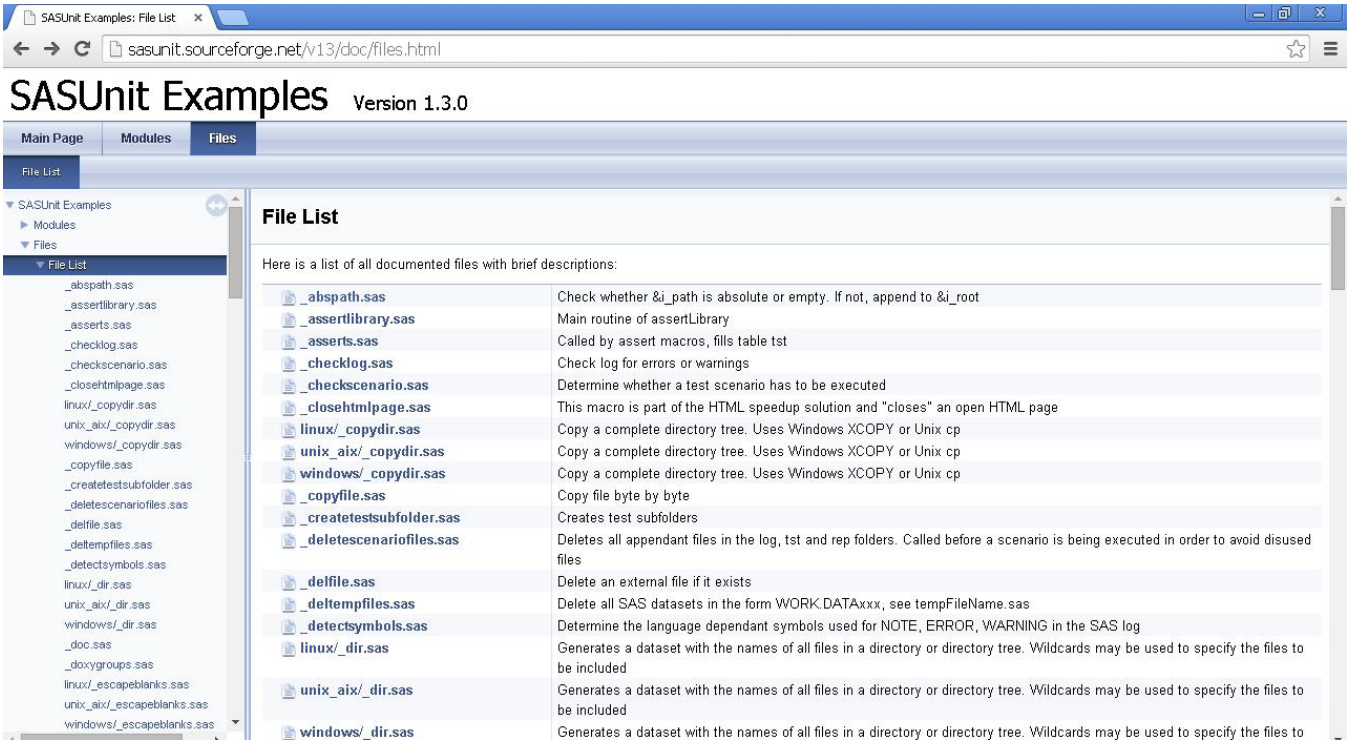

Figure 9: HTML output of Doxygen styled code.

SOFTWARE ENGINEERING:

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) IN SAS CLINICAL TRIALS - IMPLEMENTATION OF AGILE VS WATERFALL MODELS AND THEIR BENEFITS

"The waterfall model is a sequential design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance."[6]

The waterfall model of software development works in a linear fashion and progress to the next stage only occurs after the current stage is completed. The process is limited as the model cannot easily adjust to changes of requirements at an advanced stage of the project.

The stages of the water fall model are illustrated in figure 10.

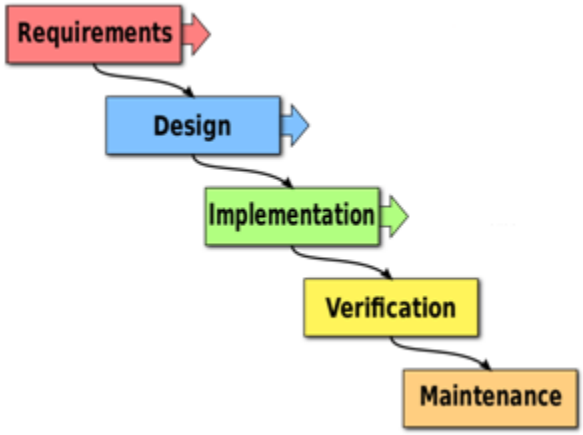

Figure 10: Stages in waterfall model

The clinical trials industry largely uses a modified version of the waterfall model. Although the approach has served the industry well, it is useful to research possible alternative approaches that have been successful in the other software development contexts such as, the agile methodology.

"Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement and encourages rapid and flexible response to change. It is a conceptual framework that focuses on frequently delivering small increments of working software."[7]

Agile method essentially has a philosophy of "do a little, deliver a little".

SCRUM is popular agile methodology widely used in software development and has 4 stages, as shown in *figure 11*:

- **Project analysis planning**: The initial Project Statistical Analysis Plan (PSAP) is planned at the start of the project. This stage could be revisited during any stage of the project to fine tune or update requirements. The PSAP keeps evolving with code and reports.
- **Sprint planning:** A sprint is a single iteration of the development of the code when a small incremental development has been achieved. A sprint can last from a week to 4 weeks. At the beginning of a sprint all the task to be done for the sprint are planned.
- **Daily scrum:** Is the daily meeting of the team to monitor the progress. If there are any blocking issues or any roadblocks to meeting sprint goals, they are addressed and resolved here.
- **Incremental release:** The end of the sprint release.

Figure 11: Stages in Agile SCRUM model [8]

Agile offers cost benefits with improved efficiency:

- Output code and reports are created with the Project Analysis Datasets (PADS). There is parallelization of work. Report development is not prevented by the development of the PADS.
- If the PADS are not moving forward, the developers can plan some other project into their sprints.
- Parallelization is achieved across projects, thereby increasing the per-developer throughput.
- Agile methodologies are known to achieve results in a short time frame at a high quality, thereby saving time and money for the company.
- Agile methodologies like SCRUM offer a lot of flexibility and firm focus on deliverables and a clean model for tech-debt management.
- The complete output report code is produced at the same time as the final version of the PADS

UML (UNIFIED MODEL LANGUAGE) AND BENEFITS

"The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system."[9]

UML provides a graphical and visual representation model of a problem at hand. This makes it easy to understand the problem and provides a clean model for discussions. If modeled well, the problem of converting a model to an implementation can be trivial.

There are two main categories of UML diagrams:

1. Structure diagrams: Represent the various components of a product and the relationship between the components.
2. Behavior diagrams: These diagrams model the behavior and the interaction of the system among its components.

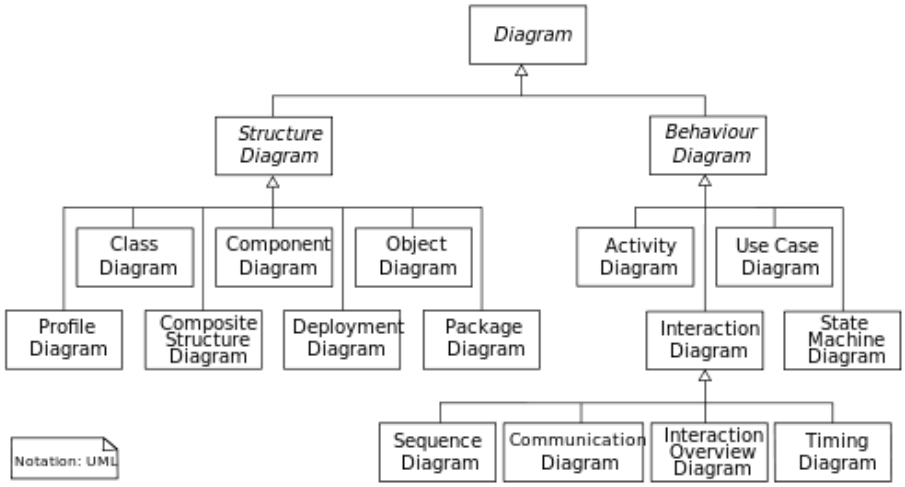

Figure 12: Hierarchy of UML diagrams [9]

Benefits of UML [10]

- "A picture speaks a thousand words". You know exactly what you are getting or what's expected of you.
- Conveying how the code should behave pictorially results in quick development and lower development costs
- A clear communication of logic results in code that will behave as you expect it to, providing fewer surprises
- Clean design results in memory and processor efficient systems. UML modeling facilitates clean design.
- System maintenance costs will be lower because of reuse of UML designs. Less relearning takes place
- Working with a new developer will be easier; a common language for programming communications exists.
- Communication with programmers and outside contractors will be more efficient
- There are many free and simple to use open source tools for generating UML diagrams available. The time required in modeling design in UML is less than the time saved during SAS program implementation due to clearer logic.

UML offers a good technique for SAS programs to be modeled upon.

DESIGN PATTERNS IN SAS PROGRAMMING

"In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design" [11]

A design pattern serves two primary purposes:

1. Reuse: A problem once solved in an efficient way in a certain context can be solved again in the same way in a similar context. There is little merit in reinventing the wheel. By reusing a known pattern, more confidence can be achieved as a result.
2. Established terminology: Design patterns offer an opportunity to name problems and their solutions. This makes it easy for developers to define and discuss a problem. Overall improvement in collaboration is aided by the use of design patterns.

Patterns of design are often used in SAS macro programming but are not as formalized in SAS like other programming languages. There is an attempt by Tabladillo[12] to formalize some patterns in SAS, of which we will look at a 'façade' pattern, see *figure 13*. "A facade is an object that provides a simplified interface to a larger body of code, such as a class library." [13]

```
%macro facade(dataset);
proc freq data=&dataset.;
      tables year*sales/list missing;
      run;
proc means data=&dataset.;
      var annualSales totalEmployees sumTurnover;
      run;
%mend facade;
%facade(dataset=sales.ca200503);
```

Figure 13: Façade pattern implementation in SAS [12]

"The façade above is the dataset name only, and the macro itself does the complex work of completing several procedures based on this simple one-variable interface." [12]

There are various questions that are answered by design patterns that could be adapted as a standard design solution.

- What should a programmer do in a situation where there should be only one way to access a resource like a database? Implement a “singleton pattern”. Have a macro that does all the interaction with the database and this macro is the only way to access the database for others.
- What should be done when a similar problem is encountered in a multi-threaded program? Implement a “master-worker pattern”. A single thread, called worker has access to the resource. All other threads talk to the database via this thread.
- What should be done when I have to move ahead with the programming when datasets are not yet available? Implement a “decorator pattern”. The pattern is also known as adaptor or wrapper. Implement a macro that mimics the behavior of a real library and unblock rest of your work. When the real library becomes available, replace the dummy code with real code without affecting any of the other code using this macro.

Benefits:

- Patternising solutions results in implementations that take little effort and there is an assurance that the solution which has been used many times before works reliably. This results in time saved on each reuse.
- Improved the efficiency and clarity of technical communications. Telling someone to implement a “singleton” tells them everything that needs to be done to solve a problem.

CONCLUSION

The paper highlighted some tools, software packages and engineering methodologies that are popular in non-SAS based software development industries. Consideration of which may improve overall efficiency, time-to-market and cost per delivery.

REFERENCES

1. Project management: http://en.wikipedia.org/wiki/Project_Management
2. Technical debt: http://www.notecolon.info/2012/10/technical-debt.html
3. Version control systems: http://en.wikipedia.org/wiki/Issue_tracking_system
4. Revision control: http://en.wikipedia.org/wiki/Revision_control
5. Doxygen: http://polywww.in2p3.fr/activites/physique/glast/workbook/pages/advanced_doxygen/usingDoxygen.htm
6. Waterfall model: http://en.wikipedia.org/wiki/Waterfall_model
7. Agile model: http://en.wikipedia.org/wiki/Agile_software_development
8. SCRUM: http://msdn.microsoft.com/en-us/library/vstudio/dd380647(v=vs.110).aspx
9. Unified Modeling Language User Guide, The (2 ed.). Addison-Wesley. 2005. p. 496. ISBN 0321267974. , See the sample content, look for history
10. UML: http://mimoza.marmara.edu.tr/~berna.altinel/courses/cse344Course/PS/UML.pdf
11. Design patterns: http://en.wikipedia.org/wiki/Software_design_pattern
12. Design patterns in SAS: http://www2.sas.com/proceedings/sugi31/173-31.pdf
13. Façade pattern: http://en.wikipedia.org/wiki/Facade_pattern
14. Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 260. ISBN 0-470-04212-5.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sameera Nandigama, Thames House, 17-19 Marlow Road, Maidenhead, Berkshire, SL6 7AA, United Kingdom
Tel: 00441628587388, Email: sameera.nandigama @inventivhealth.com, Web: http://www.inVentivHealthclinical.com

DISCLAIMER