Paper 140-30

Using Extreme Programming Processes in a SAS® Environment

Brian Mitchell, Ethicon Endo-Surgery, Cincinnati, OH

ABSTRACT

The Food and Drug Administration has mandated that the software development life cycle be used to ensure the quality of software used to report results of clinical trials. This paper examines how one of the newer approaches used in commercial software development, Extreme Programming (XP), can be applied in a SAS® programming environment to improve code quality and save development time.

XP delivers quality software by stressing the accurate determination of user requirements and the robustness of the software product. Among the key practices that it brings together are:

- High-level planning for next release iteration
- Frequent updates of functional software
- Simple design
- Testing integrated into coding
- Refactoring – improving code without changing system output
- Pair programming – 2 people code on a single PC
- Customer must be on-site and available for feedback
- Coding standards

In an environment where customer needs seem to be in a constant flux, XP provides a better way to produce quality software than the traditional waterfall approach of gathering requirements and then designing, coding, testing, and maintaining programs.

WHAT IS EXTREME PROGRAMMING?

Extreme Programming (XP) is one of many new lightweight software development life cycle methodologies (http://www.itstudyguide.com/papers/rwDISS725researchpaper1.htm). These methodologies have evolved from previous failures to produce complex software of acceptable quality in a reasonable amount of time. XP can provide many of the same benefits in a SAS® programming environment that it does for small teams using object-oriented languages for commercial software development. It includes many of the same features as other lightweight processes but its emphasis on testing is unique.

XP is defined by values, activities, and practices:

VALUES

Simplicity

✓ Provide what the customer needs right **now**
✓ Focus on the minimum requirements
✓ Simple programs are easier for teams to work with

Communication

✓ Regular face-to-face communication between programmers and customers
✓ Side-by-side communication between programmers - Pair programming
✓ The greatest obstacle to success is a lack of common understanding

Feedback

✓ The continual design process requires rapid feedback on each release
✓ Customer feedback keeps the team focused on priorities
✓ Continuous testing against customer needs guarantees quality

Courage

✓ Boldness to change direction quickly based on feedback
✓ Simplicity and testing support programmer confidence in change

PROGRAMMER ACTIVITIES

On a day-to-day basis, an XP programmer spends his or her time:

Listening
- ✓ Using active listening to minimize miscommunication

Testing
- ✓ Preparing automated tests before developing requirements code
- ✓ Testing for conformance and performance, not just defects

Coding
- ✓ Using code rather than comments, to express his or her intent
- ✓ Refactoring code, programming in pairs, and reviewing code

Designing
- ✓ Continually reviewing the design as customer needs change and as bug patches threaten to overwhelm the current design

XP PRACTICES

These dozen practices summarize what is meant by Extreme Programming:

1. Planning Game – develop a short-term plan to achieve the next release
2. Small Releases – Release to the customer every time new functionality is added
3. Metaphor – use a metaphor to communicate the intent of the project
4. Simple Design – use the simplest approach that is expected to work
5. Testing – build automated tests before writing project code
6. Refactoring – improve code simplicity and efficiency while maintaining functionality
7. Pair Programming – development environment shared by two programmers
8. Collective Ownership -any team member can change code at any time
9. Continuous Integration – components of the project are jointly tested at least daily
10. 40 hour work week – regular hours support quality work
11. On-site Customer – the customer is available onsite at all times
12. Coding Standards – Everyone follows common conventions

THE SOFTWARE DEVELOPMENT LIFE CYCLE UNDER XP

XP assumes a very close collaboration between the programmers and the customers. The customer defines the business value of the software throughout its development and provides requirements in the form of user stories. A user story is a non-technical description of how the user will use the system/program to satisfy a business need.

The programmer responds with an estimate of the time to produce a functional prototype to meet the minimum requirements of that user story. At first, the estimate will be very rough but with successive iterations, it becomes dependable. Producing these estimates will often require developing prototype code (called spiking) to ensure that a given programming approach is feasible.

In the succeeding iterations, the customer provides feedback on how well the software has met their needs and what additional user stories are needed for the next iteration. The customer controls deployment and makes the trade offs between development time and required functionality. The developer decides which user stories are incorporated into each successive software release and keeps the customer up-to-date on the technical risks of implementing each story and the progress of the development team.

In our environment, deployment is usually by a cutover to the new program without a transition period as no previous system exists. For complex systems, you may have to run both old and new programs concurrently for some time (*in situ* validation) or phase in the new program by discrete functions. A good approach to doing this, for example, is to replace the user interface while still using the existing system to execute behind the new interface. After deployment, XP can still provide value by using refactoring to improve efficiency and built-in testing to ensure code quality under maintenance.

XP PROGRAMMING PRACTICES IN ACTION

Several XP practices can impact SAS® programming. Let's look at these.

Simple Design

The design phase under XP focuses on clarity and simplicity:

- ✓ The module (SAS® macro or program) contains no duplicate code
- ✓ The intent of each piece of code is clear
- ✓ The module uses the fewest possible pieces of code to achieve that intent

Simple design means never designing code to meet potential requirements, only what is needed for the current iteration.

Testing

XP flips the traditional approach to testing on its head. The developer writes the code to carry out testing against the requirements (user stories) *before* beginning development of the project code. Since the tests will assure compliance with requirements, the developer writes the simplest code that will pass all of the tests. It pays to spend some time in developing concise and conclusive test code as it will be run many times a day to verify that the code fulfills the business need of the customer. One of the benefits of preparing tests first is that the developers will quickly learn if they do not have a clear understanding of the customer needs since they will be unsure what tests are appropriate. In this case, they need to circle back to the customer immediately and obtain that clarity.

Refactoring

Refactoring is the process of improving code without changing functionality. In the SAS® environment, you can think of it as replacing a macro with one that has the same parameters and accomplishes the same tasks but internally may use a completely different approach to getting there.

Pair Programming

This is something radically different about XP. It really requires a team of programmers to implement and so we have not tried this in my workplace. In essence, it means two programmers working together at a single PC, taking turns using the keyboard.

Far from halving productivity, this means that all design decisions are supported by two minds, that all code is constantly reviewed, and that while one programmer focuses on the details of coding, the other focuses on context and alignment with customer need. Since the cost of coding errors rises exponentially with the delay until they are identified, pair programming radically reduces costs since most errors can be caught at the very moment that they would have been created

Much like police partners, developing suitable pairs for programming can take a lot of effort. Fortunately, it can be used for coding sessions just an hour or two at a time, to minimize the stress of constantly working towards consensus.

Collective Ownership

When a programming team is used for an XP project, it's important that all team members share ownership of the code. In the traditional framework, if you're calling someone else's macro but it doesn't quite have the features you need, you'd go to them and ask them to modify it. This can seriously hold up a project if the other programmer is either unavailable or unwilling to give your request any priority. With XP everyone owns the code, so you just go ahead and make the needed changes yourself, and notify the team of what you have done. You should find this fairly easy to do since the code is simple and everyone follows the same conventions.

Coding Standards

In keeping with the simplicity principle and the close collaboration needed for pair programming, coding standards are a necessity for XP. At EES, we have divided standards into mandatory ones and guidelines or best practices. This division allows some room for programmer individuality while ensuring sufficient standardization that anyone on the team can easily understand, and, when necessary, modify, the code at any point in development. The mandatory conventions can also help resolve petty disputes around code formatting among the team. Your standards should encompass indentation, capitalization, commenting, naming conventions, and error handling. You

can find suggestions for SAS® coding conventions in Rhodes (2004) and Levin (2003). For non-language-specific coding conventions, see McConnell (1993).

Continuous Integration

The XP team needs to test the current code base at least daily. Integration errors can be the most difficult to solve and regular testing catches them as soon as possible. This testing validates that the entire code base is functional and ready for potential deployment at any time.

Forty-Hour Work Week

Before Agile processes, projects often had almost no functionality until they were complete, so teams either added more people or worked longer hours when the project fell behind schedule. Research has shown that this strategy is rarely successful as more mistakes are made during long hours which can actually delay the project rather than speed it up. XP assumes a normal work schedule where morale remains strong and well-rested team members produce quality code.

AN XP WALKTHROUGH

For a simple example, suppose that your customer, a statistician, needs summary statistics from one or more variables in a SAS® dataset. How does the XP programmer proceed?

First, they spend some time with the customer to *really* understand their expressed need and its context. How will the summary statistics be used? What is the business need for this data? Is there an underlying problem that some other information would better satisfy?

The customer presents some user stories. In this case, there is a single user story, "I want summary statistics for some variables in a SAS® dataset". The user story allows the customer to get the most possible value out of your programming efforts and the programmer to clearly understand what is needed.

Once you're convinced that the customer need is, in fact, best addressed by SAS® code that generates some summary statistics, it's time to translate that need into functional tests for the macro that you intend to create. This macro obviously requires a SAS® dataset containing one or more variables to summarize. You will need to test for the existence of the dataset and the user-selected variables. Are the variables of the right type (say, numeric) to be summarized in the desired manner? What if all or almost all of the values are missing? You must develop tests for all foreseeable outcomes that would fail to provide the output expected by the customer. Finally, what specific output is needed and where does the output need to go? (summaries, error messages and the results of the testing). In this case, since the statistician is very familiar with SAS®, it makes sense to route any error messages to the log window using the ERROR statement. You might also want to route messages to a specific file for testing purposes.

Try to use XP components when doing testing in order to minimize complexity and the likelihood of creating errors in the test code. For example, I used a set of macros for DDE access to WORD documents that essentially write WordBasic commands to WORD's input file. Rather than testing these by creating a WORD document and using DDE, I just redirected the output to a text file and compared this file to one containing the expected commands.

Once the necessary tests have been coded, the programmer looks for the simplest piece of code that could possibly satisfy those tests. Use a PROC, if possible, rather than writing any code at all. There's an additional benefit there in that some PROCs incorporate testing, further simplifying the programmer's job. If you do have to write code, ignore efficiency and use the simplest statements possible. If you feel that you must add a comment to explain what the code is doing, it's probably not simple enough.

You may be tempted to add a bit of code to allow the input of options for the PROC since certainly the customer will need to make choices about those. Stop right there! That's a bad idea. There's no test for that feature since it wasn't a user story and it shouldn't be coded in this iteration. Save that thought to discuss with the customer at the next review session.
Once the code passes all of the tests, it's time to release it to the customer. After they've had a chance to work with it, you meet again to discuss whether or not it is meeting their needs and what new functionality they'd like to see in the next release.

Once the customer agrees that the software has sufficient functionality (and/or they have reached a must-deploy date), the software goes into production. The XP process has ensured that the software is reliable and has at least some functionality to address the business need. Now is the time for the programmer to circle back and look at refactoring to improve the efficiency of the software (if warranted).

In our environment at EES, datasets are typically small so that program efficiency is rarely a concern. Our most limited resource is programmer time and XP promises to minimize unnecessary work and rework

REFERENCES

Baird, Stewart. 2003. *Sams Teach Yourself Extreme Programming in 24 Hours.* Sams Publishing. Indianapolis, IN.

Fowler, Martin. 1999. *Refactoring – Improving the Design of Existing Code.* Addison Wesley.

Jeffries, R., Anderson, A. and C. Hendrickson. 2001. *Extreme Programming Installed.* Addison-Wesley.

Levin, Lois. 2003. *SAS® Programming Conventions.* In the 28th Annual Proceedings of the SAS® Users Group International.

McConnell, Steve. 1993. *Code Complete: A Practical Handbook of Software Construction.* Microsoft Press. Redmond, WA.

Rhodes, Dianne Louise. 2004. *Programming Standards, Style Sheets, and Peer Reviews: A Practical Guide.* In the 29th Annual Proceedings of the SAS® Users Group International.

Wonak, Ronald G. 2001. *SDLC on a Diet.* < http://www.itstudyguide.com/papers/rwDISS725researchpaper1.htm>

CONTACT INFORMATION

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

Brian Mitchell
Ethicon Endo-Surgery, Inc.
4545 Creek Road, ML22
Cincinnati OH 45242
513-337-3337
bmitche3@eesus.jnj.com