Source Code Control Workflows for Open Source Software

Kevin Gary
Department of Engineering
Arizona State University
Mesa, AZ 85212*)*
kgary@asu.edu

Ziv Yaniv, Ozgur Guler, and Kevin Cleary
The Sheik Zayed Intitute for Pediatric Surgical Innovation
Children's National Medical Center
Washington, D.C. 20001
zyaniv,oguler,kcleary@childrensnational.org

Andinet Enquobahrie
Kitware, Inc.
Carrboro, NC, 27510
andinet.enqu@kitware.com

***Abstract*—Many open source projects rely on the dedicated and highly skilled members of distributed development teams. These teams often employ agile methods, as the focus is on concurrent development and fast production over requirements management and quality assurance. The image-guided surgical toolkit is an open source project that relies on the collaboration of a skilled distributed development team, yet addresses a safety-critical domain. Due to this rare intersection of agile and open source development processes and a safety-critical domain, the IGSTK team has had to enhance the process with key elements and a set of best practices to augment commonly applied agile methods. This paper presents our experiences and lays out some research questions for the future.**

***Index Terms*—agile, open source, safety-critical software.**

I. Introduction

As agile methods have matured, so has the realization that these methods are not dogmatic in their approach. Agile methods encourage *the right amount* of ceremony; therefore if a safety-critical system requires a greater emphasis on non-coding process activities like documented design and requirements management, then an agile approach will include these as necessary activities and not ceremony. Furthermore, we argue that agile and open source approaches focus more on code-level quality that most classic software engineering process models, which often talk about quality in every phase of the lifecycle *except* implementation.

We present our experiences on the image-guided surgical toolkit (IGSTK) project as a backdrop for this discussion. IGSTK is an open source software project that has employed agile best practices for the past nine years. In that time we started with the

assumption that a lighter process is better, focusing on evolving code, and only adding in process elements where the need has arisen. The IGSTK team just released version 5.2 to the community, and in the past year has adopted modifications to its software processes.

II. Related Perspectives

Boehm [2] articulated the widespread belief that agile methods, due in part to their lack of emphasis on documentation, requirements stabilization, planning, and other large-scale synchronization points in the software process. But recently, literature has started to appear suggesting this may not always be the case, particularly in healthcare applications. Dwight and Barnes [5] describe a "lean-to-adaptive (L2APP)" variant on agile methods in a clinical research setting to streamline value delivery by utilizing a parallel flow side-by-side with laboratory validation. The three-phase L2APP model (speculation, collaboration, learning) strikes us as an agile way to manage requirements in an innovation-driven domain, though it does not say much about downstream processes related to design review and in-construction change management. The authors acknowledge the process is leveraged in the clinical lab to develop innovations for later large-scale production by shops prepared to take a technology to market. To us this represents a reasonable tradeoff in upstream efficiency but may punt too much responsibility downstream – how early does evidence of traceability and design rigor have to be accumulated?

Ge, Paige, and McDermid [8] present a detailed discussion of Agile versus plan-driven methods using Boehm's central premise as the framework for discussion. The authors then go on to suggest a semi-agile process that incorporates aspects of traditional plan-driven processes such as up-front design and hazard analysis with iterative development *and* iterative development of a safety argument. The presentation admits there is a lot of devil-in-the-details, with no current general principles for deciding how much agility is too much or too little. The key seems to be in calibrating the process to do just enough at each point in the process, a complex process goal.

Despite the complexity, we philosophically agree with [8] in our own recent paper [7]. Agile methods do embrace activities like planning, design, and validation as long as they are *without ceremony*, meaning they are not performed for performance sake; they are performed at the right times and to the extent needed (and no more) to achieve product requirements (for example, certification). Exploration of general principles, reference process frameworks, or evaluation criteria to guide practitioners in adopting "just enough agile" is a worthy pursuit. Finally we note that none of this discussion contradicts Boehm's original assertions; Boehm noted that knowing the right places to apply the right process is critical, and we view these explorations as an investigation into where agile can fit as a means of harnessing its benefits in the healthcare domain.

III. IGSTK

IGSTK is an open source framework for creating surgical applications. IGSTK is distributed under a BSD-like license that allows for dual-use between academic research labs and commercial entities. Image-guided surgery involves the use of preoperative medical images to provide image overlay and instrument guidance during procedures. The toolkit contains the basic software components to construct an image-guided system, including a tracker and a four-quadrant view, incorporating image overlay. IGSTK also leverages other open source projects, specifically ITK for segmentation and registration, VTK for visualization, and FLTK and Qt for the user interface.

IGSTK has geographically distributed developers, complex application requirements, and framework constraints for extensible and reusable architecture components. This is obviously a tremendous challenge compounded by the safety critical nature of the domain. The IGSTK team has created its software processes to balance an agile development philosophy with an integrated requirements elicitation and management approach, and consequently has arrived at a methodology that is fast and flexible, yet meets the stringent needs of this application domain.

IGSTK development presents interesting challenges from a methodology perspective. These complexities derive from the nature of the requirements, the makeup of the team, the dependence on pre-existing software packages, and the need for high quality standards within this domain.

The first challenge to IGSTK development derives from the nature of the framework-level requirements, which are difficult to completely understand before applications are constructed upon it. Waterfall-style methodologies [11] that attempt to define requirements completely before development begins are not considered suitable. Rational Unified Process (RUP) use-case driven modeling [10] is selectively applied through a customized process C-PLAD [1], as we cannot assume non-functional requirements derived from a set of applications known today represent a complete set of requirements for the future.

The second challenge to IGSTK development is the makeup of the team, comprised of academic and commercial partners collaborating in a distributed setting. Most if not all of the team members have other demands on their time. These factors create challenges for setting project deliverables and expectations over medium- and long-term horizons. Fortunately, most developers are deeply familiar with the domain and with a common set of tools and libraries from which to begin development. The requirements, team composition, and use of pre-existing software suggest that agile methods [4] should be applied to IGSTK. All team members have significant exposure to agile methods; some have even developed agile-ready tools that are employed on IGSTK [12].

Another challenge to IGSTK development – the high quality standards demanded the application domain – suggests that some agile practices need to be reinforced. For example, FDA guidelines for approval of medical devices require traceability of requirements through implementation and testing. To address these complexities, IGSTK adopted an agile approach augmented by a set of best practices we have previously described in detail [6] so we merely list here:

Best Practice #1: Recognize people as the most important mechanism for ensuring high quality software. This agrees with the philosophy espoused by the agile community [3].

Best Practice #2: Promote constant communication.

Best Practice #3: Produce iterative releases.

Best Practice #4: Manage source code carefully. A paradox of open source development in this space is that on the one hand you want to encourage community contributions and innovation, but on the other you need to manage change to software artifacts carefully. We expand on our recent process modifications to address this issue (in part).

Best Practice #5: Validate the architecture. This best practice is a nod to the specialty of the domain, and is discussed in more detail in the next section.

Best Practice #6. Emphasize continuous builds and testing.

Best Practice #7. Support the process with robust tools.

Best Practice #8. Emphasize requirements management in lockstep with code management.

Best Practice #9. Focus on meeting exactly the current set of requirements; it makes traceability easier, not harder.

Best Practice #10. Allow the process to evolve.

IV. Best practices for Community Source Control

These best practices are not foreign to agile practitioners, or even to non-agile practitioners. In the safety critical domain, following only these practices is unusual and not sufficient. Key process elements need augmentation to ensure safety. In a previous paper [7] we explored architecture validation (best practice #5) and requirements management (best practice #8). In this section we describe our past and new activities to support best practice #4, *manage source code carefully*.

Agile methods are certainly highly iterative; the predominant agile process models, Scrum and XP, use short time-boxed iterations as a mechanism for managing change. But beyond short iterations, agile methods have other practices facilitating an almost continuous checkpointing of the process – the daily scrum, pair programming, scrumboards, and continuous integration and testing dashboards. However these practices are typically best implemented when the team is physically co-located and dedicated to the project. Hurdles, ideas, and other communications are addressed in real-time. Even with powerful online tools, geographically distributed teams can only rarely achieve this real-time interaction. Time boundaries, language barriers, network infrastructure issues, and local distractions and responsibilities at multiple sites are common causes for this degradation. It is exacerbated in open source communities, where participants are often dispersed individuals working for different organizations and only part-time in that community. Further, the community has to have either formal or informal rules regarding who can do what with which source code modules. IGSTK operates under such constraints, with global participation from researchers in hospital labs and universities, industry partners, and practitioners who made partial contributions over time. IGSTK has employed some traditional mechanisms for managing this collaboration, including developer meetings, user group meetings, online wikis and support forums, two mailing lists (adopters and core developers), and restrictions on core component development to only the core team.

Source code control is a critical practice in managing change in an agile and open source environment. At any point in time a community developer may be working on a defect fix, a new core feature, a new non-core feature, a refactoring, or an application-specific behavior or integration. That developer may be working in isolation, with little visibility in the rest of the community until the time arrives that s/he desires submission of the changed code. Should the code be accepted? Does it adhere to defined quality policies? Has it been code reviewed? It is an experimental or application-feature or a feature identified as desired by the community? These and more questions arise in this situation, and all pose risks when developing in a safety critical domain.

Over the past decade IGSTK has used a traditional, centralized approach to source code control common in many software projects. This approach supports a “branch-and-merge” centralized workflow. IGSTK further adopted a multiple codeline practice

known as *sandboxing* to allow for experimental features to be developed under lower quality conditions. But problems arose over time. The sandbox codeline grew larger, much larger, than the main product codeline, to the extent that less rigorous traceability on the sandbox led to inevitable technical debt. In other words, the sandbox repository became filled with incomplete features whose owners may have gone inactive and whose documentation and issue management in other tools was outdated. Even if a community member identified a desire to complete a sandboxed feature, they were often forced into significant rework or to scrap the sandbox module and start over in order to meet the quality policies on the mainline.

In the past year IGSTK has moved to the popular distributed version control system, Git. Git enjoys significant popularity now, though many projects use it in the same manner as traditional centralized *delta* repositories. Git's distributed repository model encourages many practices that go against low-level practices taught in the traditional model – for example, instead of "check-in early and check-in often" (to minimize merge conflicts in optimistic centralized tools), the distributed model encourages local branches with infrequent merges, preferring to merge only when a feature is complete and up to quality policy. The need for a sandbox is gone. Further, it encourages self-sustaining communities; community practitioners may maintain their own forked versions of repositories without burdening the core team with constant review of their features. Gone are the days of "contrib. modules" one may "use at their own risk" from the centralized repository; now one may publish their own forked repository and leave it to the market of adopters to decide what to use. A concern in this model is with the overall safety properties of the forked repositories – who has the overall ability to validate the safety properties of these forks with the core? A good research question! For the time being, this model saves the IGSTK core team valuable time in reviewing non-critical development.

Because of the peer distributed repository model used by Git and like-minded tools, a large variety of workflows may be employed on a project [3]. The IGSTK team reviewed the Git workflow literature and practices from related communities like ITK to adopt a variation of a *branchy workflow*. In this workflow two branch types are defined, a *topic branch* and an *integration branch*. The topic branch commits represent work on a single new feature or fix. The local developer(s) who work(s) on it name it locally but the branch is not public on the blessed IGSTK repository, so no other developers can branch from it. The integration branch is where merges of two or more topics happen. These branches have quality policy constraints enforced, and are publicly named on the blessed repository (one may pull from it).

Figure 1 depicts the relationship between topic and integration branches.

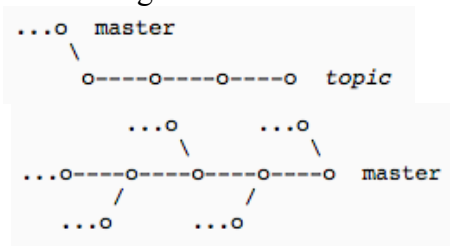

Figure 1. Topic and Integration branch commit patterns

Neither topic nor integration branches represent the main codeline, this is maintained separately in Git as the default branch (*master*). In other words, merges are not done directly into master, but into new integration branches, which are then merged into master after the integration is deemed stable and up to quality policies. Figure 2 shows what a sequence of commits may look

like in IGSTK between the master, a topic branch, and an integration branch named next. Further details on the IGSTK Git workflow may be found on the IGSTK Wiki at http://www.igstk.org/Wiki/Git/Workflow/Topic.

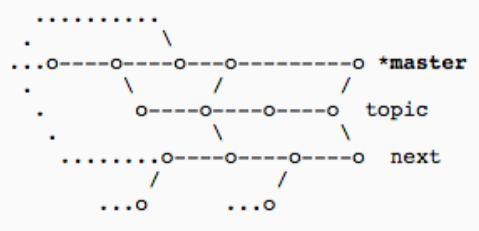

Figure 2. A possible sequence of commits on all branch types

This was a detailed presentation of a particular best practice in the agile IGSTK open source community. The level of rigor in daily collaborative practices of community developers is significant, and suggests if the trail of data of these practices can be harnessed and analyzed, it could provide a basis for safety case evidence for software in healthcare.

V. Discussion

IGSTK's agile approach is neither as rigorous nor as complete as it could be for a safety-critical domain. IGSTK is principally used in academic research centers and some small commercial endeavors outside the USA, which can afford to be more forgiving. Yet, the tale of IGSTK's agile evolution, we think, offers lessons and hope for applying agile methods to safety-critical domains. The work is laborious; to create and faithfully execute agile practices in a distributed open source community, every detail of the daily practices must be examined for the right amount of ceremony. We presented our revised approach to source code control as an example.

As we indicated at the end of section II, we believe there is an opportunity to create guidelines, models, and/or quality process criteria for the introduction of agile methods in the healthcare domain. Foremost, we believe it is a necessity – the explosion of personal medical devices and information management platforms such as the fitbit (fitbit.com) or smartphone-based sensor apps has serious implications for future patterns of individual-to-clinician healthcare. It will eventually become a necessity in systems development in healthcare (for example, tele-robotic surgery). Changes in medical device regulatory evaluation to a more evidentiary case-based approach [9] opens the door for agility. If agile methods can be instrumented to collect and aggregate daily practices into such evidence, then the possibility exists for expanded opportunities in healthcare development. Certainly the economic drivers are there. In our view research is needed on how to instrument agility to collect the evidence required for safety cases. Agile's benefits include the lightweight but constant management of detailed activity, and making this visible and transparent to all stakeholders. The increasing adoption of tools within the agile process focuses on communication between stakeholders (chickens) and developers (pigs). The identification and instrumentation of daily safety-related activities needs to be included in such toolsets to make this evidence collection continuous, feasible, and complete. Safety-based micro-evidence may then be aggregated to uncover macro-trends and introduce process improvements. This is a current focus of our research in this area.

Acknowledgment

This work was funded by NIBIB/NIH grant R01 EB007195. This paper does not necessarily reflect the position or policy of the U.S. Government.

References

[1] Blake, M.B., Cleary, K., Ibanez, L., Ranjan, S.R., and Gary, K., "Use Case-Driven Component Specification: A

Medical Applications Perspective to Product Line Development," ACM Symposium on Applied Computing, Santa Fe, NM (2005).

[2] Boehm, B. Get ready for agile methods, with care. *IEEE Computer* 2002; **35**(1):64–69.

[3] Chacon, S. *Pro Git*. APress, 2009.

[4] Cockburn, A.: "Characterizing People as Non-linear, First-order Components in Software Development." 4th International Multi-Conference on Systems, Cybernetics and Informatics, Orlando, Florida, (2000).

[5] Dwight, Z. and Barnes, A. Laboratory Driven, Lean-to-Adaptive Prototyping in Parallel for Web Software Project Identification and Application Development in Health Science and Research. *Software Engineering and Applications*, 2012; 5:62-68.

[6] Gary, K., Ibanez, L., Aylward, S. Gobbi, D., Blake, M.B., and Cleary, K. IGSTK: An Open Source Software Toolkit for Image-Guided Surgery. *IEEE Computer, vol. 39, no. 4, pp.46-53*, April 2006.

[7] Gary, K., Kokoori, S., Muffih, B., Enquobahrie, A., Cheng, P., Yaniv, Z., & Cleary, K. "Agile Methods for Safety-Critical Open Source Software", *Software: Practice and Experience*, 41:945-962, April 2011.

[8] Ge, X., Paige, F., and McDermid, J.A. An Itaretive Approach for the Development of Safety-Critical Software and Safety Arguments. *AGILE Conference* (2010).

[9] Geisler, J. "Software for Medical Devices", in *Mission-Critical and Safety-Critical Systems Handbook Design and Development for Embedded Applications* (Fowler, K. ed.) 2010 Elsevier Inc.

[10] Kruchten, P. *The Rational Unified Process—An Introduction, 2nd Edition*, Addison-Wesley (2000).

[11] Royce, W.W.: "Managing the development of large software systems: concepts and techniques." IEEE WestCon, Los Angeles, 1970.

[12] Schroeder, W.J., Ibanez, L. Martin, K.M.: "Software Process: The Key to Developing Robust, Reusable and Maintainable Open-Source Software." Proceedings of the IEEE International Symposium on Biomedical Imaging. Arlington, VA 2004.