Can All Sequential Processes Grow Up to Be Iterative and Incremental?

Hakan Erdogmus

My first exposure to Extreme Programming (XP) was through Kent Beck’s October 1999 article published in Computer (“Embracing Change with Extreme Programming,” pp. 70–77). I hadn’t paid much attention at the time. I had simply glossed over it and forgotten about XP for a whole year. But I remember staring at the figure on the article’s first page for a while.

The figure depicted a stereotypical waterfall process on the left (phases in the shape of rectangular boxes stacked on top of each other), an iterative process in the middle (three mini versions of the same stack of boxes staggered on a staircase), and XP on the right (several illegally tiny versions of the stack squished on a sloping line). When I looked at it, I remember thinking, “Aha, it’s simply a composition of microprocesses, each of which looks like a waterfall.” Understood. No need to read the rest of the article: take a sequential process, shrink it, replicate the shrunken version, recompose the replicates, and you get an iterative and incremental one. Very nice. Let’s call this idea the periodicity principle. We can apply the periodicity principle to just about anything, right? Sure, pictorially, but modern iterative and incremental processes are not, structurally or temporally, just finer versions of, say, the spiral model. By “structurally,” I mean from the perspective of how work flows through a process at different levels of granularity. By “temporally,” I mean how work is scheduled inside a workflow. So what’s missing here?

Beyond Periodicity

Kanban systems (a just-in-time production method that limits work in progress) and lean development supply the missing link: divisibility of work. Intuitively, the idea is simple. Divide the work up into independent pieces and apply periodicity by executing the same sequential process individually on those pieces: pick one piece, apply the process; then pick the second piece and repeat. Voila: you get an iterative and incremental process spread in time as a periodical sequential process. But we can do better. Corey Ladas explains how Kanban systems allow divisibility of work to be leveraged in software development in his collection of writings Scrumban: Essays on Kanban Systems for Lean System Development (Modus Cooperandi Press, 2008).

I came across Ladas’s work while working on a practice guide for software acceptance. I was help-
ing the guide’s authors with revisions, extensions, and background research. The guide advocated an iterative and incremental software acceptance process whose underlyng workflow (the path a single work item would follow from beginning to end) mimicked the Stage-Gate Process, a general product development model that is sequential in spirit. However, my colleagues’ approach accommodated both multiple overlaid streams and repetitions of the same workflow, similar to the more recent and flexible interpretations of the Stage-Gate Process by its inventor Robert C. Cooper (“The Stage-Gate Idea-to-Launch Process—Update, What’s New, and NexGen Systems,” J. Product Innovation Management, May 2008).

Another familiar software development model to which the guide refers, the V-model (Mark Fewster and Dorothy Graham, Software Test Automation, ACM Press, 1999), also has a sequential depiction. It can also be recast as iterative and incremental by applying two additional principles that Ladas derives from divisibility of work. Ladas’s principles are powerful because they allow such recasting at any granularity, be it releases, iterations within releases, feature sets within iterations, or individual features within feature sets. I’ll refer to these principles, together with periodicity, as unraveling principles, because they let us unravel a sequence of steps into smaller, intertwined, repeating threads (see the “Further Reading” sidebar for more information).

**Essential Workflow and Other Unraveling Principles**

Ladas defines essential workflow as the implied ordering of the steps, from beginning to end, applied to a work item flowing through a development process to implement an improvement, whether a new functionality, a cross-cutting concern, a quality requirement, or a bug fix. Figure 1a depicts a simple essential workflow with three steps.

Ladas’s first unraveling principle, pipelining, allows reorganizing a sequential essential workflow as parallel streams with re-
current merging points. The merging points are where the outputs from the streams are integrated. Merging introduces new steps into the essential workflow, demonstrated in Figure 1b. Ladas’s second unraveling principle allows separating the workflow from how the work that flows through it is scheduled. Ladas calls this separation workflow-schedule independence. Applying the resulting three principles—periodicity, pipelining, and workflow-schedule independence—in tandem reveals an intrinsically sequential process in a new light.

**Pipelining**

Perhaps pipelining isn’t that aptly named because it gives the impression of making something sequential from a bunch of individual steps. Well, yes, pipelining involves just that kind of transformation when resources are shared, but such instances would represent its trivial reduction to pure periodicity. Pipelining, the way Ladas defines it, is actually more about the opposite transformation, that of making something parallel from something sequential.

Pipelining leverages temporal independence of the steps in an essential workflow. The steps that don’t share resources can be paralleled, resulting in a workflow with staggered branches and less excess capacity. However, we need the second principle as well to demonstrate the economics due to better capacity usage. Pipelining’s caveat is the additional integration step in Figure 1b, which is necessary to recombine the different streams at certain points in the workflow. Pipelining increases the efficiency of a process, provided that the resulting capacity gain compensates for the cost of such integration.
Figure 2 illustrates a pipelined workflow with concurrency. There are six resources: an analyst who can double as a designer or coder, two people who can both design and code, two dedicated testers, and a single integration server. Each path through the pipelined workflow is an instantiation of the essential workflow. A distinct work item will pass through the pipeline only once. A resource can be active in one cell at a time, but multiple resources can be active concurrently. Concurrency will improve resource utilization, but it’s of no use if the work is indivisible. This brings us to the second unraveling principle.

**Workflow-Schedule Independence**

Ladas’s second principle leverages the independence of workflow steps instead of the independence of the workflow’s steps themselves. Such independence results from how work is divided up in smaller chunks in the beginning of a workflow, often during an elaboration or requirements analysis activity. A chunk of work is independent to the extent that it can be completed individually, that is, to the extent it can be transformed into delivered functionality without requiring other pieces to be in place. This extent in turn determines whether the chunks can be scheduled to flow through the development process individually, in groups of smaller batches, or as a single blob.

The more independent the chunks are, the smaller the batches can get, down to the level of the individual requirements. If they form independent clusters of interdependent requirements, then the clusters can be scheduled independently, but not the requirements contained in them. Mark Denne and Jane Cleland-Huang’s concept of minimum marketable features (Software by Numbers: Low-Risk, High-Return Development, Prentice-Hall, 2003) is an example of such a requirements cluster whose interdependencies force the require-
ments to be developed or released together, and therefore whose scheduling might constrain maximum resource utilization. Regardless of the granularity of the work items that flow through the process, the underlying essential workflow is invariant. The essential workflow is replicated again and again for each such work item. Adding workflow-schedule independence to pipelining and periodicity, we get not only an iterative and incremental process but also a parallel one.

Figure 3 illustrates workflow-schedule independence in action for the workflow in Figure 1b, but with the Integrate and Release steps combined into a single Integrate/Release step. The process has a capacity of two resources shared between development and testing and a dedicated resource for integration and release. Articles must be original and not exceed 4,700 words including figures and tables, which count for 200 words each. No again, because not only does each process have a different essential workflow, but it also has multiple intertwined workflows with various cadences and work-item granularities.

So far, we have periodicity, pipelining, and workflow-schedule independence. Do we need anything else? No, that’s it. Does this imply XP or any other iterative and incremental process is an unraveling of a stereotypical waterfall process? No again, because not only does each process have a different essential workflow, but it also has multiple intertwined workflows with various cadences and work-item granularities.

Further Reading

Software Board
Departing board member Martin Robillard has been the associate editor for our Development Infrastructures area since September 2007. He has been instrumental in helping us grow and strengthen this newly established area dedicated to covering software development tools, frameworks, platforms, and environments. I thank Martin for his contributions to IEEE Software and wish him the best in his future endeavors. We’ll announce Martin’s replacement by the end of 2010.

FROM THE EDITOR

IEEE SOFTWARE

NEXT ISSUE

Multiparadigm Programming

STAFF

Lead Editor
Dale C. Strok
dstrook@computer.org

Content Editor
Brian Brannon

Manager, Editorial Services
Jenny Stout
Senior Editor
Linda World

Publications Coordinator
software@computer.org
Production Editor/Webmaster
Jennie Zhu

Contributors
Thomas Centrella, Molly Gamborg, Jean Taylor, Alex Torres

Director, Products & Services
Evan Butterfield
Senior Manager, Editorial Services
Lars Jentsch

Senior Business Development Manager
Sandra Brown

Membership Development Manager
Cecilia Huffman

Senior Advertising Coordinator
Marian Anderson
manderson@computer.org

CS PUBLICATIONS BOARD
David A. Grier (chair), David Bader, Angela R. Burgess, Jean-Luc Gaudiot, Phillip Laplante, Dejan Milojicic, Dorée Duncan Seligmann, Don Shafer, Linda I. Shafer, Steve Tamimoto, and Roy Want

MAGAZINE OPERATIONS COMMITTEE
Dorée Duncan Seligmann (chair), David Albonesi, Isabel Beichl, Carl Chang, Krish Chakrabarty, Nigel Davies, Fred Douglass, Hakan Ergodmus, Lars Jentsch, Carl E. Landwehr, Simon Liu, Dejan Milojicic, John Smith, Gabriel Taubin, Fei-Yue Wang, and Jeffrey R. Yost

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author’s or firm’s opinion. Inclusion in IEEE Software does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society’s Web-based system, Manuscript Central, at http://cs-ieee.manuscriptcentral.com/index.html. Be sure to select the right manuscript type when submitting. Articles must include full details of what is included with tables, figures, and other information, which count for 750 words each.