(To appear in the September/October 2013 issue of **IEEE Software***)*

Sounding Board

Is the New Software Engineering Curriculum Agile?

Armando Fox and David Patterson
UC Berkeley

Although the *Agile Manifesto* was considered controversial when released in 2001, agile is an accepted practice today, taking its place among disciplined and plan-and-document processes such as spiral and the Rational Unified Process. A recent survey of 66 large software projects in industry found that the majority used agile.[1] The latest editions of two popular software engineering textbooks now introduce agile early, with one author summarizing the conventional wisdom that the decision of whether to use agile depends on the circumstances, as Figure 1 shows.[2,3]

1. Is a specification required?
2. Are customers unavailable?
3. Is the system to be built large?
4. Is the system to be built complex (for example, real time)?
5. Will it have a long product lifetime?
6. Are you using poor software tools?
7. Is the project team geographically distributed?
8. Is the team part of a documentation-oriented culture?
9. Does the team have poor programming skills?
10. Is the system to be built subject to regulation?

Figure 1. Sommerville's ten questions concerning whether to use agile. A "no" answer suggests agile; a "yes" suggests plan and document. For student projects, at least 7 of the 10 questions clearly point to agile.[2]

As the last standardization effort was done in 2004, the software engineering curriculum is currently being revised. Haven't we reached the point where agile development should be part of all software engineering curricula? And if so, shouldn't new curriculum standards ensure that it is?

Confusingly, there are two such efforts underway today. The first is for *computer science* departments, which includes courses on software engineering.[10] The second, which is not as far along, is for *software engineering* departments. [11] This article is about the former, but we'll comment on the latter later.

Student Projects Match Agile

For student projects in a software engineering course—assuming the project doesn't consist of building a safety-critical app and that students use high-quality open source tools—virtually all the answers to the questions in Figure 1 point to using agile in the classroom. Indeed, our experience confirms agile's benefits.[4] We developed a new software engineering course at the University of California, Berkeley, in which students work with customers from local nonprofits to build a software-as-a-service app using the tools that come with Ruby on Rails. In particular, students perform four agile iterations in a single semester, which gives them enough experience both to learn and to appreciate the benefits of the approach. The course's success inspired us to write a new software engineering textbook[5] and to create a massive open online course[12]. Others have reported similar results,[6,7] with the bonus that agile particularly benefits women because it addresses factors that limit their participation in computer science.[8,9]

Agile's synergy with student projects is noteworthy because the 2013 computer science curriculum standard, in which ACM and the IEEE Computer Society have joined forces, starts with a call for students to learn software engineering by working in teams on projects:[10]

> *In general, students learn best at the application level much of the material defined in the [software engineering knowledge area] by participating in a project. Such projects should require students to work on a team to develop a software system through as much of its lifecycle as is possible. Much of software engineering is devoted to effective communication among team members and stakeholders. Utilizing project teams, projects can be sufficiently challenging to require the use of effective software engineering techniques and that students develop and practice their communication skills. While organizing and running effective projects within the academic framework can be challenging, the best way to learn to apply software engineering theory and knowledge is in the practical environment of a project.*

Although the standard is agnostic on whether to use plan-and-document or agile processes on these student projects, in practice, it's written from a plan-and-document-centric viewpoint. Hence our keen interest in the following question: If faculty follow the conventional wisdom to the answers to the questions in Figure 1 and use agile in classroom projects, can their course fulfill the new curriculum standard's requirements?

Table 1. The ten sections of the latest draft of the new standard.*

Section title	***Core Tier 1***		***Core Tier 2***		***Electives***		***Total***
	Num-ber	% covered	Num-ber	% covered	Num-ber	% covered	
Software processes	5	100%	2	100%	7	43	14
Software project management	0	--	9	100%	16	94	25
Tools and environments	0	--	4	100%	0	--	4
Requirements engineering	3	100%	3	100%	5	80	11
Software design	5	100%	9	78%	6	67	20
Software construction	0	--	7	86%	3	43	10
Software verification validation	0	--	7	86%	7	--	14
Software evolution	0	--	6	100%	0	60	6
Formal methods	0	--	0	--	5	25	5
Software reliability	0	--	3	67%	4		7
Overall	13	100%	50	94%	53	68%	116

* The **three middle columns** have the number of learning outcomes divided by tier. The percentage covered column lists the percent of the learning outcomes in this tier that would be covered by following the approach in this article. Our plan at UCB covers 100, 94, and 68 percent for the three levels.[13] To pass the standard, courses must cover 100 percent of the Core Tier 1 and at least 80 percent of Core Tier 2 outcomes. Electives are elective.

Digging Deeper Into the Standard

To find the answer, we delved deep into the standard. Table 1 shows its 10 sections, which cover classic software engineering topics. Each section starts with a bulleted list of topics, followed by a list of learning outcomes that make the topics clearer. The standard divides these 116 learning outcomes into three sections:

- 13 are Core Tier 1—courses following the standard must deliver on 100 percent of these learning outcomes
- 50 are Core Tier 2—courses following the standard must deliver on at least 80 percent of these outcomes, and ideally do 90 to 100 percent.
- 53 are Electives— Courses covering only core topics will not have sufficient breadth. These electives are intended to be a useful guide for additional material, but none are required.

We can further classify the learning outcomes according to depth of coverage:

- Familiarity—the student understands what a concept is or what it means. It answers the question, "What do you know about this?" (53 of the 116 outcomes have this goal.)
- Usage—the student is able to use or apply a concept in a concrete way. It answers the question, "What do you know how to do?" (58 of the 116 outcomes have this goal.)
- Assessment —the student is able to consider a concept from multiple viewpoints and/or justify the selection of a particular approach to solve a problem. It answers the question, "Why would you do that?" (5 of the 116 outcomes have this goal.)

Figure 2 gives an example that shows examples of the depth of coverage for all three tiers from one section of the standard.

Speaking as instructors and authors, one upside of the standard is that it led us to expand the material in the course and the book to be more balanced in presenting plan-and-documents along with agile development.

Learning Outcomes:

[Core-Tier1]

1. List the key components of a use case or similar description of some behavior that is required for a system and discuss their role in the requirements engineering process. [Familiarity]
2. Interpret a given requirements model for a simple software system. [Familiarity]
3. Conduct a review of a set of software requirements to determine the quality of the requirements with respect to the characteristics of good requirements. [Usage]

[Core-Tier2]

4. Describe the fundamental challenges of and common techniques used for requirements elicitation. [Familiarity]
5. List the key components of a class diagram or similar description of the data that a system is required to handle. [Familiarity]
6. Identify both functional and non-functional requirements in a given requirements specification for a software system. [Usage]

[Elective]

7. Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system. [Usage]
8. Use a common, non-formal method to model and specify (in the form of a requirements specification document) the requirements for a medium-size software system [Usage]
9. Translate into natural language a software requirements specification (e.g., a software component contract) written in a formal specification language. [Usage]
10. Create a prototype of a software system to mitigate risk in requirements. [Usage]
11. Differentiate between forward and backward tracing and explain their roles in the requirements validation process. [Familiarity]

Figure 2. **Example of three types of learning outcomes for requirements engineering** (section 4 in Table 1). [10] Note that none of the outcomes use agile terms or concepts directly, but with creativity, you can map some of them onto the traditional plan-and-document terms and concepts in these outcomes.

The Committee Forgot Agile

At first blush, the answer about whether agile projects fulfill the standard's outcomes appears to be no. Virtually all the terms come from plan-and-document processes. Here are examples:

- Compare several process improvement models such as CMM, CMMI, CQI, Plan-Do-Check-Act, or ISO9000. [Familiarity]
- Conduct a cost/benefit analysis for a risk mitigation approach. [Usage]
- Create appropriate models for the structure and behavior of software products from their requirements specifications. [Usage]

None of the 116 outcomes use explicit agile terms. Of the 50,000+ words in the standard, the word "agile" only appears twice, and there are only three uses of agile terms in the topics. It's easy—in fact, as the standard is written, it's even natural—to fulfill the 2013 standard with a software engineering course that doesn't mention agile or any of the concepts associated with it: user stories, behavior-driven design, test-driven development, scrum team management, and so on. While plausibly excusable for the prior 2004 standard, it's indefensible for 2013.

If the new standard doesn't include agile, then faculty members face a hard decision:

- follow the outcomes, but ignore the standard's missive and don't offer student projects;
- ignore the standard's outcomes and offer a student project as the standard recommends, using an agile process as the answers to the questions in Figure 1 suggest; or
- follow the outcomes and offer a student project as the standard suggests, but ignore the implications of Figure 1 to use agile.

As the standard document isn't in its final draft, there's still opportunity for change. Given that agile is obviously an accepted practice today and widely used, we strongly recommend a more even balance between plan-and-document and agile processes in the outcomes, particularly for the electives.

As the curriculum committee for software engineering departments started more recently, we certainly hope that it will learn from the omissions of the other committee and embrace agile as a valid part of the curriculum from the start.

Squeezing Agile In While Meeting the Standard

If the computer science standards committee is unyielding, after studying outcomes in more depth, and if you're creative, you can use agile for projects and meet the minimum outcomes. The standard doesn't require an instructor to cover 100 percent of Core Tier 2 or any of the electives, which makes it easier for projects to follow agile and still meet the standard. We combed through all 13 Core Tier 1 outcomes and 54 Core Tier 2 outcomes, and Table 1 shows the percentage that can be covered if you follow this strategy: 100 percent of Core Tier 1, 94 percent of Core Tier 2, and even 68 percent of the electives. The instructor's manual on the web for our course details how we revised the course and the textbook to meet the standard by including sections that contrast agile and plan-and-document methodologies for each of the 10 sections in Table I. [13]

Thus, the answer to the question in the title can be affirmative even if the computer science standards committee is absent-minded. Instructors can follow the initial call of the standard for projects by student teams while using an agile process, which is the most natural match. As long as you review both plan-and-document and agile processes in lecture, students can become familiar with both sets of terms and concepts. The more demanding outcomes can be met by the project as well, provided you look to the deeper meaning behind the plan-and-document terms to see where agile can fit.

References

1. H.-C. Estler et al., "Agile vs. Structured Distributed Software Development: A Case Study," Proc. 7th Int'l Conf. Global Software Eng., IEEE 2012, pp. 11–20.
2. R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed., McGraw-Hill, 2010.
3. I. Sommerville, *Software Engineering*, 9th ed., Addison-Wesley, 2010.
4. A. Fox and D. Patterson, "Crossing the Software Education Chasm," *Comm. ACM*, vol. 55, no. 5, 2012, pp. 44–49.
5. A. Fox and D. Patterson, *Engineering Software as a Service*, beta ed., Strawberry Canyon, 2013.
6. V. Mahnic, "A Capstone Course on Agile Software Development Using Scrum," *IEEE Trans. Software Eng*, vol. 55, no. 1, 2012, pp. 99–106.
7. T. Reichlmayr, "The Agile Approach in an Undergraduate Software Engineering Course Project," *Proc. 33rd Annual Frontiers in Education*, vol. 3, IEEE, 2003, pp. S2C-13.
8. S. Berenson et al., "Voices of Women in a Software Engineering Course: Reflections on Collaboration," *J. Educational Resources in Computing*, vol. 4, no. 1, 2004, 3.1-3.14.
9. L. Werner, B. Hanks, and C. McDowell, "Pair-Programming Helps Female Computer Science Students," *J. Educational Resources in Computing*, vol. 4, no. 1, 2004, 4.1-4.8.
10. Joint Task Force on Computing Curricula, "Computer Science Curricula 2013, Ironman Draft (version 1.0)," ACM/IEEE CS, Feb. 2013, http://ai.stanford.edu/users/sahami/CS2013/.
11. G. Hislop, M. Ardis, D. Budgen, M. Sebern, J. Offutt, & W. Visser,. Revision of the SE 2004 curriculum model. *Proc. 44th ACM Technical Symp. on Computer Science Education* (pp. 383-384). ACM, March 2013.
12. BerkeleyX and EdX CS169.1x and CS169.2x, http://www.saas-class.org/.
13. Instructor's Manual, *Engineering Software as a Service*, http://beta.saasbook.info/acm-ieee-curriculum-compliance.

Armando Fox is a Professor at UC Berkeley. Contact him at fox@cs.berkeley.edu

David Patterson is Professor at UC Berkeley. Contact him at pattrsn@cs.berkeley.edu

Mailing Address: 465 Soda Hall, University of California, Berkeley, California 94720-1776

Keywords: Agile, Curriculum, ACM-IEEE Computer Society, Software Engineering