		Core-Tier1		Core-Tier2		Electives		Total
	Section Title	*Number*	*% Covered*	*Number*	*% Covered*	*Number*	*% Covered*	
1	Software Processes	5	100%	2	100%	7	57%	14
2	Software Project Management	0	--	9	100%	16	94%	25
3	Tools and Environments	0	--	4	100%	0	--	4
4	Requirements Engineering	3	100%	3	100%	5	80%	11
5	Software Design	5	100%	9	78%	6	67%	20
6	Software Construction	0	--	7	100%	3	67%	10
7	Software Verification Validation	0	--	7	86%	7	43%	14
8	Software Evolution	0	--	6	100%	0	--	6
9	Formal Methods	0	--	0	--	5	60%	5
10	Software Reliability	0	--	3	67%	4	25%	7
	TOTAL	13		50		53		116

Figure 1: The ten sections of the Ironman draft of the new standard. The next three columns have the number of learning outcomes divided by tier. The percentage covered column lists percent of the learning outcomes in this tier that would be covered by following the approach in this paper. The ten tables in Figures 2 to 11 provide the detailed documentation of percent covered. Our plan covers 100%, 94%, and 66% for the three levels. To pass the standard, courses must cover 100% of the Core Tier 1 and at least 80% of Core Tier 2 outcomes.

Figure 1 shows the ten sections of the software engineering standard, which cover classic software engineering topics. Each section starts with a bulleted list of topics, followed by a list of a total of 116 learning outcomes, which makes the topics clearer. These learning outcomes are divided into three sections:

1. Core Tier 1: Courses following the standard must deliver on 100% of these 13 of the 116 total outcomes.

2. Core Tier 2: Courses following the standard must deliver on at least 80% of these 50 outcomes, and ideally do 90% to 100%.

3. Electives: Courses covering only core topics will not have sufficient breadth. The 53 electives are intended to be a useful guide for additional material, but none are required.

Finally, the outcomes themselves are further classified three ways regarding the depth of coverage:

1. 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.

2. 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.

3. 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.

We concluded that you could use Agile for projects and meet the minimum outcomes if you are creative. First, our book and the lectures in our course cover both types of processes, so they do fulfill many of the Plan-and-Document outcomes whose target goal is familiarity. Second, we realized that if we mapped the Agile terms onto the nearest Plan-and-Document terms, we could fulfill many outcomes whose goal is usage. A straightforward example is that Agile user stories both elicit customer requirements and document requirements, which are Plan-and-Document terms. Third, some outcomes based on Plan-and-Development are useful in a course even if they are not typically part of the Agile process. For example, a code review or inspection can

	Section 1. Software Processes Outcomes	**Book Cross Reference**
	Core-Tier1	
1	Describe how software can interact with and participate in various systems including information management, embedded, process control, and communications systems	Discussion of programming models for ActiveRecord vs. DataMapper (Chapter 3); Representation of entities and relationships in code vs. in info system (Chapter 4)
2	Describe the difference between principles of the waterfall model and models using iterations	Section 1.2
3	Describe the different practices that are key components of various process models	Section 1.2
4	Differentiate among the phases of software development	Section 1.2
5	Describe how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes	Chapter 9
	Core-Tier2	
6	Explain the concept of a software life cycle and provide an example, illustrating its phases including the deliverables that are produced	Section 1.2
7	Compare several common process models with respect to their value for development of particular classes of software systems taking into account issues such as requirement stability, size, and non-functional characteristics [Usage]	Section 1.2
	Elective	
8	Define software quality and describe the role of quality assurance activities in the software process	Section 1.4
9	Describe the intent and fundamental similarities among process improvement approaches	
10	Compare several process improvement models such as CMM, CMMI, CQI, Plan-Do-Check-Act, or ISO9000	
11	Use a process improvement model such as PSP to assess a development effort and recommend approaches to improvement. [Usage]	
12	Explain the role of process maturity models in process improvement	Section 1.2
13	Describe several process metrics for assessing and controlling a project	Section 10.7
14	Use project metrics to describe the current state of a project	Section 10.7

Figure 2: The 14 learning outcomes from the Software Processes section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

help educate the presenter and the audience, and they are easy for students to do. Finally, the standard does not require an instructor to cover 100% 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 2 shows the percentage that can be covered if you follow this strategy: 100% of Core Tier 1, 94% of Core Tier 2, and even 66% of the electives. The ten tables in Figures 2 to 11 document these statistics by listing the learning outcomes of the standard along side the exercises, sections, and chapters of the book that cover them. It is based on the Ironman draft (version 1.0) of the standard ([ACM IEEE-Computer Society Joint Task Force(2013)]).

Thus, instructors can follow the standard's initial call for student team projects while using an Agile process, which is the most natural match. As long as you review both Plan-and-Document processes 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, as long as you look to the deeper meaning behind the Plan-and-Document terms to see where Agile can fit.

For each learning outcome, if the "Book cross-reference" column of that row is empty, then nothing corresponds. Unless noted otherwise with labels [Usage] or [Competence], the depth of coverage for the outcome is "familiarity."

0.1 Tables Documenting that Agile fulfills the Standard

The ten tables in Figures 2 to 11 provide the detailed documentation of percent covered of tiers covered. Our plan covers 100%, 94%, and 66% for the three levels of Tier 1, Tier 2, and Elective. To pass the standard, courses must cover 100% of the Core Tier 1 and at least 80% of Core Tier 2 outcomes. Electives are elective.

	Section 2. Software Project Management	Book Cross Reference
	Core-Tier1	
	Core-Tier2	
1	Identify behaviors that contribute to the effective functioning of a team	Section 10.7
2	Create and follow an agenda for a team meeting [Usage]	Section 10.7
3	Identify and justify necessary roles in a software development team [Usage]	Section 10.7
4	Understand the sources, hazards, and potential benefits of team conflict [Usage]	Section 10.7
5	Apply a conflict resolution strategy in a team setting [Usage]	Section 10.7
6	Use an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required [Usage]	Section 7.9
7	List several examples of software risks	Section 7.9
8	Describe the impact of risk in a software development life cycle	Section 7.9
9	Describe different categories of risk in software systems	Section 7.9
	Elective	
10	Identify security risks for a software system [Usage]	Chapter 12
11	Demonstrate through involvement in a team project the central elements of team building and team management [Usage]	Section 10.1
12	Identify several possible team organizational structures and team decision-making processes	Sections 10.1, 10.7
13	Create a team by identifying appropriate roles and assigning roles to team members [Usage]	Sections 10.1, 10.7
14	Assess and provide feedback to teams and individuals on their performance in a team setting [Usage]	Section 10.7
15	Prepare a project plan for a software project that includes estimates of size and effort, a schedule, resource allocation, configuration control, change management, and project risk identification and management [Usage]	Section 7.9
16	Track the progress of a project using appropriate project metrics [Usage]	Section 7.9
17	Compare simple software size and cost estimation techniques [Usage]	Sections 7.5, 7.9
18	Use a project management tool to assist in the assignment and tracking of tasks in a software development project [Usage]	Section 7.4
19	Describe the impact of risk tolerance on the software development process [Competence]	Section 7.9
20	Identify risks and describe approaches to managing risk (avoidance, acceptance, transference, mitigation), and characterize the strengths and shortcomings of each	Section 7.9
21	Explain how risk affects decisions in the software development process [Usage]	Section 7.9
22	Demonstrate a systematic approach to the task of identifying hazards and risks in a particular situation [Usage]	Section 7.9
23	Apply the basic principles of risk management in a variety of simple scenarios including a security situation [Usage]	
24	Conduct a cost/benefit analysis for a risk mitigation approach [Usage]	Section 7.9
25	Identify and analyze some of the risks for an entire system that arise from aspects other than the software. [Usage]	Section 7.9

Figure 3: The 25 learning outcomes from the Software Project Management section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 3. Tools and Environments	**Book Cross Reference**
	Core-Tier1	
	Core-Tier2	
1	Describe the difference between centralized and distributed software configuration management	Section 10.3
2	Identify configuration items and use a source code control tool in a small team-based project [Usage]	Sections 10.3, 10.2
3	Describe the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing	When to use/not use user stories and Cucumber (Chapter 7); difficulty of stubbing/mocking in static languages (Chapter 8); when to use /not use Tracker (Chapter 7); augment continuous integration by comparing to traditional build automation (Chapter 12)
4	Demonstrate the capability to use software tools in support of the development of a software product of medium size [Usage]	Chapters 7, 8, 10
	Elective	

Figure 4: The 4 learning outcomes from the Tools and Environments section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 4. Requirements Engineering	**Book Cross Reference**
	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	Sections **??**, 7.9
2	Interpret a given requirements model for a simple software system	Sections **??**, 7.9
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]	Section **??**
	Core-Tier2	
4	Describe the fundamental challenges of and common techniques used for requirements elicitation	Sections **??**, 7.9
5	List the key components of a class diagram or similar description of the data that a system is required to handle	Section 11.2
6	Identify both functional and non-functional requirements in a given requirements specification for a software system [Usage]	Section 7.9, Chapter 12
	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]	Sections **??**, 7.9
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]	Section 7.9
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]	Chapters 7, 8, 10
11	Differentiate between forward and backward tracing and explain their roles in the requirements validation process	Section 7.9

Figure 5: The 11 learning outcomes from the Requirements Engineering section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 5. Software Design	Book Cross Reference
	Core-Tier1	
1	Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation	Chapter 2
2	Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design [Usage]	Aspect orientation (Chapter 4); OO analysis and design (Chapters 2, 4); client-server vs. P2P (Chapter 3); event-driven vs request-reply (Chapter 6)
3	Construct models of the design of a simple software system that are appropriate for the paradigm used to design it [Usage]	Chapter 3
4	For the design of a simple software system within the context of a single design paradigm, describe the software architecture of that system	Chapter 3
5	Within the context of a single design paradigm, describe one or more design patterns that could be applicable to the design of a simple software system	Show how patterns apply in OO Analysis and Design (Chapter 11)
	Core-Tier2	
6	For a simple system suitable for a given scenario, discuss and select an appropriate design paradigm [Usage]	Chapter 11
7	Create appropriate models for the structure and behavior of software products from their requirements specifications [Usage]	
8	Explain the relationships between the requirements for a software product and the designed structure and behavior, in terms of the appropriate models and transformations of them [Competence]	
9	Apply simple examples of patterns in a software design	Chapter 11
10	Given a high-level design, identify the software architecture by differentiating among common software architectures such as 3-tier, pipe-and-filter, and client-server	Chapter 3
11	Investigate the impact of software architectures selection on the design of a simple system [Competence]	Alternatives to client-server, request-reply (Chapter 3)
12	Select suitable components for use in the design of a software product [Usage]	Chapter 11
13	Explain how suitable components might need to be adapted for use in the design of a software product	Chapter 11
14	Design a contract for a typical small software component for use in a given system [Usage]	Chapters 2, 8
	Elective	
15	Discuss and select appropriate software architecture for a simple system suitable for a given scenario. [Usage]	Chapter 11
16	Apply models for internal and external qualities in designing software components to achieve an acceptable tradeoff between conflicting quality aspects [Usage]	
17	Analyze a software design from the perspective of a significant internal quality attribute [Competence]	Chapter 11
18	Analyze a software design from the perspective of a significant external quality attribute	
19	Explain the role of objects in middleware systems and the relationship with components	
20	Apply component-oriented approaches to the design of a range of software, such as using components for concurrency and transactions, for reliable communication services, for database interaction including services for remote query and database management, or for secure communication and access [Usage]	ActiveRecord encapsulates database, handling failed transactions (Chapter 4); How SSL abstracted by ActionController component (Chapter 12)

Figure 6: The 20 learning outcomes from the Software Design section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 6. Software Construction	**Book Cross Reference**
	Core-Tier1	
	Core-Tier2	
1	Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness	Efficiency: discussion of caching and indices, how they affect code (Chapter 12); discussion of main threats to SaaS and how to code against them (Chapter 12)
2	Build robust code using exception-handling mechanisms [Usage]	Chapters 5, 8, 9, 12
3	Describe secure coding and defensive coding practices	Never trust the client (Chapter 3); Security threats like XSS, CSRF, SQL injection (Chapter 12)
4	Select and use a defined coding standard in a small software project [Usage]	Section 10.1
5	Compare and contrast integration strategies including top-down, bottom-up, and sandwich integration	Sections 10.1, 10.7
6	Describe the process of analyzing and implementing changes to code base developed for a specific project	Chapter 9
7	Describe the process of analyzing and implementing changes to a large existing code base	Chapter 9
	Elective	
8	Rewrite a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions [Usage]	Rewriting to avoid SQL injection and XSS (Chapter 12)
9	State and apply the principles of least privilege and fail-safe defaults	Sanitizing SQL and HTML to avoid injection and XSS attacks, deploying using low-privilege processes (Section 12.9); use of `attr_accessible` (Section 5.2)
10	Write a simple library that performs some non-trivial task and will not terminate the calling program regardless of how it is called [Usage]	

Figure 7: The 10 learning outcomes from the Software Construction section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 7. Software Verification Validation	**Book Cross Reference**
	Core-Tier1	
	Core-Tier2	
1	Distinguish between program validation and verification	Section 1.4
2	Describe the role that tools can play in the validation of software	Section **??**
3	Undertake, as part of a team activity, an inspection of a medium-size code segment [Usage]	Section 10.7
4	Describe and distinguish among the different types and levels of testing (unit, integration, systems, and acceptance).	Section 1.4
5	Describe techniques for identifying significant test cases for unit, integration, and system testing	Chapters 7, 8
6	Use a defect-tracking tool to manage software defects in a small software project [Usage]	Use GitHub Issues (Section 10.3) or Tracker (Section 7.4)to track bug resolution
7	Describe the issues and approaches to testing parallel and distributed systems	
	Elective	
8	Create, evaluate, and implement a test plan for a medium-size code segment [Usage]	Section 8.10
9	Compare static and dynamic approaches to verification	
10	Discuss the issues involving the testing of object-oriented software [Usage]	
11	Describe techniques for the verification and validation of non-code artifacts	Section 10.7
12	Describe approaches for fault estimation	
13	Estimate the number of faults in a small software application based on fault density and fault seeding [Usage]	
14	Conduct an inspection or review of software source code for a small or medium sized software project [Usage]	Section 10.7

Figure 8: The 14 learning outcomes from the Software Verification Validation section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 8. Software Evolution	Book Cross Reference
	Core-Tier1	
	Core-Tier2	
1	Identify the principal issues associated with software evolution and explain their impact on the software life cycle	Sections 1.9, 9.1
2	Estimate the impact of a change request to an existing product of medium size [Usage]	Section 9.7
3	Identify weaknesses in a given simple design, and removed them through refactoring	Section 9.6
4	Discuss the challenges of evolving systems in a changing environment [Usage]	Sections 1.9, 9.1
5	Outline the process of regression testing and its role in release management [Usage]	Sections 1.4, 10.7, 9.1, 10.6
6	Discuss the advantages and disadvantages of software reuse	Sections 1.5, 9.1
	Elective	

Figure 9: The 6 learning outcomes from the Software Evolution section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 9. Formal Methods	Book Cross Reference
	Core-Tier1	
	Core-Tier2	
	Elective	
1	Describe the role formal specification and analysis techniques can play in the development of complex software and compare their use as validation and verification techniques with testing	Section 8.10
2	Apply formal specification and analysis techniques to software designs and programs with low complexity [Usage]	
3	Explain the potential benefits and drawbacks of using formal specification languages	Section 7.9
4	Create and evaluate program assertions for a variety of behaviors ranging from simple through complex [Usage]	
5	Using a common formal specification language, formulate the specification of a simple software system and derive examples of test cases from the specification [Usage]	

Figure 10: The 5 learning outcomes from the Formal Methods section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

	Section 10. Software Reliability	Book Cross Reference
	Core-Tier1	
	Core-Tier2	
1	Explain the problems that exist in achieving very high levels of reliability	Section 12.3
2	Describe how software reliability contributes to system reliability	System is whole SaaS server farm, methods like failover and rolling reboot allow SW to mask HW problems (Section 12.3)
3	List approaches to minimizing faults that can be applied at each stage of the software lifecycle	
	Elective	
4	Compare the characteristics of three different reliability modeling approaches	
5	Demonstrate the ability to apply multiple methods to develop reliability estimates for a software system [Usage]	
6	Identify methods that will lead to the realization of a software architecture that achieves a specified reliability level of reliability [Usage]	
7	Identify ways to apply redundancy to achieve fault tolerance for a medium-sized application [Usage]	Horizontal scaling for redundancy (Chapter 12)

Figure 11: The 7 learning outcomes from the Software Reliability section of the Software Engineering curriculum and corresponding concept found in a chapter or section of the book.

References

[ACM IEEE-Computer Society Joint Task Force(2013)] ACM IEEE-Computer Society Joint Task Force. Computer science curricula 2013, Ironman Draft (version 1.0). Technical report, February 2013. URL http://ai.stanford.edu/users/sahami/CS2013/.