Creating an Accreditable Software Engineering Bachelor’s Program

Stephen T. Frezza, Mei-Huei Tang, and Barry J. Brinkman, Gannon University

Since 1936, the accreditation of undergraduate engineering programs has been a cornerstone of modern engineering practice. Accreditation validates that an engineering program serves its constituents and the public well. In an emerging discipline such as software engineering, accreditation standards also serve as a target—a source of requirements for validating an SE program’s design. Accreditation standards also help ensure that a program’s graduates will meet or exceed national and international standards expected of the Bachelor of Science graduate.

For accrediting a BS in Software Engineering (BSSE) program in the US, the primary source is the Accreditation Board for Engineering and Technology. ABET hosts the Engineering Accreditation Commission (EAC), which maintains the accreditation criteria for engineering and related programs.1 (For information on other countries’ accreditation sources, see the sidebar “Software Engineering Accrediting Bodies around the Globe”).

ABET, while authoritative in the US, is not the only source for BSSE program development. The Software Engineering Curriculum Guidelines (SE2004) were developed to support BSSE program development across the globe.2 SE2004 is available as a volume in the Computing Curricula 2005 (CC05) series, which has been developed, maintained, and published by the Joint Task Force for Computing Curricula.3 This is a cooperative project of the Association for Computing Machinery, the Association for Information Systems, and the IEEE Computer Society.4,5 Other useful sources include the Guide to the Software Engineering Body of Knowledge (SWEBOK)5 and other published materials on SE as a discipline.6

However, these sources define only one set of the requirements; they don’t define a complete, accreditable program, nor one that a particular university can successfully field according to its unique vision, organizational constraints, and practical constraints on program delivery. For example, in the initial development of an accreditable BSSE program at Gannon University, the constraints included the university’s mission, its core courses, and the need to maximize reuse of the current courses and program flow.7 The university board of directors approved our initial BSSE program in 2003, and it launched in 2004. The first class will graduate in May 2008 and the program will un-
dergo self-study and ABET visitation in fall 2008.

As with effective software design, one key to success in program design is validation. Using the various requirements sources available, we rigorously applied traceability matrices to match proposed program components (usually courses) to program requirements. The tables also served as a structure for tracking changes and as a discussion focus regarding feasibility issues, pedagogy, course prerequisite structures, and staffing issues. We performed traceability analysis in two areas that support program development: knowledge, which refers to what the graduates need to know, and student (or program) outcomes, which define a broader picture of the graduate.

Defining the BSSE graduate

In general terms, outcomes describe what students are expected to know and be able to do by the time of graduation. Outcomes relate to broadly defined skills, knowledge, and behaviors that students should acquire as they progress through the program. So, they provide a yardstick for measuring a program’s effectiveness. If a graduate achieves all the program outcomes, this indicates that the student meets the program’s stated educational objectives and is equipped to function as expected of a BSSE graduate.

The design of an accreditable SE program must meet the educational objectives (for example, what career and professional accomplishments the student is prepared to achieve) and criteria established for a BSSE program. Essentially, these objectives and criteria define the minimal knowledge and skills for a BSSE graduate. In the US, the EAC provides two broad categories of objectives and criteria that apply to the design of engineering programs. The broader set is the EAC Program Educational Objectives, which apply to all engineering programs. More specific program criteria similarly define objectives for each engineering discipline.

Figure 1 lists the EAC Program Educational Objectives in the left-hand column. These objectives cover a wide spectrum of engineering accomplishments, ranging from applying knowledge and functioning as a team member to understanding ethical responsibility and maintaining knowledge of contemporary issues. The right-hand column of figure 1 lists the three EAC criteria for SE programs. In general, these criteria indicate that software engineers apply general engineering principles to complex software system development in one or more application domains. These are intentionally broad and leave much latitude for BSSE program development.

Although these objectives and criteria are definitive for accreditation, they don’t provide as much detail as the SE2004 guidelines. The SE2004 lists seven outcomes for SE programs, which are shown in figure 1’s middle column. We’ve rearranged the SE2004 criteria’s order of presentation to support a gap analysis between the two sets of criteria. The single arrows in figure 1 illustrate the overlap between the two sets of EAC criteria. Double arrows indicate the relationship between the outcomes that do not completely cover a particu-
lar criterion. Gaps exist where criteria have no relationship or incomplete coverage. For example, two such gaps occur in the omission of “Broad education ...” and “Contemporary issues ...” from the SE2004 student outcomes.

The overlap, as well as the gap between EAC program objectives and SE2004 outcomes, implied that we needed to synthesize the objectives and outcomes into a usable summary of the outcomes expected of a BSSE graduate. We identified 11 source statements as key to this summary (they are shown in bold in figure 1):

1. Show mastery of the software engineering knowledge and skills and professional issues necessary to begin practice as a software engineer.

2. Appropriately apply science, discrete mathematics, empirical techniques, probability and statistics, and relevant topics in computer science and supporting disciplines to the development of complex software systems.

3. Work as an individual and as part of a multidisciplinary team to develop and deliver quality software artifacts.

4. Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, and existing systems and organizations.

5. Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal, and economic concerns.

Figure 1. Comparison of Education Accreditation Commission program objectives, software engineering student outcomes as described in SE2004, and EAC program criteria for SE. The bold statements comprise the summary outcomes expected of a BSSE graduate.
6. Understand professional and ethical responsibility.
7. Demonstrate an understanding of and apply current theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, and documentation.
8. Demonstrate an understanding and appreciation for the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment.
9. Learn new models, techniques, and technologies as they emerge, and appreciate the necessity of such continuing professional development.
10. Grasp contemporary issues.
11. Have a broad education necessary to understand the impact of engineering solutions in a global, economic, environmental, and societal context.

These essential program outcomes, while necessary, are insufficient. An individual institution seeking accreditation is expected to not only cover the outcomes required for accreditation but also customize them to fit its constraints and capitalize on its unique strengths. The customized outcomes also become tools to measure and improve program effectiveness.

Although important for defining and continually improving program effectiveness, the outcomes by themselves don’t provide adequate depth into the “software engineering knowledge and skills” and “professional issues” referenced. What specifically should we teach students? What should the courses contain? SE fortunately has other sources that define more specifically what students must know and be able to do.

**What software engineers need to know**

SE2004 defines the body of knowledge that every SE degree graduate fresh out of college needs to know as the Software Engineering Education Knowledge. SEEK is designed as a guide to support the development of undergraduate SE education curricula. SEEK defines 10 education knowledge areas; each KA is divided into smaller modules called knowledge units, and each KU is further divided into topics. Some topics are designated as “essential” and constitute the core knowledge required to obtain an undergraduate SE degree. Table 1 lists the SEEK KAs, the KUs defined for each KA, and their abbreviations.
SWEBOK, on the other hand, characterizes the contents of the SE discipline—the knowledge needed for practicing SE after four years in the workforce—in 10 KAs. Figure 2 lists the KAs for both SEEK and SWEBOK. The green links connect similar KAs, while KAs without links indicate where the SEEK and SWEBOK KAs differ. Blue links in the figure indicate distributed relationships, and orange indicates links that contain significant although incomplete overlap.

As depicted in figure 2, the differences are noticeable in the SEEK Mathematical and Engineering Fundamentals (FND) and Professional Practice (PRO) KAs relative to the SWEBOK. Some of these differences are intentional: because SWEBOK focuses on what’s within the boundary of SE, its creators intentionally omitted the knowledge of related domains and disciplines. Similarly, the SWEBOK KA Software Requirements is covered as a part of Software Modeling and Analysis (MAA), while SWEBOK’s Software Construction and Software Configuration Management are partially covered by the SEEK Computing Essentials (CMP) and Software Management (MGT) KAs, respectively. The SWEBOK Software Engineering Tools and Methods KA is embodied inside the CMP, MAA, DES, VAV, EVO, PRO, QUA, and MGT topics.

Both SEEK and SWEBOK define the specific knowledge and skills a software engineer requires. For undergraduate SE graduates fresh out of college, the knowledge they attain comes from the courses they completed, so curricula play an important role in deciding what knowledge students would possess when they graduate. While both SEEK and SWEBOK are designed as a guide/foundation for SE curriculum development, SEEK is especially designed for undergraduate SE curriculum development. It defines topics in detail and expects accredited programs to reflect this set of knowledge and skills.

Case study: Designing a BSSE curriculum for accreditation

The design of an accreditable program has as its primary output a series of academic courses that give students opportunities for mastering particular course objectives, which, when combined, lead to the students achieving program objectives. Because the courses provide both the content and support for the outcomes, linking courses to the program requirements (both outcomes and KAs) is both a design and validation method. Similarly, the courses selected for the program must also meet the other nonfunctional program requirements, such as university mission, particular program vision, organizational constraints, and practical constraints on program delivery.

We applied these analysis and design principles to a new BSSE program in our university. In our program design, identifying the known constraints proved to be a useful starting point.

Working within existing constraints

Curriculum design in our university setting has three major driving constraints:

- provide adequate credit hours for university core (general education) courses,
- maximize the reuse of existing courses, and
- limit the program to 135 (semester) credit hours.

While the first constraint limits all programs in our university, it leaves room for specialization and creativity. The second constraint resulted from needing to produce a new program compatible with the existing computing programs. The CS and SE programs’ introductory and core material was very similar, because the entire faculty saw an overlapping need for solid, practical foundations in programming, systems, networking, distributed systems, introductory design, testing, and other SE topics. Furthermore, relatively few students enroll in these programs, so we

Figure 2. Relationships between SEEK and SWEBOK knowledge areas. Green lines connect similar KAs, blue lines indicate distributed relationships, and orange ones indicate links with significant but incomplete overlap. The KAs without lines indicate differences between SEEK and SWEBOK KAs.
needed to make the programs as alike as possible to make optimal use of staff and other resources and to let the students switch majors without needing to significantly extend their time at the university.8 Unfortunately, this decision restricts students' opportunities for SE project experiences early in the program.

Table 2 shows the results of our preliminary design. It lists groups of existing courses that the program reuses, their associated credit hours and core contact hours, and the SEEK KUs that the courses address.

Table 2 indicates that the preliminary program design left at least 97 of 494 core contact hours (≥20 percent), spread among 25 or more SEEK KUs, to be integrated into the program. It also shows there were 19 (135 – 116) or fewer credit hours to accomplish this. So, the remaining design problem was to optimize the delivery of SE-specific courses to meet or exceed the needs of the remaining SEEK KUs and to ensure that program outcomes were met.

A successful design process must balance an individual course’s cognitive load while ensuring adequate coverage throughout a program. In this context, cognitive load refers to the new knowledge and skills that a student must master in a particular course; a successful cognitive load for a particular course is highly instructor- and institution-dependent. However, overloaded courses are more difficult to teach and generally less successful for all but the brightest students. Conversely, underloaded courses tend to bore both the students and faculty and use program resources less efficiently.

### Balancing cognitive loads

For the SE-specific courses, we distributed the remaining SEEK KAs among the remaining credit hours using a detailed traceability analysis of the SEEK topics.8 Table 3 summarizes a traceability analysis for a single course: Personal Software Process. We used traceability analyses for all the program courses that were assigned primary or secondary roles for particular topics. Primary roles are most important because they indicate where the program focuses content delivery for a topic. Other courses also address several of the topics.

This traceability exercise for developing new courses provides detailed topical material

---

### Table 2

<table>
<thead>
<tr>
<th>Existing course groups</th>
<th>Credits</th>
<th>SEEK knowledge units addressed</th>
<th>Core contact hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>General education requirements</td>
<td>36</td>
<td>FND.ec, PRF.com</td>
<td>15</td>
</tr>
<tr>
<td>Computing and engineering fundamentals</td>
<td>35</td>
<td>CMP.cf, CMP.ct, CMP.tt, FND.ef</td>
<td>187</td>
</tr>
<tr>
<td>Mathematics</td>
<td>15</td>
<td>FND mf</td>
<td>56</td>
</tr>
<tr>
<td>Domain electives</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Science</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional practice</td>
<td>7</td>
<td>PRF.psy, PRF.com, PRF.pr, VAV.rev, QUA.cc, MGT.con, MGT.pp, MGTctl</td>
<td>55</td>
</tr>
<tr>
<td>Software engineering</td>
<td>6</td>
<td>MAA.md, MAA.tm, MAA.af, MAA.rfd, DES.con, DES.dd, DES.ste, VAV.fnd, VAV.rev, VAV.tst</td>
<td>84</td>
</tr>
<tr>
<td>Total reused credits</td>
<td>116</td>
<td>Total core contact hours</td>
<td>397</td>
</tr>
<tr>
<td>Remaining credits</td>
<td>19</td>
<td>Remaining hours</td>
<td>97+</td>
</tr>
</tbody>
</table>

---

### Table 3

<table>
<thead>
<tr>
<th>Knowledge unit</th>
<th>Topic number</th>
<th>Topic description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FND.ef</td>
<td>3</td>
<td>Measurement and metrics</td>
</tr>
<tr>
<td>FND.ef</td>
<td>6</td>
<td>Theory of measurement (for example, criteria for valid measurement)</td>
</tr>
<tr>
<td>VAV.rev</td>
<td>1</td>
<td>Desk checking</td>
</tr>
<tr>
<td>VAV.rev</td>
<td>2</td>
<td>Walk-throughs</td>
</tr>
<tr>
<td>PRO.con</td>
<td>4</td>
<td>Measurement and analysis of software processes</td>
</tr>
<tr>
<td>PRO.con</td>
<td>5</td>
<td>Software engineering process improvement (individual and team)</td>
</tr>
<tr>
<td>PRO.imp</td>
<td>3</td>
<td>Life-cycle process models and standards (for example, IEEE and ISO)</td>
</tr>
<tr>
<td>PRO.imp</td>
<td>4</td>
<td>Individual software process (model, definition, measurement, analysis, improvement)</td>
</tr>
<tr>
<td>MGT.pp</td>
<td>4</td>
<td>Effort estimation</td>
</tr>
</tbody>
</table>
that faculty can discuss in designing individual courses and balancing the cognitive trade-offs between existing and proposed courses. The SEEK topic detail, although extensive and at times cumbersome, facilitates discussion of the prerequisites, outcomes, potential texts, and feasibility of each course.

Traceability analysis also supports iterative program improvement. We’ve reviewed the traceability plans biannually for all program courses to help ensure that the current course instantiations are realistic with respect to the program and accreditation expectations.

Table 4 summarizes the traceability exercise, indicating the KUs that each new SE course addresses, either partially or completely.

The design process steps of working with existing constraints and balancing cognitive loads provides a draft of a nearly complete and feasible program. Figure 3 provides a quick glance at the BSSE program we derived using this process (see www.gannon.edu/departmental/ecs/cis/cisdefault.asp and www.gannon.edu/PROGRAMS/UNDER/softengr.asp). The program attempts to assign each SEEK knowledge topic to a primary (and occasionally second-
ary) course responsible to deliver this KA. Having repeated this exercise several times during the initial program development effort, and subsequently during annual program review, we have three observations:

- The courses could not cover all units (in fact, not even all essential units).
- Knowledge topics and units cover a broad spectrum of university courses.
- SE-specific courses (both new and reused) covered the majority of knowledge units.

Refining program outcomes

What remains to complete the design is to validate the program with respect to the program outcomes. We did this at Gannon through a series of proposals and analysis of the outcomes for both applicability and feasibility. We organized “outcome workshops” with participating faculty to develop and refine outcome definitions and to trace these to courses to validate delivery.

The workshops resulted in the development of related BSSE program outcomes as well as outcomes for the BS programs housed in the same academic department—one in computer science, the other in management information systems. Table 5 provides the mapping of these refined program outcomes to their equivalent general BSSE program outcomes.

Our department has since adopted these refined BSSE outcomes and reviews them annually. The outcomes are the cornerstone for a continuous quality improvement (CQI) process aimed at improving the student-learning and program outcomes. These refined outcomes are important because they support both the anticipated ABET accreditation visit and Gannon’s review of the program and its effectiveness. Biannually, we review the refined program outcomes and map them to individual course outcomes, for two reasons: to ensure program completeness and to collect objective evidence needed for CQI work.

Table 5

Mapping Gannon’s BSSE program outcomes to generic BSSE student outcomes

<table>
<thead>
<tr>
<th>Gannon software engineering (BSSE) program outcomes 2006–7</th>
<th>Generic outcomes expected of BSSE graduates*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply problem-solving strategies to software development</td>
<td>1</td>
</tr>
<tr>
<td>Interface with business and analytical professionals to solve software or systems development problems</td>
<td>X</td>
</tr>
<tr>
<td>Comprehend ethical decisions and their ramifications as professionals</td>
<td>X</td>
</tr>
<tr>
<td>Demonstrate effective verbal, written, and listening communication skills as required for professional, group, and team interactions</td>
<td>X</td>
</tr>
<tr>
<td>Demonstrate the ability to continue professional development and expand professional interests</td>
<td>X</td>
</tr>
<tr>
<td>Maintain a comprehension of changing technology and its ramifications</td>
<td>X</td>
</tr>
<tr>
<td>Realize and manage high-quality software development life-cycle processes in one or more application domains</td>
<td>X</td>
</tr>
<tr>
<td>Apply discrete mathematics and abstract structures to system development</td>
<td>X</td>
</tr>
<tr>
<td>Apply quantitative measures in the evaluation of software components and systems</td>
<td>X</td>
</tr>
</tbody>
</table>


While extensive, the work we invested in program design has paid off so far: our institution has a new BSSE program, and early signs show that we have satisfied students. We hope that once the first graduates emerge, it will join the list of ABET-accredited BSSE programs.
Acknowledgments

We thank and acknowledge the pioneering work of the SE program leaders who formed and led the Software Engineering Institute’s Working Group on Software Engineering and Education and Training. In particular, we acknowledge the program leaders at the Rochester Institute of Technology, Drexel University, Embry-Riddle Aeronautical University, and the Milwaukee School of Engineering, from which our program development team drew inspiration, useful models, critiques, and supportive advice.

References


For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.