The Software Factory: Combining Undergraduate Computer Science and Software Engineering Education

John D. Tvedt
The Catholic University of America (CUA)
Department of EE & Computer Science
201 Pangborn Hall
Washington DC 20064 USA
+1 202 319 5299
tvedt@cua.edu
tvedt@orangebunny.com

Roseanne Tesoriero
CUA and
Fraunhofer Center for
Experimental Software Engineering
College Park, MD 20740 USA
+1 301 403 8937
tesoriero@cua.edu
rtesoriero@fc-md.umd.edu

Kevin A. Gary
UNICON, Inc.
3140 N. Arizona Ave.
Suite 113
Chandler, AZ 85225 USA
+1 480 926 2368
garyk@cua.edu
garyk@unicon.net

Abstract

Industry often complains that current university curricula fail to address the practical issues of real software development. This paper outlines a proposal for an innovative core curriculum for a Bachelor of Science in Computer Science. The proposed core curriculum contains elements of traditional computer science programs combined with software engineering via a team-oriented, hands-on approach to large-scale software development. In addition to traditional lecture/project/exam courses, students are required to take an eight-semester sequence of "Software Factory" courses. Software Factory courses put the students' newly acquired skills to work in a real software organization staffed and managed by all students in the program. Students from all courses in the Software Factory sequence meet simultaneously to fulfill their roles in the software organization. We expect the students will be better-prepared software engineering practitioners after completing a curriculum that combines traditional courses with practical Software Factory experience.

1. Introduction

With the explosive growth of the Internet and the permeation of software into nearly every aspect of our lives, the need for qualified software developers to build quality software is apparent. Industry would like educational institutions to train future employees in the latest technology. However, the value system of universities emphasizes long term education rather than training in short term skills. While it may be tempting to focus on the latest technologies to satisfy industry demand, from a pedagogical perspective, doing so would not serve students in the long term. Once the latest technology becomes obsolete, so does the students' knowledge. Industry complains that current university curricula fail to address the practical issues of real software development [4] [5] [9] [15] [16]. With current, undergraduate, computer science curricula, it is rare for students to encounter large-scale development requiring teamwork, written and oral communication skills, maintenance, management, and quality activities. Additionally, traditional computer science courses do not expose students to the non-technical issues that often drive decision making in a real development environment.

Universities have attempted to address industry complaints with curricula in computer science. Assigning team projects in semester courses [6] [11] [7] [12], team projects that span multiple semesters [8] [13], and encouraging internships [14] are a few of the ways universities have attempted to include practical experience along with traditional coursework. However, project courses and internships often begin late in a student's career and are typically short in duration. The exposure does not provide the depth of experience to appreciate the responsibilities of the roles and the implications of their decisions on future development. Nor, do the students have the opportunity to learn from their mistakes and apply their experience to future projects.

The Catholic University of America (CUA) is a small, private institution in the heart of Washington D.C., which has one of the nation's fastest growing technology sectors. This growth is fueled by an established government services market and a rapidly expanding telecommunications market. Of course, much of the technology developed for these sectors has as its core, complex software. Like many higher educational institutions these days, CUA has difficulty retaining and motivating students who see better immediate opportunities in these markets. The challenge facing CUA today is one echoing through the halls of Computer

Science departments throughout the country (if not the world), "How can we make our curriculum meaningful in today's technology-driven world without compromising the essential knowledge and training a student in computer science must receive?"

The purpose of this paper is to describe CUA's answer to this challenge, the Software Factory. The Software Factory combines traditional computer science coursework with experience-based learning in which students participate in the development of a real-world software project. Each student gains experience in every participatory role in the software lifecycle, from serving as a requirements analyst to a software tester to a project manager, and all roles in between. In this paper, we present the Software Factory curriculum.

The main objectives of the Software Factory curriculum are the following:

1. meet industry needs for producing computer scientists familiar with today's technology and processes;
2. ensure computer science students are given a solid and lasting foundation in computer science by providing an accredited computer science program;
3. attract and retain high quality students;
4. conduct empirical software engineering research; and
5. encourage multidisciplinary collaboration.

2. The Software Factory

The Software Factory concept combines traditional computer science coursework with Software Factory courses. The traditional courses cover the fundamental topics of computer science (see Appendix A). Software Factory courses expose students to large-scale, team-oriented development in a software development organization staffed and managed by students under the guidance of faculty. There are eight Software Factory courses that combine to form a software development organization. Each course represents a specific, software engineering role or job within the development organization. The eight semester sequence progresses the students through the following roles: (1) Software Factory process and tools trainee, (2) software system tester, (3 & 4) software developer and maintainer, (5) requirements analyst and test planner, (6) software designer and (7 & 8) software project manager. Students from all courses in the Software Factory sequence meet simultaneously to fulfill their roles in the software organization. The enrollment in the program allows for multiple teams within the organization.

Software Factory courses are hands-on courses that require student participation in the Software Factory throughout their undergraduate career. The Software Factory is a software organization staffed and managed by students in the Computer Science program. Software Factory courses are facilitated by an instructor, but they emphasize learning through real work-experience. These classes meet twice each week. One class meeting lasts one hour and is led by the instructor/"consultant." During this meeting, the "consultant" introduces concepts that are relevant to the current work being performed in the factory and addresses problems faced by the students at the factory. The second class meeting lasts two hours. During this second meeting, all Software Factory classes meet simultaneously in one location, thus fully staffing the factory. During this session, the instructor is a "facilitator" who does not decide right or wrong (as in traditional courses), but instead facilitates learning the pitfalls and peaks in development processes. The facilitator may perform various roles external to the organization, such as "customer", "patent agent", "end-user", "certification agent" (health-case, aviation, etc.), and so forth.

The purpose of the Software Factory is to provide students with practical experience in software development. The students should gain business experience, as well as technical experience. It is important for the students to be exposed to the constraints that business decisions place on technological decisions. The classroom space for the Software Factory simulates a real working environment with cubicles, meeting rooms and office equipment.

The projects for the Software Factory will be chosen by the managers (senior computer science students). These projects may reflect current trends in industry. For example, the Internet has created unique opportunities for students in the Software Factory to gain experience in both new technologies and entrepreneurial areas. The students could create e-commerce web sites to provide services and software. One potential project for the Software Factory would be to build and run an online auction site. The site could be targeted at students, allowing them to auction used textbooks, or other small items. Revenue from the web site could be used to support the Software Factory.

Additional projects might be related to ongoing research at the university. The students could negotiate with faculty to develop software that would support their research projects, thus, encouraging multidisciplinary collaboration. The Software Factory could negotiate with industry for projects that would promote university/industry partnerships.

Software Factory Course Sequence/Description

Each course in the eight-semester sequence through the Software Factory is described below.

1st Semester:
CSC 151: Software Development Process and Tools (3)

This course introduces students to the software development process and the tools that support it.

Students learn about software processes, in general, as well as the process in use at the factory. These students learn about the roles and activities of the members of the factory, such as developers, testers, QA, and management. Finally, the students learn about and use the tools that support the roles and activities at each different stage in the software development process.

2nd Semester:
CSC 152: Software System Testing (3)
In this course, students are put in the role of software testers. Student responsibilities include: writing test plans that test the requirements, writing test cases, running test cases, documenting test results, and following the documented process and standards.

3rd Semester:
CSC 251: Software Development and Maintenance I (Code and Unit Test) (3)
In this course, students are put in the role of software developers. Student responsibilities include: writing software that adheres to the design, unit testing, integration testing, documenting, performing peer reviews, and following the documented process and standards.

4th Semester:
CSC 252: Software Development and Maintenance II (Code and Unit Test) (3)
This course is a continuation of CSC251.

5th Semester:
CSC 351: Software Requirements and Test Planning (3)
In this course, students are put in the role of software requirements analysts and test planners. Student responsibilities include: meeting with customers, analyzing customer requirements, writing a requirements specification document, writing test plans, performing peer reviews, and following the documented process and standards.

6th Semester:
CSC 352: Software Design (3)
In this course, students are put in the role of a software designer. Student responsibilities include: writing a design document that meets the requirements specification, performing peer reviews, and following the documented process and standards.

7th Semester:
CSC 451: Software Project Management I (3)
In this course, students are put in the role of a software project manager. Student responsibilities include: project planning, resource allocation, project estimation, project tracking, risk analysis/mitigation, personnel management, SQA, and planning the future direction of the Software

8th Semester:
CSC 452: Software Project Management II (3)
This course is a continuation of CSC451.

3. Discussion

We believe that the Software Factory concept meets the objectives listed in Section 1. This section describes how the design of the Software Factory meets each objective.

Meet the needs of industry
The needs of industry addressed by the Software Factory design include: exposing students to new technologies, teamwork, large-scale development, management activities, maintenance activities, quality activities and written and oral communication. By having the students meet in one location simultaneously, we simulate a real world, development organization. By participating in the different roles of the Software Factory, students are learning the skills needed by industry as well as gaining an appreciation for their use in industry. For example, configuration management can be taught in a traditional course, but when a student is confronted with a large-scale system, the need for configuration management is better appreciated. The structure of the Software Factory allows the course to be flexible and adaptive to new technology.

Create an accredited program
The coursework designed for the Software Factory follows guidelines for software engineering education [2][10][1] and meets the ABET computer science curriculum requirements [3]. A mapping from the requirements to the coursework is given in Appendix B.

Attract and retain quality students
Students often complain that they do not get enough exposure to coursework in their major until later in their academic careers. The Software Factory concept immediately immerses students into their area of academic interest. By offering a unique program that gives students an opportunity to have early exposure in their area of concentration, we can attract quality students. With exposure to the latest technologies and interesting projects and opportunities, we hope to be able to retain those students.

Conduct empirical software engineering research
The Software Factory will benefit the computer science faculty as well. With a fully staffed Software Factory, there will be multiple teams working within the organization. Those teams could work independently to develop distinct versions of the same product. This setup would provide an opportunity to conduct empirical software engineering studies. For example, an experiment could be run to test the benefits of implementing software inspections. Some teams in the Software Factory could act as a control group while others could develop the same product utilizing inspections.

Encourage multidisciplinary collaboration

Faculty are confronted often with the need for software developers when working on and applying for research grants. As with industry, often the supply does not meet the demand. The Software Factory could provide resources to support those grants. The students would be exposed to problems from various disciplines. The end result is an environment that encourages collaboration among the departments of the university.

4. Implementation plan

Implementing the Software Factory concept requires special consideration. In this section, we describe alternatives for implementing the Software Factory and give a concrete example of how we plan to implement it in our environment.

The Catholic University of America is a small, private university that uses the semester system. There is an existing computer science program in place. The enrollment goal of the computer science program is to have 25 new, freshmen students each Fall. Once we achieve that goal, the Software Factory will have five teams, each consisting of twenty students. These numbers do not take into account attrition, however a small attrition rate should not greatly impact the success of a team. Attrition is one of the biggest problems in industry. It's another real world problem that the students must face.

Physical Environment

Each team will have its own laboratory session in which the students on the team will staff the Software Factory. The physical location for the Software Factory laboratory must be able to accommodate 20 people. The laboratory should be set up like an office space with meeting rooms, cubicles equipped with computers and miscellaneous office equipment and supplies.

Flow of Work

In our instance of the Software Factory, we will start with a waterfall-like process model. Each project will have a two-year cycle time (See Figure 1).

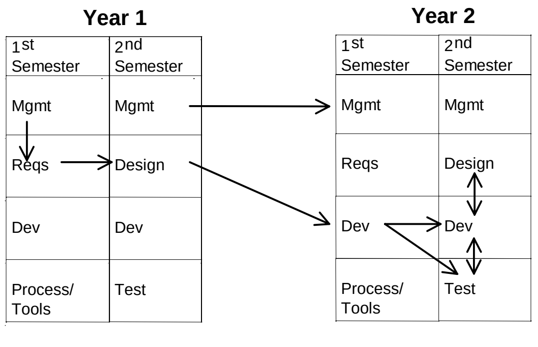

Figure 1. Two-year project work flow

Factory will accept during the first few weeks of the Fall semester. Upon acceptance, those projects will undergo requirements analysis, test planning and design by the junior class during the academic year. The managers will be responsible for guiding and measuring the development and testing of the projects for which they wrote the requirements and design in the previous year. The development and maintenance will be performed by the sophomore class. The testing will be performed by the freshmen class (second semester). The trainees (first semester freshmen) will be assigned to one of the developers as a mentor to learn about the process and the tools used in the Software Factory. The trainees will be exposed to all of the different software engineering roles.

Due to the length of the cycle time, other life cycle models are being considered. However, we do not expect to implement them initially.

Grading

The Software Factory courses are meant to be participatory classes. The management will be responsible for writing performance reviews for each team member, each semester. These performance evaluations will be the basis for the students' course grades assigned by the faculty member. The managers are better-suited to evaluate the team members than the faculty due to their working relationship. By evaluating their team members, managers will gain valuable experience in interpersonal communication skills and writing performance evaluations. The performance review will give the managers leverage to motivate the team members.

Managers' grades will be based on evaluations from team members. This grading method provides checks and balances between the management and team members.

Implementation Alternatives

Many universities already have an existing computer science program in place. Therefore, it is important to analyze the existing situation and choose an alternative that provides the most benefits and least disruption to the university and students. We have considered three alternative approaches for implementing the Software Factory in our environment: the incremental, big-bang and sandwich approaches. The incremental approach is defined as introducing the Software Factory in incremental stages, starting with the freshmen factory courses (Process/Tools and Software Testing). Each additional year, more courses are offered until all of the courses for the Software Factory are offered. The big-bang approach is the opposite of the incremental approach. In this case, all four years of Software Factory courses are offered immediately. The sandwich approach is a compromise between the incremental and big-bang approaches. With this approach, in the first year, only the freshmen and the senior (Management I and Management II) courses are introduced. In the second year, the full set of Software

Factory courses are offered. Each alternative has its own set of advantages and disadvantages.

Incremental Approach

Some of the advantages to implementing the Software Factory courses incrementally include a reduced course load and the possibility of smaller lab space requirements. By only offering the freshmen courses in the first year, only two additional courses are added instead of the full eight courses required with full implementation of the Software Factory. Additionally, since only the freshmen would be involved, the university could incrementally allocate resources and equipment to set up the lab.

Although there are some advantages to an incremental approach, there are also several disadvantages. The most serious concern is the amount of time it will take to be able to evaluate the full benefits and drawbacks of having the Software Factory. This approach would require four years to fully staff the Software Factory. This approach also delays using the Software Factory for experimentation and research purposes. A more practical issue is that although only two courses are introduced, the work products for the freshmen to use for the testing course in the second semester of the first year would have to be developed or obtained. And, in each additional year, the work products for all of the courses that have not yet been introduced would have to be developed or obtained. Implementing the incremental approach starting with the senior year and working backward to the freshman year or using one of the alternative approaches eliminates this need.

Big Bang Approach

Introducing all of the new Software Factory courses has several advantages over the incremental approach. During the first semester, the students will be producing the work products required for the second semester courses. With this approach, the only work product that would have to be developed would be a design document for the developers to use for the first semester. Many universities already offer courses in which a design document is used or developed. These existing design documents could be used for the developers in the first year. Another advantage is that the second semester courses can use the work products from the first semester. This approach provides quick feedback since the Software Factory will be staffed at all levels in the first year. This approach also makes the Software Factory available for experimentation and research more quickly than other approaches.

There are some drawbacks to this approach. The most serious is that introducing eight new courses may be difficult for existing faculty to cover. One solution would be to rely on adjuncts and lecturers to teach some of the courses. This approach also assumes that current students will want to participate in the Software Factory. While one can't expect full participation from the existing students, some students will have to participate in order to staff the Software Factory fully.

Sandwich Approach

The sandwich approach, while reducing the initial course load that is required by the big-bang approach, still has many of the problems of the incremental approach. If there are no developers in the first semester of the Software Factory, a work product for the second semester testers will have to be developed or obtained by the faculty. This approach reduces by half the time to have a fully operational Software Factory when compared to the incremental approach, but it would still take two full years. In addition, the faculty would have to develop or provide work products for the testers in the second semester of the first year and design documents for the developers in the second year.

After considering the three alternatives, the big-bang approach was considered to be the most desirable for our situation. Using this approach will require some students to incorporate some of the new courses into their existing programs. We plan to count the Software Factory courses as fulfilling current computer science electives. Students entering into the Software Factory course sequence will enter at a level based on a recommendation from their academic advisor. Transfer students will be handled in the same way as existing, computer science students.

5. Evaluation Plan

We plan to use our objectives in Section 1 to guide our evaluation plan. Based on our objectives, we would like to answer the following questions:

Meet the needs of industry

1. Is industry interested in collaborating with the Software Factory?

We will keep track of the number of inquiries from industry. We will monitor the number of projects and experiments completed for industry in the Software Factory.

Create an accredited program

1. Does the new curriculum satisfy ABET curriculum requirements for accreditation?

We have mapped the courses for the new curriculum to the ABET curriculum requirements. The Software Factory curriculum meets these requirements. See Appendix B.

2. Is industry happy with our graduates?

To address this question, we need to get feedback from the employers and potential employers of our graduates.

3. Are our graduates trained in the technologies that industry is interested in?

To address this question, we will compare the technologies used in the Software Factory with those that are in-demand in industry.

Attract and retain quality students

1. Have the number of applicants in computer science increased?

We will look at the total number of computer science applicants each year over the 5 years prior to the introduction of the Software Factory and compare it to the total number of computer applicants each year since the introduction of the Software Factory.

2. Have the GPA and SAT scores of applicants in computer science increased?

We will look at the GPA, SAT scores and quality ratings of computer science students for the 5 years prior to the introduction of the Software Factory. We will compare these scores and ratings to the GPA, SAT scores and quality ratings of applicants since the introduction of the Software Factory.

3. Has the retention rate been maintained or improved?

We will look at the retention rate of computer science students for the 5 years prior to the introduction of the Software Factory. We will look for the same or better rate of retention in each year since the introduction of the new curriculum.

Conduct empirical software engineering research

1. Has the factory been used to conduct empirical software engineering research?

The number of experiments completed in the Software Factory will be monitored. We will also keep track of the number of papers generated from work done in the factory.

Encourage multidisciplinary collaboration

1. Are other departments interested in the factory?

We will keep track of the number of grants supporting the factory, the dollar amount of funding for the factory and the number of departments utilizing the factory.

6. Summary and Future Work

With current, undergraduate, computer science curricula, students graduate with technical skills, but lack practical software engineering skills needed in industry. We believe the Software Factory concept will benefit the students, the faculty, the university and industry. Students will learn more and retain more by putting their newly acquired skills to use in a real software development organization. Faculty will have the opportunity to contract out software to students running the Software Factory and conduct software engineering experimentation. The university may be able to attract a greater number of quality students. Finally, industry will benefit by gaining access to students that have experience in a real world, software development organization upon graduation.

The new curriculum was approved by the Department of Electrical Engineering and Computer Science at The Catholic University of America during the Spring of 2000. We are working with other colleges and universities to help them with the implementation of the Software Factory curriculum.

7. References

[1] Bagert D. J., A model for the software engineering component of a software engineering curriculum. *Information and Software Technology,* 40, 4, 195 – 201.

[2] Bagert D.J., Hilburn T.B., Hislop G., Lutz M., McCracken M., and Mengel S. Guidelines for software engineering education. Version 1.0. Technical Report CMU/SEI-99-TR-032, October 1999.

[3] Computing Accreditation Commission Acreditation Board for Engineering and Technology (ABET), 2000-2001 criteria for accrediting computing programs. November 1, 2000.

[4] Coulter N. and Dammann J. Current practices, culture changes, and software engineering education. *Computer Science Education, 5, 2,* 1994, 211 – 227.

[5] Dawson R.J., Newsham, R.W. and Kerridge, R.S. Introducing new software engineering graduates to the 'real world' at the GPT company. *Software Engineering Journal, 7, 3,* (1992), 171 – 176.

[6] Dawson, R. and Newsham, R. Introducing software engineers to the real world. *IEEE Software* (November/December 1997), 37 – 43.

[7] Easterbrook S.M. and Arvanitis T.N. Preparing students for software engineering. *In Proc. of the Third International Workshop on Software Engineering Education (IWSEE3),* (Berlin, Germany, March 1996).

[8] Garlan, D., Glutch D.P. and Tomayko, J.E. Agents of change: Educating software engineering leaders. *IEEE Computer,* 30, 11, 59 – 65.

[9] Gibbs N. The SEI education program: The challenge of teaching future software engineers, *Comm. of the ACM,* (May 1989), 594 – 605.

[10] IEEE/ACM Software Engineering Coordinating Committee, Accreditation criteria for software engineering. Sep. 1998. http://computer.org/tab/Accred10.html.

[11] Hilburn, T.B. Software engineering education: A modest Proposal. *IEEE Software* (Nov/Dec 1997), 44 – 48.

[12] Horning, J.J. and Wortman D.B. Software hut: A computer program engineering project in the form of a game. *IEEE Trans. on Software Engineering, SE-3, 7,* (July 1997), 325 – 330.

[13] Moore, M. and Potts, C. Learning by doing: Goals and experiences of two software engineering project courses. In *Proceedings of the Seventh Software engineering Institute Conference on Software Engineering Education,* (San Antonio, Texas, January 1994).

[14] Powell, G.M., Diaz-Herrera, J.L., and Turner D.J. Achieving synergy in collaborative education. *IEEE Software* (November/December 1997), 58 – 65.

[15] Shaw M. Prospects for an engineering discipline of software. *IEEE Software* (November 1990), 15 – 24.

[16] Wasserman A.I. Toward a discipline of software engineering. *IEEE Software* (November 1996), 23 – 31.

Appendix A Course Sequence/Description

1st Semester:
xxx: Required Elective
xxx: Required Elective
MATH 121: Calculus I (4)
CSC 131: Computer Science I (Java) (3)
CSC 151: Software Development Process and Tools (3)

2nd Semester:
xxx: Required Elective
xxx: Required Elective
MATH 122: Calculus II (4)
CSC 132: Computer Science II (Java) (3)
CSC 152: Software System Testing (3)

3rd Semester:
xxx: Required Elective
xxx: Required Elective
CSC 211: Discrete Structures (3)
CSC 231: Data Structures (3)
CSC 251: Software Development and Maintenance I (Code and Unit Test) (3)

4th Semester:
xxx: Required Elective
xxx: Required Elective
CSC 212: Theoretical Computer Science (3)
CSC 222: Computer Organization and Assembly Language (3)
CSC 252: Software Development and Maintenance II (Code and Unit Test) (3)

5th Semester:
xxx: Required Elective
MATH 501: Linear Algebra (3)
CSC 311: Design and Analysis of Algorithms (3)
CSC 331: Programming Languages (3)
CSC 351: Software Requirements and Test Planning (3)

6th Semester:
xxx: Required Elective
MATH 531: Probability and Statistics (3)
CSC xxx: Computer Science Elective (3)
CSC xxx: Computer Science Elective (3)
CSC 352: Software Design (3)

7th Semester:
xxx: Required Elective
xxx: Required Elective
CSC xxx: Computer Science Elective (3)
CSC xxx: Computer Science Elective (3)
CSC 451: Software Project Management I (3)

8th Semester:
xxx: Required Elective
xxx: Required Elective
CSC xxx: Computer Science Elective (3)
CSC xxx: Computer Science Elective (3)
CSC 452: Software Project Management II (3)

Required electives: (44)

Science:
SCI xxx: 1st semester of a lab course (4)
SCI xxx: 2nd semester of a lab course (4)
SCI xxx: Science Elective (3)
SCI xxx: Science Elective (3)

English:
ENG 101 (or 105,103,104): English Composition (3)

Religion:
REL 201 (or 203): Christian Difference (201 cannot be taken 1st semester freshman year) (3)
REL xxx: Religion Elective (3)
REL xxx: Religion Elective (3)

Philosophy:
PHIL 362: Professional Ethics in Engineering (3)

Liberal Studies:
LS xxx: Liberal Studies Elective (3)
LS xxx: Liberal Studies Elective (3)
LS xxx: Liberal Studies Elective (3)
LS xxx: Liberal Studies Elective (3)
LS xxx: Liberal Studies Elective (3)

Additional requirements

1. To be accepted as a major, a student must have completed CSC 131, CSC 132, CSC 211, and CSC 231 with a minimum G.P.A. of 2.5 in these three courses.
2. To ensure competence in the core material, all core math and computer science courses must be passed with a grade of *C* or better to satisfy the requirements of the degree.
3. To ensure breadth in the choice of the six computer science electives, at least one course, and no more than two courses, must be taken from each of the following areas: Computer Science Foundations (CSC x1x), Computer Systems (CSC x2x), Software Systems (CSC x3x), and Computing Methodologies (CSC x4x).

Appendix B Mapping to ABET curriculum requirements [3]

The table below contains section IV from the ABET requirements for accreditation. The standards are given on the left and the evidence of the Software Factory curriculum on the right.

IV. Curriculum	
Intent: The curriculum is consistent with program's documented objectives. It combines technical requirements with general education requirements and electives to prepare students for a professional career in the computer field, for further study in computer science, and for functioning in modern society. The technical requirements include up-to-date coverage of basic and advanced topics in computer science as well as an emphasis on science and mathematics.	
Standards:	**Evidence:**
General	
IV-1. The curriculum must include at least 40 semester hours of up-to-date study in computer science topics.	**63 semester hours** CSC 131: Computer Science I (Java) (3) CSC 151: Software Development Process and Tools (3) CSC 132: Computer Science II (Java) (3) CSC 152: Software System Testing (3) CSC 231: Data Structures (3) CSC 251: Software Development and Maintenance I (Code and Unit Test) (3) CSC 212: Theoretical Computer Science (3) CSC 222: Computer Organization and Assembly Language (3) CSC 252: Software Development and Maintenance II (Code and Unit Test) (3) CSC 311: Design and Analysis of Algorithms (3) CSC 331: Programming Languages (3) CSC 351: Software Requirements and Test Planning (3) CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) CSC 352: Software Design (3) CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) CSC 451: Software Project Management I (3) CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) CSC 452: Software Project Management II (3)
IV-2. The curriculum must contain at least 30 semester hours of study in mathematics and science as specified below under Mathematics and Science.	**31 semester hours** **Math:** MATH 121: Calculus I (4) MATH 122: Calculus II (4) CSC 211: Discrete Structures (3) MATH 501: Linear Algebra (3) MATH 531: Probability and Statistics (3) **Science:** SCI xxx: 1st semester of a lab course (4) SCI xxx: 2nd semester of a lab course (4) SCI xxx: Science Elective (3) SCI xxx: Science Elective (3)

IV-3. The curriculum must include at least 30 semester hours of study in humanities, social sciences, arts and other disciplines that serve to broaden the background of the student.	**30 semester hours** **English:** ENG 101 (or 105,103,104): English Composition (3) **Religion:** REL 201 (or 203): Christian Difference (201 cannot be taken 1st semester freshman year) (3) REL xxx: Religion Elective (3) REL xxx: Religion Elective (3) **Philosophy:** PHIL 362: Professional Ethics in Engineering (3) **Liberal Studies:** LS xxx: Liberal Studies Elective (3) LS xxx: Liberal Studies Elective (3) LS xxx: Liberal Studies Elective (3) LS xxx: Liberal Studies Elective (3) LS xxx: Liberal Studies Elective (3)
IV-4. The curriculum must be consistent with the documented objectives of the program.	
Computer Science	
IV-5. All students must take a broad-based core of fundamental computer science material consisting of at least 16 semester hours.	**45 semester hours** CSC 131: Computer Science I (Java) (3) CSC 151: Software Development Process and Tools (3) CSC 132: Computer Science II (Java) (3) CSC 152: Software System Testing (3) CSC 231: Data Structures (3) CSC 251: Software Development and Maintenance I (Code and Unit Test) (3) CSC 212: Theoretical Computer Science (3) CSC 222: Computer Organization and Assembly Language (3) CSC 252: Software Development and Maintenance II (Code and Unit Test) (3) CSC 311: Design and Analysis of Algorithms (3) CSC 331: Programming Languages (3) CSC 351: Software Requirements and Test Planning (3) CSC 352: Software Design (3) CSC 451: Software Project Management I (3) CSC 452: Software Project Management II (3)
IV-6. The core materials must provide basic coverage of algorithms, data structures, software design, concepts of programming languages, and computer organization and architecture.	CSC 311: Design and Analysis of Algorithms (3) CSC 231: Data Structures (3) CSC 352: Software Design (3) CSC 331: Programming Languages (3) CSC 222: Computer Organization and Assembly Language (3)
IV-7. Theoretical foundations, problem analysis, and solution design must be stressed within the program's core materials.	CSC 212: Theoretical Computer Science (3) CSC 311: Design and Analysis of Algorithms (3) CSC 351: Software Requirements and Test Planning (3) CSC 352: Software Design (3)
IV-8. Students must be exposed to a variety of programming languages and systems and must become proficient in at least one higher-level language.	CSC 331: Programming Languages (3) CSC 131: Computer Science I (Java) (3) CSC 132: Computer Science II (Java) (3)
IV-9. All students must take at least 16 semester hours of advanced course work in computer science that	**18 semester hours** CSC xxx: Computer Science Elective (3)

provides breadth and builds on the core to provide depth.	CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) CSC xxx: Computer Science Elective (3) To ensure breadth in the choice of the six computer science electives, at least one course, and no more than two courses, must be taken from each of the following areas: Computer Science Foundations (CSC x1x), Computer Systems (CSC x2x), Software Systems (CSC x3x), and Computing Methodologies (CSC x4x).
Mathematics and Science	
IV-10. The curriculum must include at least 15 semester hours of mathematics.	**17 semester hours** MATH 121: Calculus I (4) MATH 122: Calculus II (4) CSC 211: Discrete Structures (3) MATH 501: Linear Algebra (3) MATH 531: Probability and Statistics (3)
IV-11. Course work in mathematics must include discrete mathematics, differential and integral calculus, and probability and statistics.	CSC 211: Discrete Structures (3) MATH 121: Calculus I (4) MATH 122: Calculus II (4) MATH 531: Probability and Statistics (3)
IV-12. The curriculum must include at least 12 semester hours of science.	**14 semester hours** SCI xxx: 1st semester of a lab course (4) SCI xxx: 2nd semester of a lab course (4) SCI xxx: Science Elective (3) SCI xxx: Science Elective (3)
IV-13. Course work in science must include the equivalent of a two-semester sequence in a laboratory science for science or engineering majors.	SCI xxx: 1st semester of a lab course (4) SCI xxx: 2nd semester of a lab course (4)
IV-14. Science course work additional to that specified in Standard IV-13 must be in science courses or courses that enhance the student's ability to apply the scientific method.	SCI xxx: Science Elective (3) SCI xxx: Science Elective (3)
Additional Areas of Study	
IV-15. The oral communications skills of the student must be developed and applied in the program.	ENG 101 (or 105,103,104): English Composition (3) CSC 451: Software Project Management I (3) CSC 452: Software Project Management II (3)
IV-16. The written communications skills of the student must be developed and applied in the program.	ENG 101 (or 105,103,104): English Composition (3) CSC 352: Software Design (3) CSC 351: Software Requirements and Test Planning (3) CSC 451: Software Project Management I (3) CSC 452: Software Project Management II (3)
IV-17. There must be sufficient coverage of social and ethical implications of computing to give students an understanding of a broad range of issues in this area.	PHIL 362: Professional Ethics in Engineering (3)