Software Engineering for the 21st Century: A basis for rethinking the curriculum

Mary Shaw (editor)

February 2005

CMU-ISRI-05-108

with contributions from discussions with Jonathan Aldrich, Ray Bareiss, Shawn Butler, Lynn Carter, Owen Cheng, Steve Cross, Jamie Dinkelacker, Dave Farber, David Garlan, John Grasso, Martin Griss, Tim Halloran, Jim Herbsleb, Carol Hoover, Lisa Jacinto, Mark Klein, Deniz Lanyi, Beth Latronico, Jim Morris, Priya Narasimhan, Joe Newcomer, Linda Northrup, Ipek Ozkaya, Mark Paulk, David Root, Mel Rosso-Llopart, Walt Shearer, Bill Scherlis, Todd Sedano, Gil Taran, Jim Tomayko, and Tony Wasserman

Institute for Software Research International
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

This work was supported by principally by the educational programs of the Institute for Software Research International and by the A. J. Perlis Chair of Computer Science

Keywords: software engineering education, software engineering, education

Abstract

Progress in both software and hardware technology over the past decade make it timely to re-examine our curriculum in software engineering and related topics. This manifesto describes the Carnegie Mellon approach to software engineering, the essential capabilities of a software engineer, and the pedagogical principles that guide our curriculum design.

Our objective here is to articulate Carnegie Mellon's core academic values for the discipline of software engineering. This characterization of software engineering covers undergraduate, professional, and research curricula. It is informed by other software engineering curriculum designs, but it is independent of them. Curriculum design must reconcile the objectives of numerous stakeholders; this document states the case of the academic-values stakeholder.

1 Definition

Software engineering is the branch of computer science that creates practical, cost-effective solutions to computing and information processing problems, preferentially by applying scientific knowledge, developing[1] *software systems in the service of mankind.* Software engineering entails making decisions under constraints of limited time, knowledge, and resources. The distinctive character of software raises special issues about its engineering. These characteristics and issues include

- Software is design-intensive; manufacturing costs are a minor component of software product costs.
- Software is symbolic, abstract, and more constrained by intellectual complexity than by fundamental physical laws.

Software engineering rests on three principal intellectual foundations. The technical foundation is a body of *core computer science* concepts relating to data structures, algorithms, programming languages and their semantics, analysis, computability, computational models, etc.; this is the core content of the discipline. This technical knowledge is applied through a body of *engineering knowledge* related to architecture, the process of engineering, tradeoffs and costs, conventionalization and standards, quality and assurance, etc.; this provides the approach to design and problem solving that respects the pragmatic issues of the applications. These are complemented by the *social and economic context* of the engineering effort, which includes the process of creating and evolving artifacts, as well as issues related to policy, markets, usability, and socio-economic impacts; this provides a basis for shaping the engineered artifacts to be fit for their intended use.

Software engineering is often confused with mere programming or with software management. Both comparisons are inappropriate, as the responsibilities of an engineer include the deliberate, collaborative creation and evolution of software-intensive systems that satisfies a wide range of technical, business, and regulatory requirements. Software engineering is not simply the implementation of application functionality, nor is it simply the ability to manage a project in an orderly, predictable fashion.

[1] "Develop" -- Software engineering lacks a verb that covers all the activities associated with a software product, from conception through client negotiation, design, implementation, validation, operation, evolution, and other maintenance. Here, "develop" refers inclusively to all those activities. This is less than wholly satisfactory, but it isn't as bad as listing several verbs at every occurrence.

2 Core Principles: The Carnegie Mellon Way of Software Engineering

These are the broad, fundamental, pervasive, integrative principles that transcend specific details and characterize the field. These are core beliefs that shape our values about what things are important and how we approach problems. These principles characterize the distinctive Carnegie Mellon approach to software engineering

Physicists often approach problems (not just physical problems) by trying to identify masses and forces. Mathematicians often approach problems (even the same problems) by trying to identify functional elements and relations. Engineers often approach problems by trying to identify the linearly independent underlying components that can be composed to solve a problem. Programmers often view them operationally, looking for state, sequence, and processes. Here we try to capture the characteristic mindset of a software engineer.

2.1 Computer Science Fundamentals

The core body of systematic knowledge that supports software engineering is the algorithmic, representational, symbol-processing knowledge of computer science, together with specific knowledge about software and hardware systems.

Abstraction enables the control of complexity. Abstraction allows selective control of detail and consequently separation of concerns and crisp focus on design decisions. It leads to models and simulations that are selective about the respects in which they are faithful to reality. It permits design and analysis in a problem-oriented frame rather than an implementation-oriented frame. Some levels of design abstraction, characterized by common phenomena, notations, and concerns, occur repeatedly and independent of underlying technology.

Imposing structure on problems often makes them more tractable, and a number of common structures are available. Designing systems as related sets of independent components allows separation of independent concerns; hierarchy and other relations help explain the relations among the components. In practice, independence is impractical, so issues of cohesion and coupling affect the results. Moreover, recognizing common problem and solution structures allows reuse of prior knowledge rather than reinvention. Software systems are sufficiently complex that they exhibit emergent properties that do not derive in obvious ways from the properties of the components.

Symbolic representations are necessary and sufficient for solving information-based problems. Control and data are represented symbolically, and this enables their duality. Notations for symbolic description of control and data enable the definition of software. These representations allow the description of algorithms and data structures, the bread and butter of software implementation.

Precise models support analysis and prediction. These models may be formal or empirical; formal and empirical models are subject to different standards of proof and provide different levels of assurance in their results. The results support software design by providing predictions of properties of a system early in the system design. Careful documentation and codification of informal knowledge provides immediate guidance for developers and a precursor for more precise, validated models.

Common problem structures lead to canonical solutions. Recognizing common problem and solution structures allows reuse of prior knowledge rather than reinvention.

2.2 Engineering Fundamentals

The systematic method and attention to pragmatic solutions that shapes software engineering practice is the practical, goal-directed method of engineering, together with specific knowledge about design and evaluation techniques.

Engineering quality resides in engineering judgment. Tools, techniques, methods, models, and processes are means that support this end. They can enhance sound judgment, and they may make development activities more accurate and efficient, but they cannot replace sound judgment.

Quality of the software product depends on the engineer's faithfulness to the engineered artifact. This quality is achieved through commitment to understanding the client's needs; it is evaluated by assessing the properties of the artifact that are important to the client. This is the basis for ethical practice.

Engineering requires reconciling conflicting constraints. These constraints arise both from requirements and from implementation considerations. They typically overconstrain the system, so the engineer must find reasonable compromises that reflect the client's priorities. Engineers generate and compare alternative designs and refine the most promising; they prefer quantitative evaluations and predictions. Finding sufficiently good cost-effective solutions is usually preferable to optimization.

Engineering skills improve as a result of careful systematic reflection on experience. A normal part of any project should be critical evaluation of the work. Critical evaluation of prior and competing work is also important, especially as it informs current design decisions.

2.3 Social and Economic Fundamentals

The concern with usability and the business and political context that guides software engineering sensibilities is organizational and cognitive knowledge about human and social institutions. This is supported by specific knowledge about human-computer interaction techniques.

Costs and time constraints matter, not just capability. The costs include costs of ownership as well as costs of creation. Time constraints include calendar (e.g., market window) as well as staffing constraints. These factors affect the system design as well as the project organization.

Technology improves exponentially, but human capability does not. Computing and information processing capability should be delivered to end users in a form that the end users can understand and control; systems should adapt to the users, not users to the systems. The activities of computing should fit well with the users' other activities.

Successful software development depends on teamwork by creative people. Software developers must be able to reconcile business objectives, client needs, and the factors that make creative people effective.

Business and policy objectives constrain software design and development decisions as much as technical considerations do. Long-range objectives, competitive market position, and risk management affect the business case for a software development. Public policy and regulation add requirements that the client may not be aware of. These objectives should have equal standing with other objectives, such as technical and usability objectives, in the development process.

Software functionality is often so deeply embedded in institutional, social, and organizational arrangements that observational methods with roots in anthropology, sociology, psychology, and other disciplines are required. It is often relatively easy to capture the obvious functionality and constraints, but the subtle ones often go unnoticed and cause projects to fail or to incompletely satisfy users.

Customers and users usually don't know precisely what they want, and it is the developer's responsibility to facilitate the discovery of the requirements. Developers need to use appropriate techniques to help the customers and users explore the design space, and understand the relevant alternatives, constraints, and tradeoffs. This requires knowledge both of the technology and the context of use. Since most customers and users are unlikely to acquire substantial technical knowledge, developers must take the initiative to bridge the gap by working to acquire more than a superficial knowledge of the context of use.

3 Core Competencies

Software engineers should be masters of a set of core competencies. These are abstract capabilities (e.g. "ability to reason in a formal system") not specific skills (e.g., any particular choice among CSP, Z, Larch, etc or among specific software development methods), and especially not skills in using particular products (e.g., specific frameworks). The core competencies inform and pervade the curriculum. More visibly, the curriculum includes content, both mature and immature, that develops software engineering capability on the three foundations of core computer science, engineering, and the human and social context.

Different students may master the core competencies in different ways. So we envision descriptions of degree programs that include explanations of how the curriculum covers these competencies. We might say, for example, that each masters student should demonstrate proficiency in reasoning in symbolic systems by using two such systems at some point during the masters program; this might be in a class, in the major studio project, as part of an independent study project, etc. This model becomes increasingly important as the flexibility in the programs and the diversity of student activity increases.

To support this, we envision a mapping from our educational offerings (courses, projects, etc) to the competencies. Imagine a table with competencies as rows, each teaching unit as a column that shows how that unit contributes to the competencies, and a special column that shows the coverage requirement for that competency

To describe the content, we develop a rough classification that allows us to plan curricula, to assess students' skills, and to identify intellectual gaps. Software engineers should have these capabilities:

- Be able to discover client needs and translate them to software and system requirements
- Reconcile conflicting objectives, finding acceptable compromises within limitations of cost, time, knowledge; understand the nature of unstructured, open-ended (sometimes known as "wicked") problems
- Design appropriate solutions, using responsible engineering approaches
- Evaluate designs and products
- Understand and be able to apply theories and models that provide a basis for software design,
- Work effectively in interdisciplinary contexts, in particular to bridge the gap between computing technology and the client's technology and to interpret and respect extra-technical constraints
- Work effectively within existing systems, both software artifacts and organizations
- Understand and be able to use current technical solution elements, including both specific tools, components, and frameworks and also abstract elements such as algorithms and architectures
- Program effectively, including code creation, component use, and integration of multiple subsystems
- Apply design and development methods and techniques as appropriate to realize solutions
- Organize and lead development teams, including team-building and negotiation
- Communicate effectively, both verbally and in writing
- Learn new models, techniques, and technologies as they emerge; integrate knowledge from multiple sources to develop solutions to problems; serve as an agent of change for introducing new technology

For programs organized around courses, the columns would correspond to the courses, and the table would be fairly simple, as the curriculum design makes many of the selections about which specific models and techniques satisfy the capability requirements. For a self-paced project-based alternative, the selection of specific models and techniques is driven by individual projects; in this case the mapping will be critical to determining whether each student has satisfied the overall requirements of the program.

4 Pedagogical Principles

Many topics compete for attention in the curriculum, and many activities compete for faculty and student time. The Carnegie Mellon software engineering faculty regards education as an investment from which students should reap benefits for decades. These principles guide our curriculum and course designs. They apply to most formats for presentation, including classroom, independent work with co-located students, and distance education formats.

4.1 University education must provide knowledge of enduring value together with immediate competency.

Universities walk a careful line between education in enduring principles and training in vocational skills. We believe both that a graduate should have certain competencies and that the investment in education should continue to pay off over a long period of time, and accordingly we ask our courses to serve both ends. There is ample evidence that the two are compatible, because many, perhaps most, students learn the principles best by working out examples that apply those principles. The balance shifts somewhat among undergraduate, professional masters, and research programs.

It follows that the objectives for our courses include a combination of "understands" and "can-do" objectives. Further, the "doing" parts of the course must support the "concept" parts. We probably all agree that courses in which students can simply hack their way to apparent success aren't serving their ends.

It also follows that reflection and interpretation are more important than extensive routine drilling, comprehension more important than specific technology-specific skills.

4.2 In engineering, tools and skills cannot replace judgment.

Engineering requires finding cost-effective solutions from among many and diverse alternatives. Methods, tools, processes, skills, heuristics, and other tools and techniques can help to organize the solution search to concentrate on good candidate solutions. These engineering tools and techniques remind you to consider possibilities that you might otherwise ignore. They can help a good (or even adequate) engineer find better solutions more effectively. They are not -- and cannot be -- a substitute for actually understanding the problem and making sound judgments about solutions. An intrinsic characteristic of engineering is the requirement to strike appropriate balances among conflicting goals. So pursuing one tool or technique to the exclusion of all others is only very rarely, if ever, appropriate. Above all, we should teach our students engineering judgment and the commitment to use it; all the specifics support this end.

4.3 The Carnegie Plan provides excellent guidance about engineering education.

Each student should learn not only specific content but also the principles and mindset of the profession, the ability to learn new material independently, and the perspective and judgment to be a responsible adult. In particular, *graduates should be able to assume responsibility for their own continued professional development*. Therefore they should learn not only today's methods and technologies, but also the underlying principles and critical abilities that will allow them to select and master new methods and technologies as they emerge. This idea is captured in the Carnegie Plan, which Carnegie Mellon established half a century ago as part of a major restructuring of engineering education, both at Carnegie Mellon and throughout North America. Not only does the Carnegie Plan carry institutional memory for our university, it is an enduring statement that provides guidance for blending theoretical understanding and practically focused experience into durable skills and the ability to learn new material. Appendix A gives a statement of the Carnegie Plan.

The Carnegie Plan provides guidance for blending theoretical understanding and practically focused experience into durable skills and the ability to learn new material. Reflective practice is entirely consistent with the Carnegie Plan's combination of education grounded in enduring principles, skills in applying these principles and continual improvement, and experience grounded in the real world.

4.4 Hands-on, attentive time on task is critical.

Herbert Simon showed us that the strongest correlation between demonstrable learning and any of the student and instructor activities is with the student's engaged, attentive time practicing with whatever was being learned/taught. He also told us that experts know 50,000 chunks, taking 10 years to acquire this expertise. So a reasonable aspiration for a university course is to get a good start on the 50,000 chunks by providing the student with a conceptual roadmap, lots of hands-on practice in various parts of the space of 50,000 chunks, and the ability to fill in more of the chunks.

It follows that students must assume principal responsibility for their own learning.

4.5 Curriculum design is at heart a resource allocation problem.

The scarce resource is student attention, measured however imperfectly by courses, hours spent, pages of reading, numbers of projects. To provide the greatest value, we must require each course to contribute to both enduring value and immediate competency. This favors content backed by good theory, because good theories compress lots of content into tidy packages; however, this leverage should not be allowed to drive out important (but partially codified) content in favor of pure theory. On the other hand, extensive exercises in a process that is likely to be obsolete in a couple of years can (usually) only be justified to the extent that they support long-term knowledge. It is easy to identify "important" content that more than fills the space in the program -- whether space is measured as class time, student attention, hours of work, or something else. It is much harder to set the priorities that lead to a curriculum that strikes the right balance of coverage and depth.

4.6 Sampling is sufficient; it is not necessary to cover everything.

For students of the high quality that we admit, thorough mastery of a few exemplars coupled with principled overviews should provide a sufficient basis for learning other related material. We should take care, though, to provide sufficient coverage. This is a reasonable decision because curriculum space is a scarce resource and, per the Carnegie Plan, our students can assume responsibility for their own professional development.

4.7 Admissions should be selective, and we should make every effort to help admitted students succeed.

The overall quality of students in a class affects the level at which the class functions, especially in interactive settings. When, as in many of our activities, students work in teams, the quality of each student's experience depends on the quality of the other students. Further, failure of some students affects the whole community of students. It follows that we should attempt to admit students with a good chance of success and commit to making them successful.

4.8 The educational setting should enable students to learn effectively.

Learning depends chiefly on the active engagement of students, who make a substantial investment of time and resources. We should provide opportunities and resources that allow students to do this effectively. While providing sufficient resources, we should at the same time provide a realistic development setting.

Appendix A Carnegie Plan for Engineering Education

Carnegie Mellon does not have a definitive statement of the Carnegie Plan; minor revisions are regularly made for different settings; and the version presented here is a close variant of versions that have appeared in University publications over the years. This is the version of March 1, 1993, from Granger Morgan.

Undergraduate Education at Carnegie Mellon University:

Statement of Mission

A Carnegie Mellon undergraduate education aims to prepare students for life and leadership. In a continually changing world, the most important qualities we can help our students develop are the ability to think independently and critically, the ability to learn, and the ability to change and grow. As future leaders they must have courage to act, be sensitive to the needs and feelings of others, understand and value diversity, and honor the responsibilities that come with specialized knowledge and power.

Carnegie Mellon's undergraduate educational programs are designed to help students acquire:

> *Depth of knowledge* in their chosen areas of specialization and *genuine intellectual breadth* in other fields.
>
> *Creativity and intellectual playfulness*, moving beyond established knowledge and practice to create imaginative ideas and artifacts.
>
> *Skilled thoughtfulness and critical judgment*, which allow them to evaluate new ideas; identify and solve or explore problems; and appreciate a variety of different forms of analysis and thought.
>
> *Skills of independent learning*, which enable them to grow in wisdom and keep abreast of changing knowledge and problems in their profession and the world.
>
> *A considered set of values*, including commitment to personal excellence and intellectual adventure, a concern for the freedoms and dignity of others, and sensitivity to the special professional and social responsibilities that come with advanced learning and positions of leadership.
>
> *The self-confidence and resourcefulness* necessary to take action and get things done.
>
> The *ability to communicate with others* on topics both within and outside their chosen field of specialization.

Most instruction at Carnegie Mellon is focused on fundamentals useful in later learning, rather than on particulars of knowledge and techniques, which may soon become obsolete. Advanced courses provide students with the opportunity to refine their skills by applying and exercising the fundamentals they have acquired in earlier courses and by exploring new analytical and creative directions. We are committed to bring together the traditions of liberal and professional education. In a world which has sometimes placed too little emphasis on "skill," we take pride in educating students who display excellence in application, students who can do useful things with their learning.

Values, including a sensitivity to the feelings, needs, and rights of others, are learned in part through example. To this end, the faculty and staff of Carnegie Mellon work to provide a supportive and caring environment that values and respects intellectual, philosophical, personal, and cultural diversity. The faculty strive to identify and discuss with their students, both in formal classroom settings and in a variety of informal contexts, their responsibilities as professionals, citizens and human beings, and to teach through example.

The educational programs at Carnegie Mellon are designed to help our students become accomplished professionals who are broadly educated, independent, and humane leaders.