The Journal of Systems and Software 125 (2017) 354–364

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Software Systems Engineering programmes a capability approach☆

Carl Landwehr[a], Jochen Ludewig[b], Robert Meersman[c], David Lorge Parnas[d,*], Peretz Shoval[e], Yair Wand[f], David Weiss[g], Elaine Weyuker[h]

[a] *Cyber Security Policy and Research Institute, George Washington University, Washington, DC, USA*
[b] *Institut für Software, Universität Stuttgart, Stuttgart, Germany*
[c] *Institut für Informationssysteme und Computer Medien (IICM), Fakultät für Informatik, TU Graz, Graz, Austria*
[d] *Middle Road Software, Ottawa, Ontario, Canada*
[e] *Ben-Gurion University, Be'er-Sheva, Israel*
[f] *Sauder School of Business, University of British Columbia, Vancouver, BC, Canada*
[g] *Iowa State University, Ames Iowa, USA*
[h] *Mälardalen University, Västerås, Sweden and University of Central Florida, Orlando, FL USA*

ARTICLE INFO

*Article history:*
Received 24 May 2016
Revised 22 November 2016
Accepted 19 December 2016
Available online 23 December 2016

*Keywords:*
Engineering
Education
Software education
Information systems
Software design
Software development
Software documentation

ABSTRACT

This paper discusses third-level educational programmes that are intended to prepare their graduates for a career building systems in which software plays a major role. Such programmes are modelled on traditional Engineering programmes but have been tailored to applications that depend heavily on software. Rather than describe knowledge that should be taught, we describe capabilities that students should acquire in these programmes. The paper begins with some historical observations about the software development field.

## 1. Background

Many universities have created educational programmes to teach the development of software intensive systems. There is a great deal of variation among these programmes and a number of programme names are used. In this paper, we use the term "*Software Systems Engineering" (SSE) to* refer to such programmes. Some types of SSE programmes are discussed in more detail in Section 5 of this paper to illustrate what we mean by Software Systems Engineering.

There have been many efforts to define *bodies of knowledge* for computing disciplines. A list of some of these efforts can be found in The Joint Task Force for Computing Curricula (2005). Some (e.g., Parnas, 1998; Lutz et al., 2014; Ardis et al., 2015) describe programmes that have been developed by individual institutions. Others, (e.g., Computing Curricula, 2005), compare the bodies of knowledge associated with various computing disciplines. In Glass et al. (2004), there is a comparison of computing disciplines based on the research areas associated with each. The SE2004 report (Lethbridge et al., 2006), (and its updated version SE 2014 (Ardis et al., 2015)), propose knowledge that should be taught in undergraduate software oriented programs. They also provide sample courses and curriculum patterns.

This paper takes a complementary approach. Noting that:

- Science programmes present an organized body of knowledge and teach students how to verify and extend that knowledge.
- Engineering programmes present an organized body of knowledge and teach students how to apply that knowledge when developing products.

Instead of discussing the knowledge that would be conveyed to students during their education, this paper focusses on things that a software developer must be able to do while developing and maintaining a product. Like Lethbridge et al. (2006), and Ardis et al. (2015), this paper discusses a set of Engineering programmes in which software development plays a central role; un-

☆ Work on this paper began while the authors served on a committee advising Israel's Council of Higher Education that was chaired by David Parnas. The opinions presented in this paper are the personal opinions of the authors. All authors made substantive contributions to the paper; they are listed in alphabetical order. David Parnas is the corresponding author.

* Corresponding author.
*E-mail address:* parnas@mcmaster.ca (D.L. Parnas).

http://dx.doi.org/10.1016/j.jss.2016.12.016
0164-1212/© 2016 Elsevier Inc. All rights reserved.

like Lethbridge et al. (2006), and Ardis et al. (2015), it does not prescribe courses or curricula. Rather than describing knowledge or research areas, we propose a *body of capabilities*. Many different curricula could help students to acquire the capabilities described in this paper.

Because software is a rapidly changing field, we expect that the associated “body of knowledge” will continue to grow quickly and curricula will need to be revised frequently. In contrast, the capabilities discussed in this paper are fundamental. We base them on observations that were made when the profession of software development was first identified (Brooks, 1995; Buxton and Randell, 1969; Naur and Randell, 1968). They were needed then, they are needed now, and we expect them to be needed in the far future.

We do not believe that the capability approach is a replacement for “Body of Knowledge” or Curriculum proposals. We believe that looking at capabilities as this paper does, provides a perspective that will help institutions to develop, compare, and update curricula.

- Section 2 of this paper reviews discussions that took place when the term “Software Engineering” and similar terms were first introduced.
- Section 3 discusses some capabilities that Software Systems Engineers need.
- Section 4 discusses the role of projects in Software Systems Engineering education.
- Section 5 describes a few of the many distinct disciplines that fall under the rubric of Software Systems Engineering.
- Section 6 discusses how to use this paper when designing or revising a curriculum.
- An appendix provides a more detailed discussion of the most important learning outcomes for Information Systems Engineering.

## 2. Searching for a definition of “Software systems engineering”

In the 1960s, some computer scientists began to use the phrase “Software Engineering”[1] without providing a clear definition. They expressed the hope that software developers would learn to construct their products with the discipline and professionalism associated with professional engineers (Buxton and Randell, 1969; Naur and Randell, 1968).

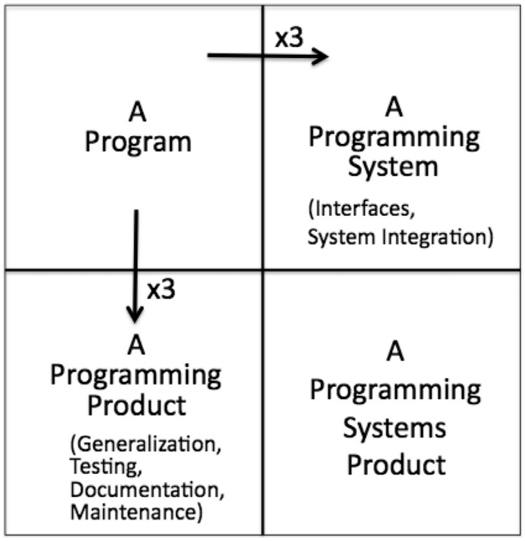
When the term “Software Engineering” was first introduced, many asked, “How is that different from programming?” More recently, when post-secondary “Software Engineering” programmes were introduced, some asked, “How is that different from Computer Science?” Some who asked these questions questioned the need for a new term; others wanted to know what, beyond programming and computer science, would be taught to students of “software engineering”. In the discussion that followed, two simple, but consequential, answers emerged. Although both definitions are old, they have withstood the test of time, are consistent with current usage, and remain relevant today

### 2.1. Brian Randell’s answer

One of the best answers to the question was provided by Prof. Brian Randell, one of the organizers of the first two international Software Engineering conferences and co-author of two frequently referenced reports on those meetings (Buxton and Randell, 1969; Naur and Randell, 1968). In private discussions, he described Software Engineering as “multi-*person* development of multi-*version* programs”. This pithy phrase implies everything that differentiates professional software engineering from programming. Software engineers must be able to work in teams to produce programs that will be used, and revised, by people other than the original developers. Although performing that job requires programming skills, many other capabilities are required as well.

### 2.2. Fred Brooks’ answer

The diagram below appears In Fred Brooks’ classic book, “The Mythical Man-Month” (Brooks, 1995). The vertical dimension denotes “productizing” and the horizontal one “integration”.


x3
A Program
A Programming System
(Interfaces, System Integration)
x3
A Programming Product
(Generalization, Testing, Documentation, Maintenance)
A Programming Systems Product

***Fred Brooks’ explanation of why software engineering is more than programming.***[2]

- By testing, documenting, and preparing a program for use and maintenance by other people, one transforms that program to a “programming product”.
- By integrating a program with other, separately written, programs, one moves from a program to what Brooks called “a programming system”.
- Doing both of these results in a “programming systems product”. Going from a program to a programming systems product results in a massive increase in cost and effort.

Brooks’ formulation, like Randell’s, makes it clear that there is much more than programming skill required of a software engineer. Software engineers must master programming, but they must also be able to integrate separately written programs and “productize” the result.

## 3. What should Software Systems Engineers be prepared to do?

The decision to create Software Systems Engineering programmes that are distinct from “Computer Science” programmes makes these old questions relevant today. We have to ask how Software Systems Engineering programmes should differ from Computer Science Programmes and what criteria should be applied when evaluating them.

[1] Historically, the term “Software Engineering” was used. However, we believe that what is said in this section applies to all Software Systems Engineering disciplines.

[2] Figure redrawn from (Brooks, 1995). The “x3” annotation, denotes a 3-fold increase in effort.