ADDRESSING THE NEEDS OF REAL-TIME EMBEDDED SOFTWARE
A CASE FOR SOFTWARE SYSTEMS ENGINEERING

Robert G. Pettit IV
Flight Software and Embedded Systems Office
The Aerospace Corporation

1 Problem and Motivation

As a society, we have come to rely on software to control a vast array of devices that we depend on in every-day life. Software, rather than hardware, is now the dominant force in the control of embedded systems from hand-held telephones and music players to mission/life-critical devices such as medical devices, and vehicular controls for automotive, rail, and aerospace systems. Yet the disciplines of software engineering and computer science in their current form vastly ignore the intricacies of these embedded systems. Rather, recent trends in academic curricula and industrial practice have focused on fad processes and reliance on what was once considered super-computing power, but is now found in the average desktop system. This has resulted in a loss of foundational instruction and rigor, particularly in undergraduate curriculum.

To address these issues, we must clarify what is covered in the fields of software engineering and computer science, refocus on the critical aspects and reintroduce a rigorous approach. We must also consider the field of software *systems* engineering, in which we educate software engineers on the aspects of systems engineering and the implications of hardware platform choices as well as educating systems engineers on the discipline of software engineering.

This position statement addresses the necessity to clarify definitions for software engineering, computer science, and software systems engineering. We also address key theoretical foundations that must be addressed for mission-critical, real-time, and embedded software systems.

2 Positions Aligned with Semat Goals

In this position statement, we specifically address the Semat Vision Statement goals of Definitions and Theory.

2.1 Definitions

As we move forward with the effort to refound software engineering, the definition of *software engineering* must be clearly stated in a way to both link it to and differentiate it from *computer science*. This could be seen as a macro/micro view, whereas "computer science" focuses on the specific sciences of implementing algorithms in software and "software engineering" focuses on the entire software

development lifecycle and the engineering of large-scale software systems. Thus, software engineering must be firmly grounded in computer science, but requires additional discipline in the areas of software processes, requirements engineering, software architecture, and software design. The later stages of software testing as well as software system maintenance should also be addressed by "software engineering".

We also propose the need to address *software systems engineering*. This concept goes beyond the basic scope of software engineering and addresses the need to develop software engineers who are also grounded in systems engineering concepts. We feel this is a critical aspect to the success of real-time embedded systems, where the software engineer must understand system level implications of the software requirements, architecture, and design. They must also have a basic understanding of the hardware-level engineering of the systems on which the software will be hosted as well as the sensors and actuators with which the software must interact.

2.2 Theory

Modern computer science and software engineering efforts (academic and industrial) are plagued with process and technology fads. Scientific principles of *computer science* have been diluted with the availability of powerful, inexpensive desktop computers. *Software engineering* also tends to fall short of the disciplines of other engineering fields. To move forward, we must firmly ground our computer science and software engineering programs in solid methods, theory, and principles. Towards this goal, the development of *software systems engineers* is critical to the success of large-scale mission-critical software systems, such as those prevalent in modern medical, automotive, and aerospace systems.

Addressing the specific needs for real-time embedded systems, we find the following aspects to be particularly critical to software engineering theory:

> **Architecture**. Well-defined methods, practices, and patterns for the development and assessment of software architectures is critical to large scale software systems and should be a fundamental aspect to the discipline of software systems engineering. For real-time embedded systems, the understanding of architectural characteristics such as performance, predictability, and reliability is particularly important.
>
> **Performance**. Modern computer science and software engineering gives great attention to the satisfaction of functional requirements, but gives relatively little attention to quality of service. For real-time embedded systems that must be highly reactive to their environment, the performance of the software is often of equal importance to the function of the software.
>
> **Predictability**. As software is responsible for the increasing control of embedded devices, we need solid theory in place to guarantee the predictability of the software. From requirements analysis through architecture, design, implementation, and testing, we should employ

methods that allow us to reason about the predictability of the software system and guarantee the desired behavior under all operating conditions.

Efficiency. Before modern desktop computers placed virtually unlimited processing and memory resources in the hands of software engineers, the fundamentals of computer science focused on efficient algorithms and data structures to allow software to be constructed for resource-constrained systems. Most modern embedded software systems must still operate in the context of resource-constrained platforms, yet fewer and fewer software engineers are grounded in the methods and theory needed to effectively design and develop software in this environment.

Model-based methods. Finally, model-based methods show great promise in dealing with the complexity of modern software systems. Software systems engineers should be well-versed in model-based methods at all phases in the software development lifecycle. We should not constrain software systems engineers to a single model-based method, but should ensure that they are educated in a breadth of model-based technologies, including methods from other engineering disciplines such as the use of mathematical-based models. We should also provide model-based methods not only for the construction of software systems, but also for analyzing, simulating, testing, and verifying software systems.

3 Conclusions

The size and complexity of software used to control embedded computing systems will only continue to increase. As we increasingly rely on software to control systems that affect the quality and safety of our lives, it is imperative that we ensure software engineers are properly educated in the foundations of the discipline and in rigorous approaches, backed by proven methods, patterns, and theory. While providing this rigor, though, we must also ensure that research into new methods, patterns, and theories is not impeded, but at the same time, we must define clear criteria for the adoption of any of these advances.