

# Embedded Software—Technologies and Trends

**Christof Ebert**, *Vector*

**Jürgen Salecker**, *Siemens*

**O**ur world and society are shaped and governed by embedded systems. It's difficult to imagine day-to-day life without such systems. There would be no energy ready to use, no running water, and no food supplies; business or transportation would immediately be disrupted; diseases would spread; and security would dramatically decrease. In short, our society would disintegrate rapidly. Examples of embedded systems include pacemakers, implanted biosensors,

RFID tags, cell phones, home appliances, satellites, train control systems, and automotive components.

Embedded systems are microcontroller-based systems built into technical equipment. They're designed for a dedicated purpose and usually don't allow different applications to be loaded and new peripherals to be connected. Communication

with the outside world occurs via sensors and actuators; if applicable, embedded systems provide a human interface for dedicated actions.

The software executed in those systems is called embedded software. It's an integral part of the system itself. Embedded software is defined as a special-purpose software system built

into a larger system. The end user usually doesn't recognize embedded software as software in the traditional way; instead, he or she perceives it as a set of functions that the system provides. Embedded software is a key driver of most industries.

In this introduction to the special issue, we provide a snapshot of the topic of embedded software. We will show similarities to general-purpose IT and highlight embedded systems' peculiarities. This introduction isn't a tutorial on all facets of engineering and maintaining embedded software; rather, it highlights trends and topics worth thinking about. Above all, it relates hands-on experiences from industry projects from which all of us can learn, independent of which type of software and domain we deal with in our day-to-day engineering work. (For other useful sources of embedded software information, see the sidebar.)

## The Challenges of Embedded Software

Too often, when speaking about software, we concentrate on IT systems; we think about general-purpose PCs, big IT systems, and online Internet applications. However, such IT systems incorporate less than 2 percent of the microprocessors produced. Most microprocessors are in systems for cars, mobile communication, washing machines, aircraft, robots, traffic management, cameras, and audio equipment.

This trend toward embedded software in practically all systems is accelerating. The world market for embedded systems is approximately 160 billion euros, involving approximately 3 billion embedded units delivered per year and a compound annual growth of 9 percent.<sup>1-3</sup> Owing to the nature of embedded systems, most of these sales are on the hardware side; however, the relevance of software and services is rapidly increasing.

Cars today have 100 Mbytes of software running, with a complexity growing more quickly than that of IT systems such as those of SAP, Oracle, and Microsoft. The same holds for pacemakers and satellites, starting from a smaller size, of course. To highlight embedded software's ubiquity, Figure 1 shows the size and annual distribution volume of selected embedded software.<sup>4</sup> Although these figures are comparable to those for the world's biggest software packages, such as Microsoft Windows, embedded software's complexity is larger by far owing to real-time and interface constraints that IT, application, and desktop software don't face.

## Embedded Software Resources

Embedded software has many facets. The following list of resources will facilitate an easy start-up for those new to the field, independent of whether they're students or professional engineers who wish to grow their embedded knowledge. It also provides some depth with specific topics we deem relevant for those who actively contribute to embedded software engineering.

### Introductory Books

- J.J. Labrosse et al., *Embedded Software*, Newnes, 2007
- D.E. Simon, *An Embedded Software Primer*, Addison-Wesley Professional, 1999

### Books on Specific Topics

- B. Broekman and E. Notenboom, *Testing Embedded Software*, Addison-Wesley Professional, 2002
- J.E. Cooling, *Software Engineering for Real-Time Systems*, Addison-Wesley, 2002
- D.W. Lewis, *Fundamentals of Embedded Software: Where C and Assembly Meet*, Prentice Hall, 2001
- S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, 2004
- A. Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, Morgan Kaufmann, 2005

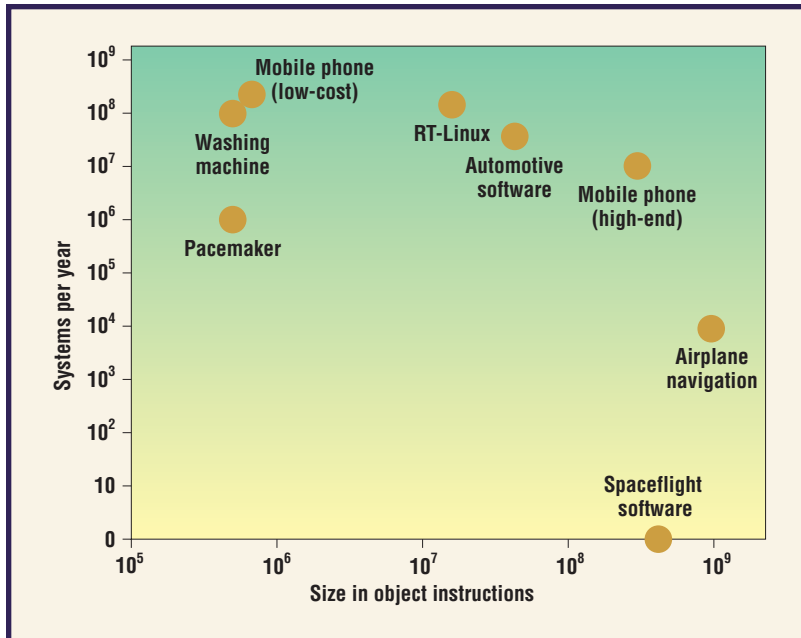
### Associations and Newsgroups

- Hands-on introductions, weekly newsletter, and tons of useful industry practice: [www.embedded.com](http://www.embedded.com)
- Research overview at the Center for Hybrid and Embedded Software Systems: <http://chess.eecs.berkeley.edu>
- Software measurements and benchmarks from the International Software Benchmarking Standards Group: [www.isbsg.org](http://www.isbsg.org)
- Studies on themes such as safety, security, and embedded software roadmaps from the Software Engineering Institute: [www.sei.cmu.edu](http://www.sei.cmu.edu)

You could argue that software is software is software. This is true on the microscopic level. But embedded systems involve many challenges that transcend the ordinary requirements for application and enterprise-style IT systems. These challenges fall into the following six main categories.

### Real Time

Embedded software is by definition part of a larger system, such as a technical process, the human body, or an industrial automation system. These systems all pose external constraints that must be addressed in real time. The embedded system's timing must provide the expected action within a maximum specified time under all circumstances.



**Figure 1. Embedded software size and deployment.<sup>4</sup> Embedded software spans a broad scope of different sizes and numbers of instantiations depending on the system controls.**

### Reliability

Unexpected behavior from an embedded system might seriously damage its environment. Often, such software must operate for decades without service; there's no possibility for upgrades and service patches. End users demand deterministic long-term behaviors from these systems.

### Safety

Much embedded software one way or another impacts people and thus potentially poses a safety hazard. Safety has become a key requirement, specifically as mechanical backups are for cost reasons disappearing in systems such as airplanes, cars, or industrial plants. The entire life cycle thus is governed by standards that demand systematic processes, state-of-the-practice technologies, and continually educated engineers.

### Security

Current embedded systems are highly interconnected to each other and to sensors, actuators, and interfaces. In many embedded systems, security directly means safety. A car contains approximately 30 to 70 embedded systems that communicate with each other across a variety of standardized bus systems. Embedded security is crucial to avoid life-threatening situations. The "Internet of things" will work to our advantage only if we reliably ensure security—much more than in current IT systems.

### Limited Resources

Embedded software is constrained by small memory space and limited data-processing capabilities,

low-cost microcontrollers, stringent regulations with respect to "green" behaviors, and low power consumption. For instance, implanted medical devices must operate for years without battery replacement, and mobile phones must work for hours on subwatt batteries.

### Heterogeneity

Owing to embedded software's long lifetime, it must conform to a wide range of changes in its environment. Processors, sensors, and hardware parts change over time, whereas the software remains almost the same. In addition, the software requires portability, autonomy, flexibility, and adaptability. "Design-for-x" is thus a key paradigm to cope with multiple conflicting requirements.

### Lessons from Embedded Software

We can learn much from embedded software engineering. Too often, software development is still based on the paradigm that software can be rather easily repaired and changed after release. This has led to the bizarre situation where we're used to software being faulty by definition—a situation that for any other product simply isn't imaginable. However, this isn't the case with embedded software; nobody would download a patch for his or her pacemaker from the Internet. Independent of our own software application domains, we should learn from the embedded world in terms of engineering approaches, systematic processes, and the strong focus on quality demands from a variety of operational scenarios and environmental conditions.

### In This Issue

The articles in this issue highlight relevant current technologies and approaches but also emerging trends in embedded system development. "Trends in Embedded Software Engineering," by Peter Liggesmeyer and Mario Trapp, summarizes current advances in embedded software engineering. You could argue that "regular" IT and application software development have long used some of the described techniques. True and not so true, as our Point/Counterpoint authors Les Hatton and Michiel van Genuchten highlight with interesting insights.

One of the most relevant is toward more abstraction and thus being able to better manage complexity throughout the life cycle. This trend began some years ago, when C replaced assembly language. It might sound odd, but performance issues, memory space, and the need for deterministic behaviors still mandate C and assembly language, which are the primary languages in over 80 per-

cent of embedded systems.<sup>3,4</sup> In the near future, embedded system developers will use model-driven development (MDD) to work at different abstraction levels. “UML-Based Model-Driven Development for HSDPA [High-Speed Downlink Packet Access] Design,” by Jesús Martínez and his colleagues, shows how they introduced MDD to embedded software development. The application and development of domain-specific languages is well suited for the embedded domain as well. This approach can easily take into account the embedded domain’s hardware constraints.

The trends to add more and more functions into an embedded system but to consume as little power as possible are contradictory. One possible solution for this dilemma involves multicore microcontrollers, which are entering the embedded domain just about now. This raises two questions: how do you deal with the complexity of developing software for multicore microcontrollers, and how do you best introduce parallel programming? “Embedded Multiprocessor Systems-on-Chip Programming,” by Jean-Yves Mignolet and Roel Wuyts, sheds light on these questions and might help professionals avoid common traps when entering this domain.

Owing to embedded software’s close association with critical environments and often life-threatening risks, it faces high quality requirements. Systematic, thorough, and completely traceable verification and validation is key to good quality. In “Formal Modeling and Verification of Safety-Critical Software,” Junbeom Yoo and his colleagues show how they applied such techniques to safety-critical software in a nuclear reactor protection system. In the Software Technology department article, “Ensuring the Integrity of Embedded Software with Static Code Analysis,” Ben Chelf and Christof Ebert second this need for systematic verification and introduce static code analysis and current related practices and tools for embedded software.

High quality and performance requirements combined with fierce cost pressures demand strong engineering and management processes that continuously improve. In “A More Agile Approach to Embedded Systems Development,” Michael Smith and his colleagues show how to keep good processes sufficiently lean to allow for flexibility and efficiency. Finally, in “Experiences in Improving Flight Software Development Processes,” Ronald Kirk Kandt emphasizes the impact of a higher level of software development maturity on software engineering activities. He underlines that without good processes, we won’t be able to meet embedded software’s many challenging requirements.

## Recommendations

The reason our planet can bear over six billion people is embedded software. Because software is so ubiquitous and embedded in nearly everything we do, we need to stay in control. We must ensure that the systems and their software run as we intend—or better. Staying in control means knowing what’s going on and knowing the limits of what we’re doing. This is relevant for all software engineering but is critical for embedded software. If embedded software has insufficient quality, serious damage will occur, whether injuries, deaths, or catastrophes.

For us to cope with these truly embedded challenges, awareness of embedded software and its impacts must improve across society. Here are a few recommendations that each of us can pursue in our own environments.

### Understand Impacts and Potentials

Policy makers tend to overlook embedded systems’ perspectives and challenges because those systems don’t get much media coverage. We must educate them on the challenges and risks and convince them to allocate funding to improve the scientific basis and its application. A good understanding in due time mitigates the risk of accidents and thus negative impacts on our society.

### Provide Hands-On Education

Education tends to emphasize theoretical concepts and overly small classroom examples. Embedded education must cover computer science as well as electrical engineering and systems engineering. Educators must confront engineering students with real-world challenges and thus ingrain a true engineering spirit of solving complex problems.

### Look to the Entire Life Cycle

Research today is fragmented and divided into technology, application, and process domains. It must provide a consistent, systems-driven framework for systematic modeling, analysis, development, test, and maintenance of embedded software in line with embedded systems engineering.

### Invest into Abstraction

Industry tends to incrementally evolve existing proprietary approaches bottom-up. So, complexity and cost grow, whereas quality at best stagnates over time. Standardization, shareable models, open interfaces, and top-down verifiability are necessary to better control complexity and thus ensure high quality from concept to maintenance.

**Too often, software development is based on the paradigm that software can be easily repaired and changed after release.**

## About the Authors



**Christof Ebert** is a partner and managing director at Vector. His research and consulting focus on product development and productivity improvement. Ebert has a PhD with honors in electrical engineering from the University of Stuttgart. He's a senior member of the IEEE and the editor of *IEEE Software's* Software Technology department. Contact him at [christof.ebert@vector-consulting.de](mailto:christof.ebert@vector-consulting.de).

**Jürgen Salecker** is the competence field manager for embedded systems at Siemens Corporate Technology. His research and consulting focus is on hardware and software codesign and concepts for cross-impact analysis of software architecture for embedded system development. Salecker has a graduate engineer degree in information technology from Fachhochschule Hannover and an MBA from Heriot-Watt University. Contact him at [juergen.salecker@siemens.com](mailto:juergen.salecker@siemens.com).



## Communicate and Collaborate

Engineers too often use their own language and tools to communicate in closed circles. Effective embedded software engineering demands that we communicate and collaborate across disciplines, domains, and cultures. We need to understand both software engineering and application domains. Also, we must stand up and speak when risks exist or embedded software is misused.

**E** mbedded software not only increases the variability, configurability, extensibility, and changeability of everyday products but also allows for a greater variety of functions. In the future, embedded software will be in everything—your automated home, your intelligent automobile, communication infrastructures, medical instruments and implants, and ubiquitous control systems. We'll see new energy-related technologies that increase the efficiency of electrical current transmission and provide immediate, effective ways to address energy and climate demands. We'll see embedded systems no longer being defined by the computing hardware they use. Rather, they'll be designed to do any function to achieve multiple and changing objectives, whether on a microcontroller, a microprocessor, a signal processor, a biological assembly, or any other programmable logic device.

The more quality of life we desire, the higher living standards we want to establish across the planet, and the more we demand security and safety, the more we need embedded software. Our task is to evolve embedded software engineering to master these grand challenges. ☺

## References

1. *Information Technology Outlook 2006*, Organisation for Economic Co-operation and Development, 2006; [www.oecd.org/sti/ito](http://www.oecd.org/sti/ito).
2. *Proc. 2nd OECD Conf. Innovation in the Software Sector*, Organisation for Economic Co-operation and Development, 2008; [www.oecd.org/sti/innovation/software/conference](http://www.oecd.org/sti/innovation/software/conference).
3. *Embedded Systems Study 2008*, Bitkom, 2008; [www.bitkom.org/files/documents/Studie\\_BITKOM\\_Embedded-Systeme\\_11\\_11\\_2008.pdf](http://www.bitkom.org/files/documents/Studie_BITKOM_Embedded-Systeme_11_11_2008.pdf) (in German).
4. C. Ebert and T.J. Jones, "Embedded Software: Facts, Figures, and Future," *Computer*, Apr. 2009, pp. 42–52.

For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/csdl](http://www.computer.org/csdl).

## CALL FOR ARTICLES

# Agility and Architecture— Oil and Water?

**PUBLICATION:** March/April 2010  
**SUBMISSION DEADLINE:** 17 August 2009

**A**gile development approaches have had significant impact on industrial software development practices. However, despite becoming widely popular, there is an increasing perplexity about the role and importance of a system's software architecture in agile approaches. Advocates of architecture's vital role in achieving quality goals of large-scale software-intensive systems are skeptics of the scalability of any development approach that does not pay sufficient attention to these architectural aspects, especially in domains like automotive, telecommunication, finance, and medical devices. But agile proponents usually perceive the upfront design and evaluation of architecture as being of little value to a system's customers. There is a growing interest in separating facts from myths about the necessity, importance, advantages, and disadvantages of coexistence of agile and architectural approaches.

**FOR MORE INFORMATION ABOUT THE FOCUS,  
CONTACT THE GUEST EDITORS:**

- Pekka Abrahamsson, VTT, Finland, [pekka.abrahamsson@vtt.fi](mailto:pekka.abrahamsson@vtt.fi)
- M. Ali Babar, Lero, [malibaba@lero.ie](mailto:malibaba@lero.ie)
- Philippe Kruchten, Univ. of British Columbia, [pbk@ece.ubc.ca](mailto:pbk@ece.ubc.ca)

**FOR SUBMISSION DETAILS AND GENERAL AUTHOR GUIDELINES**  
[www.computer.org/software/author.htm](http://www.computer.org/software/author.htm)  
or [software@computer.org](mailto:software@computer.org)

**IEEE  
Software**