GUEST EDITORS’ INTRODUCTION: EVOLVING CRITICAL SYSTEMS

Lorcan Coyle, Mike Hinchey, and Bashar Nuseibeh, Lero—the Irish Software Engineering Research Centre
José Luiz Fiadeiro, University of Leicester

This special issue brings together key software engineering researchers and practitioners who both influence their organizations and evaluate the emerging practice of developing these new systems.

We believe that the software engineering community must concentrate efforts on the techniques, methodologies, and tools needed to design, implement, and maintain critical software systems that evolve successfully. This special issue summarizes many of the topics discussed and embodies what we believe to be some of the most important research challenges for evolving critical software systems—without incurring prohibitive costs.

Several widespread changes in software engineering highlight the importance of evolving critical systems (ECS). We’ve identified the following five game changers:

- **Software ubiquity.** More software is being deployed in more consumer devices, which means failures are more likely to affect ordinary people.
- **Software criticality.** As software embeds itself deeper into the fabric of society, single software failures have greater potential to affect more people. This increases the potential for software to be considered critical even when it isn’t complex.
- **People-in-the-loop.** As software is deployed to control systems in which human actors participate, the issue of human interactions with software becomes more important.
- **Entanglement.** Software dependencies have become more complex, and much real-world software is entangled with software developed by third-party providers.
- **Increased evolution tempo.** The tempo of evolution will continue increasing as users expect more from software. The software market is often unforgiving when even small changes can’t be done cheaply and quickly.

Taken together, these changes characterize evolving critical systems and frame the research agenda in this emerging area (www.lero.ie/ecs/whitepaper).

**ECS MANIFESTO**

ECS research challenges add to the broad software engineering research agenda, with more of a focus on predictability, quality, and the ability to change. Table 1 characterizes the criticality of software engineering research challenges for ECS.
The fundamental research question underlying ECS research is this: how do we design, implement, and maintain critical software systems that are highly reliable while retaining this reliability as they evolve, without incurring prohibitive costs? Several demands must be met before ECS’s ideals can be realized.

The changing development environment, for example, proves that we must maintain the quality of critical software despite constant change in its teams, processes, methods, and toolkits. Likewise, we must improve our existing software design methodologies so that they facilitate the support and maintenance of ECS—how can we use agile development methodologies to evolve critical software?

We must also specify what we want to achieve during an evolution cycle and confirm that we’ve achieved the intended result (verification) and only the result intended (validation). We must thus elicit and represent requirements for change such that we ensure the changes take place correctly. Furthermore, we must develop techniques for better estimating specific evolution activities a priori, only attempting software change when we know for certain that evolution will be successful and that benefits will outweigh costs. For example, some systems shouldn’t evolve at all because the cost and risk of performing evolution successfully will exceed the system’s value by orders of magnitude. To prevent cost and time overruns, we must work toward developing objective criteria to help us decide whether a given system is in this class.

All these requirements demand strategies to make model-driven, automatic evolution a better alternative to manual change. In cases where it isn’t appropriate to mechanize change, we must develop heuristics for determining when such an approach is viable. When humans must perform the change, we need to develop support tools that make this a less risky enterprise.

We also need improved tools for traceability that keep various software artifacts—such as documentation and source code—in sync throughout the evolution cycle. Where regulatory compliance is required, these tools must ensure that evolution results in compliant software.

Finally, during runtime evolution, we must ensure that developers adhere to runtime policies. We do so by developing techniques that can monitor and model changing requirements in dynamic environments, especially autonomic and adaptive software environments. We must also develop strategies for evolution that tolerate uncertainty in the operational environment, which changes deterministically, nondeterministically, or stochastically. We must then ensure that software never evolves into a state of unstable behavior.

Given the tensions between the need for software change and the danger implicit in changing critical software, the most pressing question for practitioners is which processes, techniques, and tools can most cost-effectively evolve critical systems. We believe a concerted focus from the research community to overcome these challenges will be needed for ECS to have an impact on software engineering.

IN THIS ISSUE

This special issue contains contributions from leading participants in the field. In “Evolving Embedded Systems,” Gabor Karsai and coauthors discuss the importance of considering the interplay among requirements, processes, deployments, and tests’ evolution when developing embedded systems. They also address challenges relating to the different evolutionary time scales—whether the system is evolved at design time, load time, or runtime.

In addition, the authors examine the importance of process and system co-evolution, especially with regard to embedded systems, discussing how verification can be used to check the correctness of load-time evolution. They conclude by addressing the difficulties inherent in testing software that evolves at load time or runtime, while also relating it to the use of online testing, built-in testing for load-time evolution, and the need to evolve the tests themselves, especially for runtime evolution.

In “Evolving Software Architecture Descriptions of Critical Systems,” Tom Mens and coauthors discuss the use of architectural descriptions to describe software-intensive systems and how they can be used to handle increasing complexity to mitigate the risks incurred in constructing and evolving these systems. They also assess the use of model-transformation approaches to evolve models of software architectures and the co-evolution of architecture descriptions, software design, and implementation. They call for architectural change to be considered a first-class construct that ensures
Consider the problem of assessing criticality in automotive systems. One important aspect of most such systems is functional safety. This is addressed in ISO 26262, a forthcoming standard for functional safety of electrical and electronic (E/E) systems in road vehicles. Currently, a draft international standard (DIS), it's expected to be approved by the International Organization for Standardization by mid-2011. To reduce unpredictable product liability risks, all road vehicles brought to market after publication of the final standard must conform to ISO 26262. This means that E/E system developers must consider all development process and product properties requirements mandated by the standard from the beginning.

To specify the criticality of E/E system malfunctions, ISO 26262 defines the Automotive Safety Integrity Level (ASIL), which ranges from A (lowest) to D (highest); a QM value indicates that a malfunction isn’t safety-related. As Figure A shows, developers must estimate three parameters to determine ASIL:

- **Exposure (E).** This factor defines the probability of a system being in an operational situation that can be hazardous if coincident with a failure mode; rated on a scale from E0 (incredible) to E4 (high probability).
- **Controllability (C).** This factor assesses the potential of avoiding specific harm or damage through the timely reactions of the persons involved; rated on a scale from C0 (controllable in general) to C3 (difficult to control or uncontrollable).
- **Severity (S).** This parameter measures the potential extent of harm to an individual in a specific situation; rated on a scale from S0 (no injuries) to S3 (life-threatening or fatal injuries).

Various interpretations of the categories' meaning have been defined for E, C, and S. To avoid different ASIL classifications for the same malfunction, the automotive industry must establish common criteria.

An appropriate system design makes reducing ASIL classifications for some elements, a process known as ASIL decomposition. In this case, the elements' independence after decomposition must be assured—“lower-quality” parts mustn’t affect the operation of “critical” parts that assure safety.

For example, software malfunctions within the electronic actuator pedal (EGAS) system may lead to the ASIL B, as the top of Figure B shows. This is decomposed into QM for the function level and B for the monitoring level (as the bottom of the figure shows), which assures safety by switching off the power stages if the level malfunctions.

In the case of E/E systems, automotive engineers are thus guided by a well-established set of different safety-integrity levels rather than a general notion of criticality. However, technical challenges arise from the problem of coping with different ASILs within one application. And this situation will probably occur more frequently given the growing trend to build functionally cooperating networks of originally separate systems.

More challenges arise if safety requirements conflict with other types of criticality, such as system reliability and availability (quality of service). For example, a fail-safe solution like switching off the power stages in the EGAS might lead to customer dissatisfaction if it occurs too often.

Dieter Lienert is the head of a group for the engineering of software-intensive systems at Robert Bosch GmbH. Contact him at dieter.lienert@de.bosch.com.

Stefan Kriso is a senior project manager at Robert Bosch GmbH. Contact him at stefan.kriso@de.bosch.com.

**Figure A.** Automotive Safety Integrity Level determination. Electrical/electronic system developers must estimate three parameters—exposure (E), controllability (C), and severity (S)—to determine a malfunction’s AWIL (Source: ISO/DIS 26262-3, Table 4.)

**Figure B.** Example of ASIL decomposition for the electronic actuator pedal (EGAS) system. Top: Software malfunctions in the microcontroller (µC) are classified as ASIL B. Bottom: the monitoring level is classified as ASIL B, while the function level can be classified as QM if freedom from interference between the levels can be justified.
Boutheina Chetali, Security Labs—Gemalto

Smart-card certification’s evolution can’t be tackled as a whole, but only as an update evolving into a new service or capability. The software’s update focuses on the context of product surveillance and maintenance, two processes clearly defined by the certification scheme. To update a certified product and maintain the certificate, the developer must provide evidence that this update has no security impact.

Open smart cards

Requirements such as these could burden a business model, where software updates must be frequent to keep pace with the rapid evolution of specifications. However, the most interesting evolution deals with new services first, then with new code applications. For that, the trend is the certification of “open” smart cards.

Smart cards that have been certified as “open” could be used to load any kind of applications through binary code when in the field, while keeping its certificate intact. An open card has no applications onboard and essentially becomes an “operating system”—such as a Java Card platform.

This sophisticated application manages card resources securely by, for example, loading, installing, deleting, and delivering updates. The openness relies on the loading mechanism and its isolation properties, and these security mechanisms have been evaluated during certification to provide the necessary guarantees that the product can load any code and that two different applications can remain secure and protected from each other.

Certification rules

To maintain stability, certification includes rules called hypotheses that applications must respect before being loaded onto the card. These rules rely on blocking attack paths. One well-known rule advises that code be checked by a bytecode verifier, which means the application must be bytecode-verified before loading, if the onboard verification is not a feature of the card.

The rules set represents the card issuer policy, but if the applications originate elsewhere, in different market sectors that don’t have the same historic level of security requirements, such as banking and mobile communication, a common agreement on policy rules could present a complex task. Therefore, it seems obvious that if the product is protected against any kind of application that could be loaded on the card, the underlying software must include a large set of protective countermeasures.

For a constrained-resources device such as a smart card, adding ever more software countermeasures or onboard verification leads to performance issues during execution and also consumes card-memory space. Both are crucial for the end user’s satisfaction: the first is response time to the request, the second the number of applications that can load onto the product.

Essentially, then, the challenge resides on performance rates, and a trade-off must be made between security and speed. So the notion of evolution is not the same for each sector: the lifetime of a banking card differs from that of a SIM card. This is why taking certification requirements and evolution requirements for cards into account when hosting applications from different market sectors poses a daunting challenge for the smart-card industry.

Boutheina Chetali manages the Formal Methods Group at Security Labs—Gemalto. Contact her at boutheina.chetali@gemalto.com.

In “Evolution in Relation to Risk and Trust Management,” Mass Soldal Lund and coauthors argue that risk and trust management methodologies in general, and assessment in particular, aren’t well-equipped to deal with evolution. They go on to explore risk assessment from the perspectives of maintenance, in which old risks must be updated to take into account new risks introduced by changes before and after, in which the assessor must be aware of current risks, future risks, and risks introduced by the change process itself, and continuous evolution, which involves identifying and assessing how risks evolve. The authors also explore using each of these two perspectives from the specific viewpoint of the risk that trust relations impose on a system. They assert that the evolution of trust is much more challenging, given the highly dynamic nature of trust relations.

Finally, in “Why Critical Systems Need Help to Evolve,” Bernie Cohen and Philip Boxer analyze the difficulties in evolving complex sociotechnical systems by using the provision of orthotic services in the UK as an exemplar. They use three cuts to address the risks of ECS. The Heisenberg cut presents a mismatch between the model that defines a system and the reality of its interaction with stakeholders; the Heisenberg cut presents a mismatch between what behaviors can and can’t be predicted by its users, independently of their use of it; and the Endo-exo cut reflects the difference between what can and can’t be directly known by clients about their own needs. By enabling the members of and stakeholders in a sociotechnical system to analyze and project the experience of their own participation, the authors gain an understanding of how the orthotic service makes the three cuts and identify changes that could improve it. Significantly, these changes were ultimately rejected by the UK’s National Health Service. The authors suggest that this stemmed from a failure to understand the ecosystem in which the service was embedded and the wider implications of these changes beyond the orthotics services.

In addition to this introduction, we’ve included three shorter practitioner contributions from Robert Bosch GmbH, Security Labs—Gemalto, and the Directorate General for Informatics in the European Commission.
Citizens and businesses are empowered by eGovernment services designed around users’ needs and developed in collaboration with third parties, as well as increased access to public information, strengthened transparency and effective means for involvement of stakeholders in the policy process; • “Mobility in the Single Market is reinforced by seamless eGovernment services for the setting up and running of a business and for studying, working, residing, and retiring anywhere in the European Union”; and • “Efficiency and effectiveness is enabled by a constant effort to use eGovernment to reduce the administrative burden, improve organizational processes and promote a sustainable low-carbon economy.”

Aligned with this declaration, the European Commission is preparing to modernize its portfolio of mission-critical information systems to deliver and operate smart e-government services that are innovative and built from a user-centric viewpoint, enabling their participation in the underlying processes (empowerment); and streamline administrative processes in a learning organization to improve effectiveness, efficiency, and transparency, and to share and value intellectual assets through appropriate knowledge-management approaches. This new generation of critical systems will be built on three principles: harmonization and convergence of business processes, reusability and interoperability of information systems or systems components, and sharing services at the infrastructure level. These three layers of the EC’s IT Enterprise Architecture Framework will be enabled by an organization possessing the necessary IT governance arrangements and project management methodologies supporting top-caliber staff whose skills and knowledge will be continuously improved through collaboration in multidisciplinary teams.

The new mission-critical systems will be developed under the auspices of the EU’s Interoperability Solutions for European Public Administrations (ISA) program, which came into being on 1 January 2010. ISA’s focus is on back-office solutions to support the interaction between European public administrations and the implementation of EU policies and activities. It underlines the key role that standards and interoperability at all levels—legal, organizational, semantic, and technical—will play in ensuring these new systems contribute to European integration. Developments will conform to the ISA’s European Interoperability Strategy, will be based on its European Interoperability Framework, and will respect the related architectural guidelines. The program thus defines the architecture for the next generation of evolving critical information systems upon which the EU will rely to implement the Europe 2020 vision articulated by EC President José Manuel Barroso.

E-government is now mainstream; in the EU and elsewhere it will be the catalyst in transforming public administrations over the next decade. The challenge for today’s public-sector CIOs is to build evolving critical information systems that offer more online public services, streamline administrative procedures and cut red tape, and implement innovative service delivery mechanisms. The emerging evolving critical systems research domain will contribute to meeting this challenge.

Declan Deasy is Director of Information Systems at the European Commission in Brussels. Contact him at declan.deasy@ec.europa.eu.

Franck Noel is Deputy Head of Unit in the European Commission in Brussels. Contact him at franck.noel@ec.europa.eu.
MAY 2010

Mike Hinchey is scientific director of Lero and a professor of software engineering at the University of Limerick. His research interests include self-managing software and formal methods for system development. Hinchey received a PhD in computer science from the University of Cambridge. He is a senior member of IEEE and currently chairs the IFIP Technical Assembly. Contact him at mike.hinchey@lero.ie.

Bashar Nuseibeh is a professor of software engineering and chief scientist for Lero. He is also a professor of computing at The Open University, UK, and a visiting professor at Imperial College London and the National Institute of Informatics, Japan. His research interests include requirements engineering and design, security and privacy, and technology transfer. Nuseibeh holds a PhD in software engineering from Imperial College London and is editor in chief of IEEE Trans. Software Eng. and editor emeritus of the Automated Software Eng. J. He is also a fellow of the British Computer Society (BCS) and the Institution of Engineering and Technology, and an Automated Software Engineering Fellow. Contact him at bashar.nuseibeh@lero.ie.

José Luiz Fiadeiro is a professor of software science and engineering in the Department of Computer Science at the University of Leicester. His research interests lie in the mathematical foundations of software system modeling, including software architecture, coordination models and languages, parallel and distributed system design, and service-oriented computing. Fiadeiro received a PhD in mathematics from the Technical University of Lisbon. He is a fellow of the BCS. Contact him at jose@mcs.le.ac.uk.

Dieter Lienert and Stefan Kriso describe the emerging functional safety standard for electrical and electronic automotive systems (ISO 26262) and discuss the challenges in assessing criticality in automotive systems.

Boutheina Chetali points out that the smart-card industry faces a significant challenge in managing both certification and evolution requirements.

Finally, Franck Noël and Declan Deasy discuss how European integration within the EU has, in many cases, been enabled and accelerated by the development and evolution of their technical infrastructure.

Acknowledgments
This special issue resulted from a workshop held in Schloss Dagstuhl, Germany, in December 2009. The gathering brought together key software engineering researchers and practitioners in positions to influence their organizations’ research direction and discuss the emerging theme of ECS.

This work was supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre (www.lero.ie). We thank Schloss Dagstuhl for hosting the perspectives workshop, and all the workshop participants whose discussions led to this special issue.

Lorcan Coyle is a research fellow at Lero in the University of Limerick. His research interests include autonomic computing, context awareness, pervasive computing, and machine learning. Coyle received a PhD in computer science from Trinity College Dublin. He is a member of the Institution of Engineers of Ireland. Contact him at lorcan.coyle@lero.ie.


Where Are We Headed?

Rich Internet applications are a heterogeneous family of solutions, characterized by a common goal of adding new capabilities to the conventional hypertext-based Web. RIAs combine the Web’s lightweight distribution architecture with desktop applications’ interface interactivity and computation power, and the resulting combination improves all the elements of a Web application.

Read the latest issue of IEEE Internet Computing now!

www.computer.org/internet