

# Software Engineering: A Roadmap

**Anthony Finkelstein**  
Department of Computer Science  
University College London  
Gower St.  
London WC1E 6BT, UK  
+44 020 7380 7293  
a.finkelstein@cs.ucl.ac.uk

**Jeff Kramer**  
Department of Computing  
Imperial College  
180 Queens Gate  
London SW7 2BZ  
+44 020 7594 8271  
jk@doc.ic.ac.uk

## ABSTRACT

This paper provides a roadmap for software engineering. It identifies the principal research challenges being faced by the discipline and brings together the threads derived from the key research specialisations within software engineering. The paper draws heavily on the roadmaps covering specific areas of software engineering research collected in this volume.

## Keywords

software engineering, research, discipline, future, strategy

## 1 INTRODUCTION

This paper attempts to construct a roadmap for software engineering research. It seeks to identify the principal research challenges being faced by the discipline and to bring together the threads derived from the key research specialisations within software engineering. In doing so it draws heavily on the roadmaps covering specific areas of software engineering research collected in this volume.

Definitions are notoriously difficult but for working purposes and those of this volume - software engineering is the branch of systems engineering concerned with the development of large and complex software intensive systems. It focuses on: the real-world goals for, services provided by, and constraints on such systems; the precise specification of system structure and behaviour, and the implementation of these specifications; the activities required in order to develop an assurance that the specifications and real-world goals have been met; the evolution of such systems over time and across system families. It is also concerned with the processes, methods and tools for the development of software intensive systems in an economic and timely manner.

We do not aim to provide a summary of the overall state-of-the-art in software engineering. The reader interested in a general introduction should refer to the many excellent textbooks that are available. Best known, and a good starting point, are [5], [3] and [2], all of which are reasonably up to date. [1] affords a good start to the broader literature. The research literature on software engineering is readily available. IEEE Transactions on Software Engineering (IEEE-TSE) and ACM Transactions on Software Engineering and Methodology (ACM-TOSEM) are the principal archival journals. There are a large number of specialised journals including for example Automated Software Engineering (ASE), Requirements Engineering Journal (REJ), Software Process Journal (SPJ). IEEE Software plays an important role in bridging between the ‘pure’ research literature and practitioner-oriented articles. The International Conference on Software Engineering (ICSE) is the flagship conference of the software engineering community; papers in this conference are generally of a high standard and the proceedings reflect a broad view of research across software engineering. The European Software Engineering Conferences (ESEC) and the Foundations of Software Engineering Conferences (FSE), which are held jointly in alternate years, are similar to ICSE though have tended historically to have a slightly more ‘theoretical’ orientation. There are a large number of specialised conferences and workshops ranging from established meetings such as the International Workshop on Software Specification and Design, International Software Architecture Workshop, International Symposium on Software Testing and Analysis to the ‘hot-topic’ workshops held in conjunction with ICSE. There are excellent resources on the web (links are provided on the web site associated with this volume). General software engineering announcements are distributed through the community-wide “seworld” mailing list.

The structure of the paper is approximately as follows. In sections 2 and 3 we discuss the changing context of software system development and the changing orientation of software engineering research. In section 4 we make some broad observations about the evolution of the discipline. Then, in section 5, we analyse the key research



challenges and show how these challenges are reflected in the specialised roadmaps that comprise the volume. We finish by drawing some broad and necessarily speculative and personal conclusions about the future of software engineering.

## **2 CONTEXT**

The context of software system development is changing. Systems are rarely developed from scratch; most system development involves extension of preexisting systems and integration with 'legacy' infrastructure. These systems are embedded in complex, highly dynamic, decentralised organisations; they are required to support business and industrial processes which are continually reorganised to meet changing consumer demands. The services that such a system provides must, for the life of the system, satisfy the requirements of a diverse and shifting group of stakeholders. There is a shift towards client and user centered approaches to development and an accompanying shift from a concern with whether a system will work towards how well it will work. Overall, fewer 'bespoke' software systems are being constructed. Instead, generic components are built to be sold into markets. Components are selected and purchased 'off the shelf' with development effort being refocused on configuration and interoperability.

The resulting systems are composed from autonomous, locally managed, heterogeneous components, which are required to cooperate to provide complex services. They are, in general, distributed and have significant non-functional constraints on their operation. There are a wide range of new, and constantly changing business models relating to the provision of software and software-mediated services resulting from internet and e-commerce technology.

The overall setting is characterised by on the one hand an increasing business dependence on reliability of software infrastructure and on the other hand rapid change and reconfiguration of business services necessitating rapid software development and frequent change to that software infrastructure.

## **3 ORIENTATION**

Reflecting an increased disciplinary maturity the 'orientation' of software engineering research has changed. Software engineering research is being more carefully targeted towards "real" industrial problems. This entails thorough problem analysis. Increasingly software engineering research is aimed towards engineering solutions that are lightweight, in the sense that they make minimal assumptions about the engineering environment in which they are deployed. This is often related to the need for solutions to be simple enough that they can be adopted in practice. Much, but by no means all, software engineering research to date has been characterised by overly complex, heavyweight, solutions that have,

understandably, encountered significant resistance from practitioners.

It has taken a long time for researchers to realise that we cannot expect industry to make very large big-bang changes to processes, methods and tools, at any rate without substantial evidence of the value derivable from those changes. This, accompanied again by the increased disciplinary maturity, has led to a higher "validity" barrier which research contributions must cross. It is readily observable that research that proposes new frameworks, methods and processes are not accepted without positive evidence that they are of use rather than simply airy and unfounded speculation.

Particular attention is being paid to the issue of scalability. It has, in the past, proved all too easy for researchers to ignore or make light of the problems that the sheer scale of industrial software systems development gives rise to. Problems that appear simple in paper-and-pencil exercises in the laboratory are often far from simple when dealing with very large amounts of data. The internet too has implications for scalability. In an open internet setting there may be millions of potential users of a software service.

While research methodology remains a potent issue there is some evidence of an increasing acceptance of methodological diversity. Case studies, qualitative studies, experiments, proof and mathematical analysis are being combined judiciously to make a case for research contributions. Research is expected to, and increasingly does, build on the work of others. Using existing standards and building on, rather than in parallel to, proven research contributions characterises the best research. There is however less tolerance for reinventing the wheel.

## **4 DISCIPLINE**

In defining software engineering we described it as a "branch of systems engineering". Unfortunately systems engineering, despite a long history, is less mature than software engineering! Software engineering research is increasingly aware of the interplay between systems context and software and there are attempts to take into account the co-development of hardware and organisational systems with software. We anticipate a further shift in orientation among software engineers towards a broader systems engineering view with software engineers taking a lead in the creation of a truly integrated systems engineering discipline

A traditional theme in software engineering discourse has been "why can't we build software like other engineers build bridges" [or similar traditional engineering product]. This refrain has led to some productive thinking and has forced software engineers to think hard about achievements, aspirations and the status of our claim to be 'engineers'. It has however led us to apply inappropriate analogies with, for example, mechanical and other

artefacts, which have fundamentally different characteristics from software. It has also given rise to a perceived sense of inferiority, unjustified by the significant research accomplishments of software engineering. We believe that this self-deprecation has in turn had an adverse effect on the funding and the status of software engineering in computer science. There is some evidence that this traditional theme is finally becoming less frequently heard and that a more robust self-image with respect to other disciplines is emerging. We need to recognize, claim and publicize our many successes.

Software engineering has, to a large extent, historically defined itself in terms of testing and debugging. Most software engineering textbooks start with a discussion of early error detection and removal. Software engineers seem to enjoy talking about errors. Failures such as that of the London Ambulance Service Computer Aided Despatch system and the Ariane 5 receive much attention. While such a focus on failures can be instructive - even construction engineers study bridge failures as a means of learning lessons - it can also be said to lead to a negative orientation in which the absence of bugs rather than the positive presence of quality, however defined, is the most important goal. The changing context of software development in which there is a pressing need to roll out a service rapidly and to change it to meet new business demands, forces a change in this outlook towards a more positive ‘holistic’ view of the role of software engineering in delivering satisfaction to users.

## 5 RESEARCH CHALLENGES

A wholly mature discipline is one that is able to identify “dead problems”. Dead problems are those to which effort need no longer be devoted because they have been solved. Computer science is slowly building up a list of such dead problems reinforcing its claim as a mature discipline. By contrast software engineering is still struggling to identify its own dead problems. Any list we constructed of such problems would almost certainly be more controversial than the attempt, which follows, to pick out key research challenges across software engineering. These overall software engineering research challenges are not comprehensive and the determined reader can infer our view of dead problems from it. The questions associated with each challenge are exemplars and individual researchers may derive much more specific research questions. Clearly these challenges are not orthogonal and there are complex relationships binding them together. Many of the most interesting research programmes look at these relationships.

- *Compositionality* - When we compose components what effect does this have on the properties of those components? Can we reason about, and engineer for, the emergent properties of systems composed from components whose behaviour we understand?

- *Change* - How can we cope with requirements change? How can we build systems that are more resilient or adaptive under change? How can we predict the effects of such changes?
- *Non-functional Properties* - How can we model non-functional properties of systems and reason about them, particularly in the early stages of system development? How can these models be integrated with other models used in system development?
- *Service-view* - How can we shift from a traditional product-oriented view of software system development towards a service view? What effects do new modes of software service delivery have on software development?
- *Perspectives* - How can we devise and support new structuring schemes and methods for separating concerns?
- *Non-classical life cycles* - How can we adapt conventional software engineering methods and techniques to work in evolutionary, rapid, extreme and other non-classical styles of software development?
- *Architecture* - How can we represent, reason about and manage the evolution of software architectures? How can we relate software architecture to other parts of the software development process?
- *Configurability* - How can we allow users, in the broadest sense, to use components in order to configure, customize and evolve systems?
- *Domain specificity* - How can we exploit the properties of particular domains (telecommunications, transport) to make any of these challenges easier to address?

The detailed tables that follow look at these challenges set against the detailed challenges or pointers identified by the authors of the individual roadmaps. The tables also serve as a useful quick reference to the volume. A grayed box indicates a clear and straightforwardly identifiable relationship between a ‘big’ challenge and a more fine-grained one. Also included in the tables are links or cross-references from one set of fine-grain challenges to another.

While the challenges that we have been identified do not subsume all of the issues raised by the roadmaps they appear to subtly impact many of them. Very large proportions of the fine-grained challenges relate to the big challenges. Most of those which do not, relate different areas of research activity as indicated by links. A small proportion of the fine-grained challenges relates to neither big challenges nor other parts of the research agenda. These are, for the most part, technical problems blocking advances in particular areas.

## 6 CONCLUSIONS

This paper takes a positive view of current progress and future challenges in software engineering. We believe the discipline has delivered and is well set to continue to deliver both practical support to software developers and the theoretical frameworks which will allow that practical support to be adopted, used and extended with confidence. It is well known that software engineering innovations take a surprisingly long time to percolate through to every day use [4]. Despite this lag current software engineering practice is being radically reshaped by object-oriented design methods, CASE tools with powerful code generation, testing and analysis environments, development patterns, incremental delivery based life-cycles, component models and document management environments. All of these have been formed through software engineering research.

A vision of the future of software engineering suggests a setting in which developers are able to wire together distributed components and services (heterogeneous and sourced over the net) having established at an early stage, through rigorous (yet easy-to-use) formal analysis that the particular configuration will meet the requirements (both functional and non-functional). The overall process in which this takes place will have seamless tool support that extends through to change over the system or service life. Each facet of the resulting system or service will be traceable to (and from) the originating stakeholders who will be involved throughout the process.

This vision is in fact an old one! The difference is that making it a reality is now within our grasp. We know what we have to do. What makes our field even more exciting is that, in addition to the steady progress towards our vision, there are also the discontinuities, such as was introduced in the last ten years by the web. The impact of such major innovations cannot be predicted but they certainly offer wonderful new opportunities and challenges.

## REFERENCES

1. Dorfman, M. & Thayer, R.H. (Eds) Software Engineering, (November 1999), IEEE Computer Society.
2. Ghezzi, C. Jazayeri, M. & Mandrioli, D. Fundamentals of Software Engineering, (January 1991), Prentice Hall.
3. Pressman, R.S. Software Engineering : A Practitioner's, 4th edition (August 1996), McGraw Hill College Div.
4. Redwine S.T. & Riddle, W.E. Software Technology Maturation, *Proceedings of the 8th International Conference on Software Engineering*, 1985, pp 189-200, IEEE Computer Society.
5. Sommerville, I. Software Engineering (International Computer Science Series), 5th edition (November 1995) Addison-Wesley Pub Co.

















| 14 Software Engineering for Real-Time |   | Compositionality | Change | NF Properties | Service view | Perspectives | Lifecycles | Architecture | Configurability | Domain specificity | Links |
|---------------------------------------|---|------------------|--------|---------------|--------------|--------------|------------|--------------|-----------------|--------------------|-------|
| 14.1                                  | The development of a system architecture that supports the precise specification of the interfaces between components in the value domain and in the temporal domain, such that the components can be developed and tested independently. |                  |        |               |              |              |            |              |                 |                    | 4     |
| 14.2                                  | The constructive integration of existing prevalidated components into diverse system contexts.  |                  |        |               |              |              |            |              |                 |                    |       |
| 14.3                                  | The systematic validation of ultradependable real-time systems that are used in safety critical applications.   |                  |        |               |              |              |            |              |                 |                    |       |
| 14.4                                  | The development of a framework that supports the generic implementation of fault-tolerance without introducing additional complexity into the application software.   |                  |        |               |              |              |            |              |                 |                    | 12    |
| 14.5                                  | The derivation of tight upper bounds for the worst-case execution time of real-time programs.   |                  |        |               |              |              |            |              |                 |                    |       |

| 15 Software Engineering for Safety |   | Compositionality | Change | NF Properties | Service view | Perspectives | Lifecycles | Architecture | Configurability | Domain specificity | Links             |
|------------------------------------|---|------------------|--------|---------------|--------------|--------------|------------|--------------|-----------------|--------------------|-------------------|
| 15.1                               | Provide readier access to formal methods for developers of safety-critical systems by further integration of informal and formal methods.   |                  |        |               |              |              |            |              |                 |                    | 10, 7             |
| 15.2                               | Develop better methods for safety analysis of product families and safe reuse of Commercial-Off-The-Shelf software.   |                  |        |               |              |              |            |              |                 |                    |                   |
| 15.3                               | Improve the testing and evaluation of safety-critical systems through the use of requirements-based testing, evaluation from multiple sources, model consistency, and virtual environments.   |                  |        |               |              |              |            |              |                 |                    | 4                 |
| 15.4                               | Advance the use of runtime monitoring to detect faults and recover to a safe state, as well as to profile system usage to enhance safety analyses.  |                  |        |               |              |              |            |              |                 |                    |                   |
| 15.5                               | Promote collaboration with related fields in order to exploit advances in areas such as security and survivability, software architecture, theoretical computer science, human factors engineering, and software engineering education. |                  |        |               |              |              |            |              |                 |                    | 16, 12, 6, 11, 25 |















