Software Engineering

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.

The scientific and mathematical foundations of software engineering are derived from computer science and to a lesser extent systems science and control theory. It draws on engineering methods from other disciplines notably electronic and mechanical systems. The professional practice of software engineering is carried out by software engineers, who are qualified by virtue of experience, training and education. Ideally these are certified or otherwise approved by an appropriate professional body.

There are no satisfactory taxonomies of software engineering concerns. Rather there have grown up a diverse set of overlapping interests generally recognised by practitioners and researchers as areas of significant importance.

There are however, two main facets through which these interests can be viewed – *process* and *product*. Process is concerned with the organised set of tasks that a software engineer must carry out in order to develop a system. It is concerned with the nature of the tasks, how they are configured into an overall coherent workflow, the means by which these tasks are coordinated and environments that are necessary to support this process. By contrast product is concerned with the artefacts that are produced during a software development process, how they are represented, how they can be analysed and how key properties can be assured and the tools required to achieve this.

These two facets are best considered through an example. Establishing the requirements of users and other stakeholders in the development of a software intensive system is a major interest of software engineers. A large proportion of software development cost is attributable to this, it is a major determinant of system quality, errors associated with misunderstanding of requirements become increasingly, indeed exponentially, expensive to fix as development proceeds. This concern is known as 'requirements engineering'.

Specialists in requirements engineering are interested in processes for gathering, interpreting and managing requirements-related information. They are further concerned with how these processes fit with a larger software or system development process. Thus they are interested in how requirements relate to design, implementation, testing and so on. This is distinct from an interest in the products of requirements engineering, for example the representation of requirements-related information, using purpose-built specification languages and similar, and reasoning about this information to establish properties such as consistency, completeness and so on.

The major areas of software engineering interest are:

> requirements engineering – gathering, interpreting and managing requirements-related information, including the validation of that information with the stakeholders;
>
> software architecture – building structural descriptions of complex software-intensive system in terms of components, connectors and configurations of components and connectors;

software analysis – determination of critical properties of a software-intensive systems through the analysis of abstract models of their structure and behaviour, including verification of the relationships between such models;

software design – realising software architectures in terms of appropriate algorithms, program and data structures;

testing – supporting quality assurance through the dynamic analysis of the behaviour of the products of the software development process;

reverse engineering – the analysis of a software-intensive system to extract and create system abstractions and design information;

software maintenance and evolution – the control and management of modifications intended to ensure continued or improved operation in the context of changing environment;

reuse – the systematic reuse of knowledge gained during the software development process and artefacts ranging from requirements to program code during the development of subsequent systems or members of a product family;

formal specification – the use of representation schemes derived from discrete mathematics and logic to specify and construct large and complex software intensive systems;

middleware – the disciplined deployment of services derived from the software layer built above the network operating system to support the coordination of distributed software systems;

software reliability and dependability – engineering software-intensive systems to achieve properties related to the level of trust that users can have in the systems performing as required;

performance – engineering software-intensive systems to achieve properties related to the time dependent behaviour of those systems, particularly performance estimation and the relationship of the results of performance analysis to design models;

software safety – engineering software-intensive systems so as to avoid safety being compromised by the system entering dangerous states, particularly those in which failure may cause injury or death to human being;

real-time – engineering software-intensive systems which are required to respond to external events within a given, generally small, interval of time;

security engineering – engineering software-intensive systems where both the process must be carried out and the product must operate in the presence of potential adversaries;

mobility – engineering software-intensive systems in the context of unstable structures implied by physical mobility (mobile phones, PDAs, wireless networks) and logical mobility (mobile code, agents);

user-centered software engineering – engineering software-intensive systems where interaction with human users forms a critical component, particularly the integration of user interface design concerns within a broader software engineering process;

software configuration management – managing and controlling the products of the software development process, particularly managing the integrity of that information during system evolution;

software engineering tools and environments – developing automated support for software development tasks and frameworks through which such automated support can interoperate to underpin coherent processes;

software economics – engineering software-intensive systems with respect to value, cost and benefits, particularly the practical means for, and theoretical underpinnings of, allocating scarce development resources;

software metrics – the use of measurement of both products and process to control software-intensive development and to support decision making and risk management;

quality management – the organisation of management and technical systems for determining and assuring the conformance of the products of software development to their requirements.

Across all these areas there are a number of thematic topics that are worthy of note. Most significant of these are:

software engineering education and training – the development of curricula, materials and programmes for training and education in software engineering principles and practices;

software engineering professionalism – the development of a professional ethos, accreditation scheme and ethics for software engineers;

software engineering research methodology – the identification of appropriate research methodologies to support validation of the findings of software engineering, notably the combined use of observation, experiment, case-study and mathematical proof.

The findings of software engineering are delivered in a variety of ways primarily through processes, languages and tools. More rarely these findings are raised to the level of general principles applying to the engineering of classes of software system. The characteristic mode of delivery of software engineering findings is the software engineering method (commonly, though erroneously, methodology) an organised package of process, languages and heuristics derived from experience. The use of engineering standards as a means of delivering guidance on good practice forms a useful baseline.

It is very difficult to summarise the overall state-of-the-art in software engineering in a short space. 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 [6], [4] and [3], 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 context of software engineering is changing. Systems are rarely developed from scratch; most system development involves extension of pre-existing 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 centred 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.

Software engineering is addressing this changed context by focussing on a set of key challenges:

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 in software engineering?

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?

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?

These overall software engineering challenges are not orthogonal and there are complex relationships binding them together. Much of the most interesting recent work looks at these relationships. For an account of this see [2].

Software engineering 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. Software engineering innovations take a surprisingly long time to percolate through to every day use [5]. Despite this lag, current software engineering practice is being radically reshaped by object-oriented design methods, design tools with powerful code generation, testing and analysis environments, development patterns, incremental delivery based life-cycles, component models and document management environments, all the product of software engineering.

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 reach. What makes software engineering exciting is that, in addition to the steady progress towards the 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. Finkelstein, A. (Ed) The Future of Software Engineering", (May 2000); ACM Press.
3. Ghezzi, C. Jazayeri, M. & Mandrioli, D. Fundamentals of Software Engineering, (January 1991), Prentice Hall.
4. Pressman, R.S. Software Engineering: A Practitioner's, 4th edition (August 1996), McGraw Hill College Div.
5. 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.
6. Sommerville, I. Software Engineering (International Computer Science Series), 5th edition (November 1995) Addison-Wesley Pub Co.

Anthony Finkelstein
University College London

July 10, 2000