Software Engineering Directions

This lecture is a summary of “The Future of Software Engineering” prepared to celebrate the millennium and as part of the 22nd International Conference on Software Engineering.

`www.softwaresystems.org/future.html`

Process

- The scope of software improvement methods and models should be widened in order to consider all the different factors affecting software development activities (*e.g.* business).
- Modelling languages must allow for incomplete, informal, and partial specification.
- Must provide the software engineer with a clear state of the software development process (from many different viewpoints).
- Statistics is not the only source of knowledge. We should also appreciate the value of qualitative observations.

Requirements Engineering

- Better modelling and analysis of problem domains, as opposed to the behaviour of software.
- Development of richer models for capturing and analysing non-functional requirements.
- Better understanding of the impact of software architectural choices on the prioritisation and evolution of requirements.
- Reuse of requirements models to facilitate the development of system families and the selection of COTS.

Reverse Engineering

- Investigate infrastructure, methods, and tools for continuous program understanding to support the entire evolution of a software system from the early design stages to the long-term legacy stages.
- Develop methods and technology for computer-aided data and database reverse engineering.
- Develop tools that provide better support for human reasoning in an incremental and evolutionary reverse engineering process that can be customized to different application contexts.

Testing

- Creation of techniques and tools that can use precode artifacts, such as architectural specifications, for planning and implementing testing activities.
- Development of techniques and tools for use in estimating, predicting, and performing testing on evolving software systems.
- Establishment of effective processes for analyzing and testing software systems.
- Investigation of methods that use testing artifacts to assist in software development.

Maintenance and Evolution

- The production of new management approaches to evolution, leading to better understanding of the relationships between technology and business.
- How can software be designed so that it can easily be evolved?
- More effective tools and methods for program comprehension for both code and data
- A better formalism and conceptualisation of "maintainability"; how do we measure it?
- The development of a service-based model of software, to replace a product view.

Architecture

- Software architectures that support dynamic coalitions of software services.
- New techniques for composing heterogeneous components, and certifying the properties of those compositions.
- Software architectures that adapting themselves to their physical setting.
- Design principles for making architectural tradeoffs between correctness, resource consumption, and reliability.
- Self-monitoring systems.

Object-oriented Modeling

- To separate a core modeling language from domain-specific extensions.
- To define the semantics of a high-level, heterogeneous modeling language
- To develop means to compose and to refine complex structured models.

Analysis

- Checking conformance of code to designs is likely to become a central problem for software analysis.
- Tools that analyze designs in their own right will grow in importance.
- Abstract design models are the lynchpin for exploiting code analyses in this context: they not only make the analysis results more relevant, but can be used to focus the analysis and extend it.
- Both powerful tools that can check complex properties and simpler tools that provide rapid but rough results will be useful.
- Many kinds of analysis will play a role: static and dynamic, sound and unsound, operational and declarative.

Formal Specification

- Formal specifications need to be fully integrated with other software products and processes all along the software lifecycle.
- Specification techniques should move from functional design to requirements engineering; higher-level, problem-oriented ontologies must therefore be supported instead of program-oriented ones.
- The scope of formal specification and analysis must be extended to cover non-functional requirements that play a prominent role in architectural design – such as performance, security, fault tolerance, accuracy, maintainability, etc.
- Tomorrow's technology will provide lightweight interfaces for multiparadigm specification and analysis.

Reliability and Dependability

- Shifting the focus from software reliability to user-centred measures of dependability in complete software-based systems.
- Influencing design practice to facilitate dependability assessment.
- Propagating awareness of dependability issues and the use of existing, useful methods.
- Injecting some rigour in the use of process-related evidence for dependability assessment.
- Better understanding issues of diversity and variation as drivers of dependability.

Engineering and Performance

- To create a well understood formalism allowing performance annotations to design models.
- To create a methodology which embeds performance questions within the software lifecycle in terms of widely used approaches.
- To integrate solution tools for performance measures transparently within extended design tools, such as object oriented CASE tools.
- To develop ways of returning performance results from specialised tools in terms of the design models from which they were derived.
- To integrate performance modelling measures within a performance monitoring and testing framework in a consistent manner.

Real-Time

- System architectures that support precise specification of the interfaces between components in the value domain and in the temporal domain, allowing components to be developed and tested independently.
- The constructive integration of exisiting prevalidated components into diverse system contexts.
- The systematic validation of ultradependable real-time systems that are used in safety critical applications.
- The development of a framework that supports the generic implementation of fault-tolerance without introducing additional complexity into the application software.
- The derivation of tight upper bounds for the worst-case execution time of real-time programs.

Safety

- Provide readier access to formal methods for developers of safety-critical systems by further integration of informal and formal methods.
- Develop better methods for safety analysis of product families and safe reuse of Commercial-Off-The-Shelf software.
- 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.
- 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.

Security

- Integrating security considerations smoothly into early life-cycle activities.
- Architectures and designs that are easier to adapt to rapidly evolving security policies, and approaches to facilitate the integration of security features into legacy systems.
- The invention of cogent, flexible economic models of adversary behaviour which can underly the rational design of software copy-protection and watermarking techniques in different application contexts.
- Better techniques for formulating desirable security properties, and the development of scalable, predictable verification tools to evaluate the security of software systems.
- The development of automated, robust, flexible infra-structures for post-deployment system administration.

Mobility

- Mobility challenges old assumptions and demands novel software engineering solutions including new models, algorithms, and middleware.
- Coordination mechanisms must be developed to bridge effectively a clean abstract model of mobility and the technical opportunities and complexities of wireless technology, device miniaturization, and code mobility.
- Key concepts shaping software engineering research on mobility are the choice of unit of mobility, the definition of space and location, and the notion of context management.

Tools and Environments

- Linguistic and tool support for morphogenic software: software that is malleable for life, sufficiently adaptable to allow context mismatch to be overcome with acceptable effort, repeatedly, as new, unanticipated contexts arise.
- The development of new methodologies, formalisms, and processes to address non-traditional software lifecycles, and the tool and environment support to facilitate them.
- The development of new methodologies, formalisms, processes and tool and environment support to address the engineering of software in new, challenging domains, such as pervasive computing and e-commerce.

Configuration Management

- The semantics of basic core functions of configuration management systems, like versioning, data management or workspace support.
- Tools covering consistently and homogeneously the development and evolution of products containing software and other kind of components.
- Using the Web for CM, and CM for Web artifacts.
- A configuration management core as part of a kernel on which many company tools cooperate and interoperate.

Economics

- Principles, models, methods and tools for reasoning about and dynamic management of software development as an investment activity.
- Models for reasoning about benefits and opportunities in software development as well as costs and risks.
- Principles, models, methods and tools for dealing with uncertainty, incomplete knowledge, and market forces, including competition and change, in software development.
- Principles, models, methods, and tools for resolving multi-attribute decision issues in software design and development.

Metrics

- Software metrics should provide information to support quantitative managerial decision-making during the software lifecycle.
- Good support for decision-making implies support for risk assessment and reduction.
- Traditional metrics approaches, often driven by regression-based models for cost estimation and defects prediction, provide little support for managers wishing to use measurement to analyse and minimise risk.
- Using relatively simple existing metrics to build management decision-support tools that combine different aspects of software development and testing during the software life-cycle.
- Handling key factors largely missing from the usual metrics approaches (*e.g.* causality, uncertainty, subjective evidence).

Education

- Identifying distinct roles in software development and providing appropriate education for each.
- Instilling an engineering attitude in educational programs.
- Keeping education current in the face of rapid change.
- Establishing credentials that accurately represent ability.