Does Software Engineering have a Future?

Anthony Finkelstein

Professor of Software Systems Engineering

University College London

A.Finkelstein@cs.ucl.ac.uk

http://www.cs.ucl.ac.uk/staff/A.Finkelstein

What is 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.

Observations on a Changing Environment

- extension of pre-existing systems and integration with "legacy" infrastructure;
- systems embedded in complex, highly dynamic, decentralised organisations;
- support for business and industrial processes which are continually reorganised to meet changing consumer demands;
- services that such a system provides must, for the life of the system, satisfy the requirements of a diverse and shifting group of stakeholders.

Observations on a Changing Environment

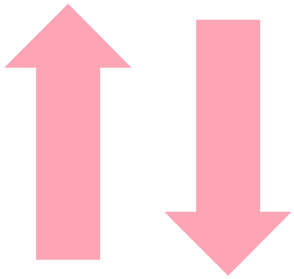
- a shift towards client and user centred approaches to development and an accompanying shift from a concern with *whether* a system will work towards how *well* it will work;
- fewer "bespoke" software systems are being constructed, instead, generic components are built to be sold into markets;
- components are selected and purchased "off the shelf" with development effort being refocused on configuration and interoperability.

Software Systems

- composed from autonomous, locally managed, heterogeneous components,
  - these components are required to cooperate to provide complex services;
  - they are, in general, distributed and have significant non-functional constraints on their operation;
- changing business models relating to the provision of software and software-mediated services

Tension

Rapid change and reconfiguration of business services



Increasing business dependence on reliability of infrastructure

Grand 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?

Grand Challenges

- 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?

Grand Challenges

- 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?
- Configurability
  - How can we allow users, in the broadest sense, to use components in order to configure, customize and evolve systems?

Grand Challenges

- 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?
- Domain specificity
  - How can we exploit the properties of particular domains (telecommunications, transport) to make any of these challenges easier to address?

Thematic Concerns

- "Malleable" software
  - Software which anticipates likely evolution scenarios and variations and is *robust* with respect to changes
- "Multi-dimensional" engineering
  - Ability to separate concerns and engineer products through any perspective

Research Goals

- Metrics - Integrated decision support for risk assessment and reduction.
- Analysis - Making formal analysis useable in real software engineering projects.
- Testing - Testing techniques for component-based development.
- Architecture - Architectural principles for systems that exhibit the scale and variability of network-centric applications and the dynamism of pervasive computation.

Research Goals

- Configuration Management - Efficient, scaleable, available management of information resources integrated with process support.
- Software Economics - Linking technical parameters with value to allow management of software development processes for, and tracking of, value.
- Empirical Studies - "Evidence-based" software engineering practice.
- Databases - Unbundling and rebundling of database components and provision of flexible cooperation schemes on top of these components.

Research Goals

- Education - Ability to impart an engineering attitude to students. Curricula which give long-term value but are flexible.
- Reverse engineering - Reverse engineering through all levels of the software development process.
- Performance - Integration of performance modelling with software architecture and design modelling.
- Safety - Safety analysis for COTS. Lightweight formal modelling.
- Requirements Engineering - An integrated "whole product-life" approach to requirements management.

Research Goals

- Security - Ability to separate the security aspect and engineer for it. Engineering for secure computation not secure computers.
- Mobility - Improved formal models of mobility and of context awareness. Middleware for mobility.

A Way Forward

- Real problems
- Lightweight solutions
- Raising the validation barrier
- Methodological diversity
- Feeding back to systems engineering
- Removing the "bug-centric" view
- Positive view of the discipline, pointing out our achievements.

And perhaps that's a good place to finish