
Construction By Configuration: a new challenge for software engineering education

Prof. Ian Sommerville
Lancaster University

Responsibilities of SE educators

- Principles
 - Teach students fundamental principles of software engineering whose rate of change will be slower than changes in technology and platforms
 - E.g. teach students about data abstraction not just how to create objects
- Technologies
 - Introduce students to current technologies
 - E.g. how to use the UML for system modelling
- Processes
 - Sensitise students to the realities of industrial software development
- Skills
 - Equip students with specific skills (e.g. system modelling) that will allow them to become economically productive members of the workforce

The changing face of SE

- For business systems development, the predominant approach for application system development is based around software reuse rather than original software development
- Only a minority of systems are developed in 'conventional' programming languages. Even when languages such as Java and C# are used, extensive reuse is the norm
- As technologies improve and processor performance increases, the same approach will inevitably be transferred to the development of other types of system

Current teaching in SE

- The majority of university curricula in software engineering are based around a conventional development process
 - Mathematical foundations (e.g. formal specification)
 - Programming in an OO language such as Java
 - UML based exercises in design
 - Projects to specify, design and program small systems
- Issues around software reuse are not given a high priority in most curricula
 - ACM 2004 curriculum guidelines suggest that only a very small percentage of time is devoted to reuse (although the document suggests that reuse should be encouraged)

Software reuse

- Software reuse is implemented in a number of different ways
 - Component-based Software Engineering (CBSE)
 - Service-oriented Systems
 - System families
 - ERP systems (SAP, Oracle, etc.)
 - COTS-based (vertical) software development

Domain abstractions

- The common features of all of these approaches to reuse is that they change the focus of system construction from technology-based abstractions to business-focused domain abstractions
- These abstractions (e.g. patients, appointments, nurses, treatments, etc.) are built into the system rather than defined.
- High level functionality is defined on these abstractions so programmer has less need to think about implementation abstractions such as selection, loops, sequences, etc.

Construction by configuration (C-b-C)

- Software development with reuse rarely means reusing abstractions without change.
- The reusable abstractions have to be configured to adapt them to their local circumstances of use.
- This can range from simple parameter setting through the definition of business rules to special purpose component development.
- Critically, reuse generally also means configuring the process (and sometimes configuring the customer) as well as configuring the software.

Reuse and configuration

- Components and services are intended to be used with limited configuration. Here the adaptation and configuration is often in the 'glue code' used to link these entities
- System families are configured by adapting specified parts of the system code
- ERP systems and generic COTS, by contrast, are designed for configuration without access to the source code of the system.
 - In these systems, the system development is the configuration.

Teaching about reuse

- Teaching about software development with reuse is important for 2 reasons:
 - Students should be aware of the realities of industrial practice
 - We should be trying to ensure that students think about problems in terms of reuse rather than inventing their own solutions

COTS-based reuse

- COTS-based reuse is generally based around some system for a specific application domain (e.g. patient information system in healthcare)
- This incorporates generic functionality and abstractions from the domain.
- These are then configured for each specific customer.
- This can be a (very) long-term process
 - The current Spanish en-route air traffic control system is being reconfigured for use in the UK. The process is anticipated to take 6 years.

Realities of COTS-based SE

- The decision on which COTS system to use is rarely based on a detailed analysis of the requirements in a specific setting
- Issues that affect the decision include:
 - Political issues
 - Platform issues
 - Cost and schedule issues
 - Availability of expertise
 - Prejudice!

A patient management system

- We have followed the deployment of a patient management system for mental healthcare in Edinburgh, Scotland
- Based around a generic COTS-package developed in the UK for hospitals in England which was designed to be adapted for different types of clinic (orthopaedic, diabetic, etc.)
- Scotland has its own legal system and laws and healthcare is a devolved responsibility. The Scottish executive sets its own targets and priorities for healthcare

Procurement issues

- A major influence in choosing that particular system was that it offered the opportunity for hospital managers (rather than doctors) to control how information was recorded
- The Executive placed a tight deadline on hospitals for reporting against a set of targets.
- There was little time to carry out a detailed comparison of alternatives and this system had already been successfully deployed elsewhere

Software engineering

- Configuring the data model required for a particular set of clinics
- Configuring the menus for the particular type of clinic and the patient information that had to be recorded
- Configuring the reports to be generated by the system
- Configuring the rules that should be applied to the system data
- **Configuring the process of use of the system**

Issues and problems

- Requirements conflict
 - The system was required to support reporting against a set of targets set by the Executive. This included reasons why the patient was referred to a clinic. However, the Executive's reasons for referral did not match the reasons normally recorded by clinicians.
- Invalid configuration assumptions
 - The language for defining the rules of the system was not expressive enough to cover the requirements of Scots law regarding the forced detention of patients
- Failure to configure the process
 - Local process differences meant that different clinics recorded different information about patients

The C-b-C process for COTS

- There is often no clear distinction between specification, design and development
- Systems are rarely completely configured before being put into use - the configuration process continues as the system is integrated with operational processes
- There is often extensive “user” configuration of the system
- It is often necessary to configure the expectations of system stakeholders

C-b-C and conventional SE

- Two-stage system requirements process
 - Identify generic environments to choose reusable system
 - Identify specific requirements from a setting of use
- More active stakeholder involvement?
- Co-realisation (process changes + s/w changes)
- Good practices such as configuration management, reviews, etc. are practically impossible to implement

System testing

- Testing is a particular problem for COTS-based systems
- Systems are not designed for running automated test suites
- There is no specification that can serve as a basis for deriving tests
- Problems that arise are often a consequence of interactions between the process of use and the system rather than system failure.

Problems of teaching C-b-C

- Principles
 - What are the principles of C-b-C that will equip students to transfer knowledge from one system to another?
- Technologies
 - What technologies can be practically introduced at a university level?
- Processes
 - How can students be introduced to the C-b-C process?
- Skills
 - Do the challenges of C-b-C require different skills from those acquired in a conventional SE course?

Principles

- The diversity of different approaches to C-b-C mean that identifying unifying principles across different systems is very difficult
- Possible examples of principles
 - Principle of visibility - make configurations explicit?
 - Principle of low coupling - reduce dependencies across configurations?
 - Principle of scalability - separate configuration of system deployment from configuration of functionality
 - Principle of localisation - localise volatile configuration entities

Technologies

- There are many different technologies that act as a basis for C-b-C. Can we choose a representative set of technologies?
- Many of these technologies are very expensive (e.g. SAP) and cannot be scaled down for use in a teaching environment
- Learning about specific (complex) technologies takes a lot of time - how can this be fitted into a (crowded) curriculum?

Processes

- Users may be partly responsible for parts of the configuration process. This causes major problems with supporting processes
 - Configuration management
 - Change management
 - Quality management
- As well as requirements, we need to discuss the implementation of processes such as system deployment and co-realisation processes.

Skills

- Selection and evaluation skills
 - Choosing the right system to configure
- People skills
 - Working with users and system stakeholders
- Prioritisation skills
 - Balancing different requirements against facilities offered by the system

Spreadsheet software engineering

- Spreadsheets offer the opportunity to introduce students to many of the issues around construction by configuration
- Spreadsheets (Excel or open source alternatives) are universally available and low-cost
- Using spreadsheet technologies allow real applications not toy programs to be developed

Learning from spreadsheets

- Alternative programming models (but do students realise that developing a spreadsheet is programming?)
- Importance of visibility - how are cells related to each other
- Working without a clear specification
- Problems of system testing

Spreadsheet projects

- Using spreadsheets offers the opportunity to work with real users in the University or SMEs
- Project examples
 - A system for costing research proposals
 - A system for supporting the development of a vegetation atlas
 - A system for tracking samples used in destructive testing
 - A system to manage information for a safety case

Conclusions

- Reuse is a reality of software engineering and we have a responsibility to incorporate this into CS curricula
- C-b-C is as technically challenging as original programming and students should be introduced to this
- More research is needed to establish a solid scientific basis for C-b-C
- However, a key question is how to find space in the curriculum for these new approaches. What should be dropped?