Software Development as the Core of Informatics

A. J. Cowling

Department of Computer Science
University of Sheffield

Structure of Presentation

Introduction
- Programming versus Software Development.

Describing Software Development
- A definition of software development;
- A brief history of software development:
  - in the curriculum,
  - and in practice.

Teaching Software Development
- What needs to be taught;
- Why it is important;
- How it underpins informatics.

Summary and Conclusions

Introduction

The Role of Programming
- The activity of creating software;
- Definition from Computing Curricula 91:
  - the entire collection of activities that surround the description, development, and effective implementation of algorithmic solutions to well-specified problems;
- Historically the core of informatics.

Creating Software in the Real World
- The importance of the software development lifecycle:
  - covers a wider range of activities,
  - many do not involve programming,
  - because real problems are often not well-structured;
- The core of informatics now needs to include all of these:
  - or at least the basic elements of them.

Defining Software Development 1

Programming versus Software Engineering
- Software development comes between them;
- They are stages in a progressive development of skills:
  - from programming,
  - through software development,
  - to software engineering.

The Role of Software Development
- A minimal meaningful subset of software engineering:
  - covering real-world applications,
  - not just adding models and processes to programming,
  - a shift in emphasis, from programs to systems,
  - a shift in emphasis, from technology to its purposes.

Defining Software Development 2

An Early Definition
- The set of basic methods for producing software systems:
  - so that SD covers the whole of the lifecycle;
  - is requirements elicitation through to deployment;
- Systems limited to feasible solutions to functional requirements:
  - in quality issues are ignored, for products and processes;
- Only two kinds of quality issues are considered:
  - the need for functional correctness;
  - the need for interfaces to achieve 'sufficient' usability.

Additional Restrictions on the Application Domains
- They must be simple and well-defined;
- Functional requirements must be typical for the domain;
- Hence domains are excluded that:
  - require particular non-functional properties, or
  - require special activities in the processes.

Defining Software Development 3

Consequences of These Restrictions
- They simplify the processes:
  - e.g. by reducing the set of models needed;
- They standardise the processes:
  - e.g. typically they imply the use of a standard architecture,
  - hence the range of design choices is reduced.

Common Factors in SD and SE
- Both involve the complete Lifecycle;
- Both focus on the need for meeting requirements.

Differences Between SD and SE
- SD only considers basic functional requirements;
- SD is a purely qualitative discipline, whereas
- SE ought to be a quantitative discipline.
Curriculum 68 and SD 1

Primary Structure
- Focus on Computer Science: used the term for all informatics;
- Identified three main divisions, and subject areas within them.

The Three Divisions
- Information Structures and Processes: divided into three subject areas;
- Information Processing Systems: divided into four subject areas;
- Methodologies: divided into ten subject areas;
- What we would now call applications areas.

Coverage of SD
- Methodologies included “Large-Scale Information Processing Systems”;
- Described as: care, the determination of the processing requirements, the statement of those requirements in a complete and unambiguous form suitable for the next steps, the design of the system, the construction of the programs and procedures, and the testing and operation of the assembled components in an integrated system;
- Hence covered the rest of the lifecycle outside programming;
- Had a limited range of content: reflected the limited knowledge of how to do SD;
- Of the 1968 Nato Conference on Software Engineering.

Curriculum 78 and SD 1

Primary Structure
- Core courses, at elementary and intermediate level: covering four sets of topics;
- Elective courses, at advanced level.

Sets of Core Topics
- Programming Topics:
- 5 topics, presented programming as an activity;
- 1 topic, “The emphasis is on the techniques of algorithm development and programming with style”;
- Hardware Organization:
- 7 topics, focusing on assembly language structures;
- Data Structures and File Processing:
- 7 topics, largely oriented towards programming.

Curriculum 78 and SD 2

Coverage of SD
- None in the core: only in two elective courses.

Software Design and Development
- This course covered three main aspects of SD:
- Design techniques, mainly focused on program design;
- Organisation and management of projects;
- A team project.

Database Management Systems Design
- The successor to the Curriculum 68 course;
- But, mainly concerned with database models and techniques;
- Largely ignored the rest of the lifecycle.

Curriculum 91 and SD 1

Primary Structure
- Separated implementations (courses) from knowledge;
- Nine compulsory technical subject areas: from the “Computing as a Discipline” report;
- Plus a tenth compulsory area:
- For social, ethical and professional issues;
- And a notional optional eleventh subject area:
- “Introduction to a Programming Language”; in practice, always included.

The Role of Programming
- Compare “Computing as a Discipline” and Curriculum 91:
- Many activities in computing are not programming ... therefore the notion that “computer science equals programming” is misleading;
- “Programming occurs in all nine subject areas in the discipline of computing.”

Curriculum 91 and SD 2

Coverage of SD
- “Software Methodology and Engineering” was a core area:
- It contained five knowledge units, and it was allocated about one sixth of the total time;
- But, two of those units were oriented heavily to programming;
- “Fundamental Problem-Solving Concepts”, and “The Software Development Process”;
- One had a significant amount of programming content: “Software Design and Implementation”;
- Only two focused on other activities within SD:
- “Software Requirements and Specifications”, and “Verification and Validation”;
- These had less than half of the time for the area.
Curriculum 91 and SD 3

The Role of Implementations
- Complete programme structures;
- defined courses at four levels, and sequences;
- mapped them into the knowledge areas and units;
- Nine main ones proposed (plus three others);
- Two of these nine were the most widely used;
- A for Computer Engineering and O for Computer Science;
- others omitted the area ‘Introduction to a Programming Language’;
- and so were generally seen as not realistic.

Features of the Two Popular Implementations
- Very similar introductory courses C101 and C102;
- The diagrammatic notations are useful in practice;
- There is no mention of the unified process.
- Three direct references to UML in CS2001;
- SD is no closer to the core of CS.

Computing Curricula 2001 and SD 1

Primary Structure
- Separate volumes for different disciplines:
  - Computer Science (CS2001),
  - Information Systems (IS2002),
  - Computer Engineering (CE2004),
  - Software Engineering (SE2004),
  - Information Technology (IT2005);
- An overview volume, actually produced last (CC2005).

Common Elements
- Each volume defined knowledge and course structures;
- CE and SE volumes imported material from the CS volume:
  - but changed its structure significantly.
- The CS volume tried to integrate its SE knowledge area better:
  - and added more non-programming material.

Course Structures
- Used levels similarly to Curriculum 91;
  - three main levels rather than four;
  - introductory, intermediate and advanced;
  - Coverage of SD moved to the intermediate level:
    - a course CS290 “Software Development”.

Content of CS290
- Still mainly focused on programming aspects:
  - event-driven programming, using APIs, computer graphics;
  - in one of the flavours of the course CS102;
- Only three syllabus sections for non-programming aspects:
  - human-centred software evaluation, human-centred software development techniques;

Overall Effect
- The CS volume tried to integrate its SE knowledge area better;
- SD was much more amenable to being taught.

Computing Curricula 2001 and SD 2

Overall Structure of Content
- Reflects the main technical activities in the SD lifecycle.

Requirements Analysis Skills
- Eliciting requirements information from clients:
  - interviews, analysis of documentation, observation, etc
- Constructing models to represent this information:
  - eg as UML class diagrams for business data,
  - UML use case and activity diagrams for functions.

Specification Skills
- Identifying the extra information needed for a specification;
- Eliciting this information;
- and maybe explaining its significance;
- Documenting it in some suitable form:
  - at least structured, but not necessarily formal.

Modern SD Practice 1

Impact On The CS Curriculum
- Better understanding of requirements analysis:
  - models and notations had been created;
  - their use had become more systematic;
- Better understanding of software design:
  - architectural and detailed design had been separated;
  - models and notations had been created;
  - standard structures (patterns) had been identified;
  - their use had become more systematic;
- The methodology wars had ended:
  - the UML had been created;
  - unified processes had been identified;
- Practice had become far more disciplined:
  - the importance of managing processes was recognised;
  - Hence, SD was much more amenable to being taught.

Modern SD Practice 2

Knowledge and Skills for SD 1

Common Elements
- Each volume defined knowledge and course structures;
- CE and SE volumes imported material from the CS volume:
  - but changed its structure significantly.
- The CS volume tried to integrate its SE knowledge area better:
  - and added more non-programming material.

Course Structures
- Used levels similarly to Curriculum 91:
  - three main levels rather than four;
  - introductory, intermediate and advanced;
  - Coverage of SD moved to the intermediate level:
    - a course CS290 “Software Development”.

Content of CS290
- Still mainly focused on programming aspects:
  - event-driven programming, using APIs, computer graphics;
  - in one of the flavours of the course CS102;
- Only three syllabus sections for non-programming aspects:
  - human-centred software evaluation, human-centred software development techniques;

Overall Effect
- The CS volume tried to integrate its SE knowledge area better;
- SD was much more amenable to being taught.
Knowledge and Skills for SD 2

Software Design Skills
- Architectural design is typically the use of a standard architecture;
- Detailed design operations within this framework;
- Involves creating models to explain the operation of the system:
  - eg UML, package and sequence diagrams;
  - Involves designing components for the system, such as:
    - its interfaces for users, and other input or output devices,
    - mechanisms for persistent storage of data.

System Construction Skills
- Programming;
- Designing test cases for unit and integration testing.

System Deployment Skills
- Creating installation and initialization procedures;
- Creating documentation for users.

Knowledge and Skills for SD 3

Relationship with CS2001 Model
- The basics for much of this material are already included:
  - processes and activities in the SE knowledge area,
  - user interface construction in the HCI knowledge area,
  - many of the component technologies;
- Additional coverage is needed for some aspects:
  - applying the skills, rather than just knowing the topics,
  - extension to larger scale constructions,
  - greater coherence of the different activities;
- These mainly affect course structures, not curriculum content.

Relationships with Other CC2001 Models
- CE2004 – similar, but for the embedded systems domain;
- IS2002 – a different structure, but similar SD knowledge is covered;
- SE2004 – covers this knowledge and skills;
- IT2005 – does not have any knowledge area for SE or SD.

Knowledge and Skills for SD 4

Relationships with CS2001 Course Structures
- Must introduce SD concepts earlier:
  - CS2001 defers most of these to advanced courses, eg CS390;
- Must not just start with programming-oriented courses:
  - eg the sequence CS101, CS102, CS103 and CS290;
- Must put much of the programming in its SD context:
  - the models needed for SD can be introduced in parallel;
- The following is a possible structure for the main core courses:

  Introductory Course Sequences (in parallel)
  CS101 Introduction to Programming → CS102 Data Structures & Algorithms
  SD101 S/w Lifecycles & Requirements → SD102 S/w Design & Construction
  Intermediate Courses (order not specified)
  Databases (eg CS270), HCI and User interfaces (eg CS250),
  Methods for S/w Analysis & Design, Integrating Project

The Importance of SD 1

Addresses Three Key Issues
- What the computing industry needs;
- The external image of informatics;
- Problems of retention of students in informatics.

Education versus Industry
- Undergraduate programmes must:
  - do more than prepare students for industry,
  - but also recognise the needs of potential employers;
- Industry needs SD, and more:
  - certainly not just programming,
  - SD is the core of what is required.

Informatics Beyond SD 1

Two Concepts of “Core”
- A formal core for each discipline in informatics:
  - defined in the volume for that discipline,
  - consists of the knowledge and skills that must be covered;
- An informal core for the whole of informatics:
  - the role of larger concepts or constructs within it,
  - such as programming or SD.

The Informal Core
- Is not just the intersection of the formal cores:
  - or some fixed subset of this intersection;
- Has slightly different roles in each discipline:
  - depending on its relationship with the rest of the discipline.
Structure of the Formal Cores
- Each has two components;
- One is the informal core of the whole of informatics:
  - should cover SD, not just programming,
  - reflects the application domains for the discipline;
- The other is the additional core for each discipline:
  - this reflects the philosophy of that discipline.

Relationships Between These Cores
- The additional material extends the scope of SD:
  - widens the range of systems that could be developed,
  - eg intelligent systems in CS, signal processing systems in CE;
- It also examines component technologies in more detail:
  - eg operating systems in CS, electronics in CE;
  - CS topics in IS and BE.

Summary
The Stages in the Argument
- SD is a clearly identifiable set of activities:
  - programming is just one of these;
- These activities have been recognised in all curriculum models:
  - but the early ones could only specify programming,
  - because the practice of the others was uncertain;
- The practice of SD is now well-established;
- Curriculum models have not kept up with it:
  - there is now a gap between them and industrial practice;
- This gap needs to be bridged:
  - to cover properties of large-scale structures,
  - to improve the external relationships of informatics;
- Bridging this gap only needs small changes in content:
  - but significant re-arrangement of course structures;
- Bridging this gap does not weaken the individual disciplines.

Conclusions
Overall Conclusion
- Software Development should be treated as the core of informatics;
- While programming is important, it is only one part of this core.

Future Work
- Course structures need to be analysed in detail:
  - to identify the changes required to support this core of
    Software Development;
- This must be done separately for each discipline.

The End
Thanks for your attention!!
Any Questions?