Software Engineering : A Career Path in Transformation

Pj Radcliffe

RMIT University, Melbourne, Australia
pjr@rmit.edu.au

Abstract : Software Engineering as a career and an industry has been in a state of flux since the 1940s. This places a special responsibility and burden on curriculum designers for any course containing a significant software content. If the wrong choices are made then graduates will have problems getting that first job, or progressing their career in the long term. We may also fail in our obligation to create software engineers who are useful to industry, government and society.

This paper first looks at key crisis and opportunities in Software Engineering as seen by practicing software engineers and academics. These include the development of new textural and graphical languages, the crisis of size, the triple calamity in 2000 where the dot-com, telecom and Y2K bubbles all burst, the stringent "immediately productive" demands from industry, and overseas competition.
Two other sources of information are examined – written graduate surveys which are of limited value, and graduate interviews which this author has found provide better insights. The interviews proved particularly useful in identifying what got a graduate get their first job, and what helps them prosper as their career matures.

Given a historical perspective and the current state of the industry some predictions can be made about the future of software engineering. These issues and predictions should be part of any software curriculum but even the ACM/IEEE recommended curriculum omits some significant items. Notable problems are a weakness in testing and a failure to concentrate on those portions of the software lifecycle that are more difficult to export.

Finally it would appear that the Australian software industry is recovering as the number of software graduates are on a downturn. This may lead to job shortages in key skill areas in the near future.

Keywords: Software Engineering, Changing Career Paths, Software Careers, Graduate Feedback.

Introduction

"Those who cannot remember the past are condemned to repeat it"
George Santayana (1863-1952)

Proceedings of the 2005 ASEE/AaeE 4th Global Colloquium on Engineering Education

The term "software engineering" encompasses the analysis, design, implementation and management of software and software projects. The term was formally coined in1969 at the first NATO conference on Software Engineering but the practical application of software engineering can be seen back as far as 1941 when Konrad Zuse demonstrated the Z3 relay based computer to a group of Berlin academics [1]. Since that time the skills required of a software engineer have changed dramatically as have the job opportunities.
Ideally universities should be at least tracking changes and preferably predicting changes. Armed with this information a university can better adjust the content and the size of their intake into their software degree programs.
An understanding of the history and predictions for the software industry is important to undergraduates and graduates as it will help them better understand the current environment for obtaining or keeping a job and deciding how to position themselves for the future.

This paper starts by examining the key historical changes in the software industry, the academic response to those changes, and the implications for software engineers. Finally predictions are made for the future of software engineering together with some recommendations for university course design. As will be seen the profession of software engineering has undergone significant changes and the pace of change will continue in the future.

First Generation Electronic Computers (Pre 1950)

The early computers such as ENIAC (US Army, operation 1946) [2] and Collosus (code breaking, UK 1943) [3] were based on thermonic valves and had to be rewired to serve a new problem. The skills needed by the "software engineers" of the time include a great deal of electronics design skills along with an understanding of digital electronics and algorithms.

At this point in time there was a very small number of computers in the world and a much larger number of electrical and electronic engineers. Workers on computers usually had to acquire a range of skills they did not learn at university.

Second Generation Solid State Computers (1950s – early 1960s)

The invention of the transistor in 1947 opened the way to smaller, more reliable, and more powerful computers. For example the thermionic valve based IBM 650 of 1954 cost $500,000, weighed around 1350 kg and required two large cabinets. The equivalent transistorised IBM 1620, which replaced the 650, was the size of an office desk [4].

Computers at this stage were freely programmable in that the program could be loaded into memory and re-wiring was not required. This technology change was a key point in the history of software engineering. A software engineer did not need to understand electronics theory in order to perform their job. At this point it was possible for a software engineering degree (computer science) to divorce itself from the electronic based material to concentrate on purely software material. Many university mathematics departments developed computing subjects and often spawned a separate computing department. Quite commonly computing also flourished in electronic and electrical engineering departments. Computing also became important to business departments who also developed software based subjects. This legacy

can be seen in many universities today where there is a two or three way tension over the ownership of software based subjects.

Expensive hardware – cheap labour: back in the 1950s and early 1960s industry found that computing hardware was relatively expensive compared to the cost of labour. The software development and management techniques of the day found ways to use cheap labour to minimise the use of expensive hardware. For example flowcharts were used to develop code and explain the function of code to reviewers before the program was allowed to run on the expensive hardware.

The role of software engineering oriented academics and courses in this period was to develop and teach tools and methods that made good use of cheap labour to service expensive hardware. For example most coding was done at assembly level (and even binary number level) as this produced shorter code that ran faster and so made more efficient use of hardware.

Third Generation IC Based Computers (Post 1964)

The third generation of computers relied on Jack St. Clair Kilby's and Robert Noyce's independent invention of the integrated circuit which led to Ted Hoff's invention of the microprocessor at Intel. These devices have follow Moore's Law [5] since 1965 which stated that the number of transistors on a chip would double every 2 years. Such sustained performance improvements are unparalleled in the history of technology. In 40 years the number of transistors on a chip has gone from 2200 [5] to around 1.7 billion in Intel's Itanium 2 [6].
Such massive technological change produced new opportunities and required a rethink of all work processes and tools.

Change : cheap hardware – expensive labour: the dramatic increase in transistors per chip led to a dramatic fall in the cost of computing hardware. In the 1950s a computer such as the IBM 650 cost around 500 years of programmer's salary. In 1978 a powerful development tool such as the Intel MDS-800 cost around 60% of a programmer's yearly salary. By 2005 a vastly more powerful computer such as a 3 GHz Pentium 4 costs around one week of programmer's salary, a ratio change of around 25,000 compared to the 1950s.

The response from academia and industry should have been to quickly change tools, methods, and hardware purchasing behaviour to use cheap hardware to minimise the relatively expensive labour costs.

Many industries were slow to respond to this change and persisted in wasting labour time to prevent buying hardware. This is not surprising as the costing methods used in most enterprises emphasised the obvious cost of hardware purchase and neglected the loss of productivity due to inefficient use of labour. The fault may lie partly with managers but universities share some of the blame. In general management and software courses of the day did not teach lifecycle costing which looks at much broader issues than monies paid for goods purchased and labour outgoings.

University courses were quite mixed in their response to the hardware-labour cost flip. Many software courses still omit lifecycle costing so graduates are never exposed to how to cost a

software project including issues such as lost labour productivity. This can lead to poor decisions about equipment purchases, and personal and company wide work practices.

Better tools and methods is one area where academia has clearly done well in research, development and teaching. The dramatic drop in hardware prices made it feasible to create and use tools that save labour time.
Examples of tools include high level languages such as C++, source level debuggers, and visual programming tools. The gains from such tools can be immense. Caper-Jones [7] has shown that a language such as C++ is around six times more productive than assembly code programming, and spreadsheet programming some fifty times more productive. Most visual programming tools such as spreadsheets, .NET environments, and packages such as Qt Designer represent tremendous improvements in productivity.

In parallel with better tools being developed better methods of developing software have been developed. Examples include Tom DeMarco's Data Flow Diagrams, Peter Chen's Entity Relation Modelling and more recently Object Oriented methods such as UML. As with better tools these better methods reduced labour costs.

A significant number of enterprises have been quite slow in adopting better methods and tools. For example until the 1990s it was common to find main frame users still writing major applications in assembler. New methods and tools are often described as disruptive technologies as they upset the norms inside an existing business and so tend to be rejected. According to Clayton M. Christensen [8] who studied the hard disk drive industry, even where an existing market player does develop a disruptive technology they tend to use it only for an existing market and not to develop new markets. Disruptive technologies and new markets are most likely to be adopted by smaller enterprises or a few more visionary larger enterprises.

Many software courses still omit any analysis of software technologies, notably disruptive technologies, and how they effect enterprises and markets in a competitive environment. Similarly the analysis of emerging markets, and indeed market failures, is missing from most courses.

There are a number of lessons that software engineers must learn from this history-

- Better tools and methods will probably be developed in the future.
- Disruptive or new tools and methods represent an opportunity for software professionals and companies to leap ahead of late adopters.
  Adopting new technologies carries with it the risk of failure but the potential gains make it attractive to new market players.
- Failure to be even a medium to late adopter may lead to business failure or job loss.
  (For example this author knows a number of top assembler programmers from the 1980s who never updated and are no longer able to get a programming job.)

Educational institutions have several responsibilities to their undergraduates-

- To explain the lessons above to the undergraduate population and to teach software project costing.
  (Most Australian courses known to this author do poorly in this regard.)
- To help develop those new tools and methods.
  (Most universities have an active research program that does address this issue.)

Complexity is the enemy: progressively more powerful computers allowed far more difficult problems to be solved using computers. Size and complexity is potentially an N^2 problem. Given N communicating objects or modules the potential interactions are N*(N-1). Doubling the size of a program can quadruple the number of interactions and thus the complexity of a design.
Larger projects have many more problems than smaller projects and have a much higher probability of failure [9]. If these new and larger software applications are going to be conquered, and provide jobs, then methods and tools must be developed to cope with ever large projects.

Why no labour oversupply? The adoption of steadily improving tools and methods should result in a shrinking labour market for programmers. A productivity increase by a factor of fifty should result in a labour pool decrease by a factor of fifty. Several factors have worked together to reduce the cost of software development and so many more applications of software have become economically viable -

- cheaper hardware,
- higher programmer productivity, and
- the ability to conquer problems of higher complexity.

The growth of the world's economy and the greater number of applications for software has produced a remarkable increase in the demand for software labour until around the year 2000.

Hype, Crash, and Export

The year 2000 saw a dramatic fall in the demand for software engineers. Three distinct factors were at work -

- The Y2K rework of the world's software was largely complete.
- Investors lost confidence in the Internet bubble thus share prices crashed and many Internet development companies were driven out of business.
- A separate bubble in communications technology, notably mobile phones and telephony, also burst.

Previous to 2000 there was a trend of increasing employment, despite vast improvements in labour productivity, because of the huge growth in software applications. The crash reduced the number of software applications being created or updated.
Even before 2000 market analysts were pointing to a share bubble effect and it was clear a crash would have to happen one day. From the perspective of an enterprise or employee what can be done given such prognostications? At minimum debt should be avoided and exit strategies should be developed. Where possible career and income diversification should be considered and speculative investments should be retired before any crash.

Exporting jobs: the year 2000 saw the exporting of software jobs begin to bite the market. Jobs at enterprises such as IBM-GSA in Melbourne Australia were sent to India where the labour cost was about one third of those in Australia.
The general industry experience in exporting software development has been mixed [10]. The successful exports appear to be projects that are large, can tolerate the added cost of management oversight of foreign operations, and projects that are well defined with a carefully worded and stable product specification and test plan. Even on these projects some types of work are more difficult to export. This includes the analysis phase which leads to the writing of a specification, user interface design, management oversight of the project, and

final testing. These work types represent key career skills that should be in the sights of any graduate aiming to practice software engineering in Australia.
The types of projects that do not export well are thus-

- Small to medium projects.
- Projects that must be completed in close contact with clients. For example projects that are suited to the XP style of software development.
- Projects with non-trivial unknowns.

While a significant number of software jobs have moved from Western countries to developing countries the job market in Western countries is still huge. In early 2005 www.mycarrer.com.au reported that Engineering and IT were the fastest growing job markets. Examining any Australian web based job site for professionals will show that there are many more IT jobs than other engineering jobs.

The Current State

The current state of the software engineering industry can be hard to assess particularly from the point of view of an academic who must decide what must be in a course and what should be left out. There are two clear responsibilities for a vocational course such as engineering : to give students the skills and knowledge necessary to gain their first job, and to enable them to prosper and adapt in the future. If this goal is met then the obligations to industry and government are almost certainly satisfied. Given this conclusion there is one obvious commentator on the course : graduates both recent and more distant in time.

In 1994 and 1996 this author tried sending out written surveys to graduates to help understand the state of industry and the ideal content of an undergraduate course. Few surveys were returned and the value of replies was very low.
Since 1996 the author has developed a new ploy. When students ask for a reference they are told "Yes" on the condition that they ring the author after they get the job and discuss the questions asked at the interview, and why it was they were offered the job and not the other candidates. Another condition is that they ring or email the author in one years time and discuss the skills and knowledge they needed to prosper and adapt. This approach has worked much better than surveying and has highlighted key issues that have effected the course content. Typical discoveries have included-

- Graduate naivety in regard to negotiation skills and work place politics was a significant problem.
- Having a major project to show at the job interview is a major asset.
- There are key technical skills that are commonly sought after in graduates.
- Graduates must be immediately useful in many enterprises and are expected to cope with a very steep learning curve.
- The most important filter in job interviews is usually not technical skills but communication skills including team work and the ability to be managed.
  The fact that communication skills rated more highly than technical skills was a surprise to this author but the same message has been delivered repeatedly over the years.

The discoveries outlined above have been incorporated into the curriculum of several subjects not just software engineering. At Electrical and Computer Engineering at RMIT there is a

stream of professional practice subjects from first to final year. These subjects deliver the theory and lectures in the financial, management and communications material. All technical subjects are expected to require students to use skills learnt in the professional practice stream.

Predicting the Future

The history of software engineering shows a number of constant factors and a number of unpredictable, disruptive developments. The constant factors will probably hold in the future and so should be a standard element is university education and the skill set of graduates. These include-

- Hardware prices will continue to drop as Moore's law should continue to hold for at least another 5 to 10 years. This implies that labour costs will remain a dominant factor in software development.
- Competitive pressures will continue. Tools and methods that are more productive will be developed and will give adopting enterprises a competitive advantage, and an advantage to individual software engineers who have mastered such tools and methods [11].
- The search for cheaper labour will continue more work will be exported to developing countries. Local enterprises and software engineers should identify and focus on project types and skills that are more difficult to export.
- Non-technical skills such as communication skills will continue to be highly rated in graduates.

Unpredictable, disruptive developments are by definition almost impossible to predict. Software engineering history shows a number of such developments and there is no reason to believe that disruptive developments will not happen again. At university level we can teach our future graduates to expect, to look for, and intelligently act upon, disruptive developments. Some teaching of the history of software engineering and the history of computing will help this goal.

Testing and reliability: there are a number of statistics that suggest future problems for software engineering in the area of testing. For example Atif Memnon [12] has found that the cost of testing has risen over the years and is now commonly 50% - 60% of project budgets. The complexity of large packages is so high that many defects are delivered to clients, for example the first delivery of Windows 2000 had 63,000 known defects [13]. Testing products at system level tends to be very weak [14].
There will probably be more work in testing as applications get more complex and customers get more demanding. Testing methodologies need to be improved.

The IEEE/ACM Software Curriculum 2004 [15] is an extremely good guide as to the content and structure of a software engineering degree. Given the foregoing discussion a few modifications can be suggested-

- Verification and Validation (testing) are covered but software creation subjects take far more time. Given the future trends perhaps V&V deserves more time.
- The analysis of labour markets, out-sourcing, and the effects of low labour cost countries are very important to graduates and industry and so should be mentioned as key issues in the curriculum.

More applications, more jobs? The dramatic increase in software applications fuelled an employment explosion until 2000 despite dramatic rises in labour productivity. What does the future hold? Will there again be a rapid rise in software applications and so job prospects? Will the exporting of software jobs stymie any growth?
The export of jobs is clearly significant but software development, unlike the textile manufacturing area or even electronics manufacturing, has significant niches that are hard to export. This will probably leave a job market that is significant but not explosive as before 2000. The evidence would appear to support an increase in IT jobs in the future-

How many software jobs?
On 7/5/2005 www.mycareer.com.au reported 455 electrical/electronics jobs and 3929 information technology and telecommunications jobs. www.seek.com.au reported the same categories as 430 and 13688 respectively.

- A visit to job web sites such as www.mycareer.com.au and www.seek.com.au will show that IT based jobs form one of the largest job markets for engineers.
- In January 2005 www.mycareer.com.au reported that IT&T (Information Technology & Telecommunications) and engineering jobs were the fastest growing job sector.
- IBM in Australia is doubling its graduate recruitment [16].
- Wendover corporation in the US is a very large surveyor of purchasing trends and claims to make over 2 million phone calls a year to collect data. Their IT industry results, based on 60,000 interviews [17], show a significant ride in IT spending in quarters 2 and 3 of 2005. This rise should link to an increase in IT jobs.
- A survey of 6000 IT employers in Australia showed 30% increasing hiring and 8.2% downsizing [18] in early 2005.
- The Career One job web site in Australia reports that Information Technology and Telecommunications recorded 56% growth over 12 months [19].

Since 2000 many universities in Melbourne such as Melbourne University, RMIT, and Monash University have shown a reduction in the intake and output of skilled software graduates. Given what could be a rise in demand there may soon be a shortage of IT professionals.

Conclusions

The history of software engineering is rich in both predictable and unpredictable changes. Universities have an obligation to produce graduates who are aware of the predictable changes and are ready to identify and cope with unpredictable changes. With this understanding graduates can optimise their own position and help to optimise the position of software based enterprises.
Software based courses in universities should include the following topics in order to achieve these goals-

- A study of software engineering economics and project costing.
- A study of the history of software engineering with an emphasis on predictable and unpredictable changes, and the effect on software engineers and enterprises.
  The concepts of personal career planning and contingency planning must be taught.

Given the predictable changes in software engineering the following specific material may need to be introduced or strengthened in a software engineering based course-

- Testing tools and methodologies to help comba t the rising costs of testing.
- Tools and methods to cope with more complex projects.
- Analysis of which portions of the software engineering process are less likely to be exported to low labour cost countries.
- Tools and methods that produce software at minimal cost.
- Communication skills including team work and being managed.
- Negotiation skills and work place politics.

The job prospects for the software industry look quite positive and with a decrease in software graduates coming through the university system there could be shortages in key skills in the near future.

References

[1] The Z3 programmable relay computer from 1941, retrieved on 30/4/2005 from
http://en.wikipedia.org/wiki/Z3_computer
http://www.cs.ncl.ac.uk/research/pubs/books/papers/133.pdf

[2] ENIAC description, retrieved on 30/4/2005 from
http://en.wikipedia.org/wiki/ENIAC
http://ftp.arl.mil/~mike/comphist/61ordnance/chap2.html

[3] Collosus description, retrieved on 30/4/2005 from
http://en.wikipedia.org/wiki/Colossus_computer
http://www.cs.ncl.ac.uk/research/pubs/books/papers/133.pdf

[4] IBM 650 and 1620 mainframe descriptions, retrieved on 31/4/2005 from
http://en.wikipedia.org/wiki/History_of_computing_hardware#Second_generation_--_late_1950s_and_early_1960s

[5] Moore's Law description, retrieved on 31/4/2005 from
http://www.intel.com/technology/silicon/mooreslaw

[6] Description of Intel's Itanium 2, retrieved on 31/4/2005 from
http://www.intel.com/pressroom/archive/releases/20040713corp.htm

[7] Caper-Jones (1995) , Backfiring: Converting Lines-of-Code to Function Points, *IEEE Computer*, November 1995, pg 87-88.

[8] Christensen, C.M. 1997, *The Innovator's Dilemma When New Technologies Cause Great Firms to Fail*, Harvard Business School.

[9] Gibbs, W.W. (1994), Software's Chronic Crisis, *Scientific American*, September 1994 pg 86.

[10] Sperling, S. (2004), Dealing with Hidden Outsourcing Costs, *Electronic News*, retrieved on 3/5/2005 from http://www.reed-electronics.com/electronicnews/article/CA381854.html

[11] Microsoft's concept of software factories, retrieved on 5/5/2005 from,
http://msdn.microsoft.com/architecture/overview/softwarefactories

[12] Memon, A. (2002), GUI Testing ; Pitfalls and Processes, *IEEE Computer*, August 2002, pg 87-88

[13] 63,000 defects in Windows 2000, retrieved on 8/5/2005 from
http://windows.about.com/od/security/l/aa021800a.htm

[14] Radcliffe, Pj (2004), Automatic Testing of Soft ware in University Courses, 15th Annual Conference of the AAEE, pg 147-154.

[15] Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, ACM/IEEE, retrieved from http://www.computer.org/cspress/CATALOG/c2350.htm

[16] IBM Doubles Graduate Recruitment (in Australia). Retrieved on 7/5/2005 from
http://www.careerone.com.au/newsviews/story/0,8523,12864873-22554,00.html

[17] Wendover corporation report on IT purchasing. Retrieved on 7/5/2005 from
http://www.itforecast.com/pages/16/index.htm

[18] IT jobs on the road to recovery. Retrieved on 7/5/2005 from
http://www.zdnet.com.au/news/business/0,39023166,20263437-1,00.htm

[19] Strong jobs growth follows slow start. Retrieved on 7/5/2005 from
http://www.careerone.com.au/newsviews/story/0,8523,12760805-22554,00.html

Bibliography

Pj Radcliffe is a senior lecturer in the school of Electrical Engineering at RMIT University. In 2004 he was one of four winners of the RMIT Student Centred Learning awards and also won a Student Choice award. His main interests are software engineering, software testing and Linux for application in the domains of electronic, networking and communication engineering.