Defining the Software Engineering Profession: Challenges Ahead

CUSEC Keynote Presentation, March 7, 2002

Timothy C. Lethbridge, University of Ottawa

Author of: “Object-Oriented Software Engineering: Practical Software Development Using UML and Java”
www.lloseng.com

Challenge 1: Understanding distinctive relationships with CS and CE

Three disciplines tussling for recognition: SE, CS & CE

Software engineering

- What is it?
  - Applying results from computer science and other areas to develop quality software **products** (and the software components of other products) within **economic constraints**
- It *doesn't matter* what type of software
  - E.g. financial, telecom, word processing, OS's
- SE Needs to assert its independence from CS and CE
  - To better promote an ethos of software quality

Challenge 1 (distinctive relationships)...

Computer science

- Was the original home and incubator of SE
  - But now needs to consider SE a separate but related community
- CS can focus more on its roots, advancing:
  - Mathematical methods
  - Languages and programming paradigms
  - Algorithms, data structures
  - Artificial intelligence
  - Innovative hardware and software architectures
  - Etc.

Challenge 1 (distinctive relationships)...

Computer engineering

- What is it?
  - Applying results from EE and computer science to develop **products** where new hardware designs are required
- However, many graduates end up focusing exclusively on software, hence there is a clash with SE
- CE developed from EE so is better understood in some professional engineering societies

Challenge 1 (distinctive relationships)…

A key problem:

- CS, CE and community college graduates often end up doing exactly what SE grads are educated to do
- So attitudes develop such as:
  - “If these other folks can do a good-enough job, why do we need specialist software engineers?”
  - “We only need specialist SEs for safety-critical or massive systems”
  - “SEs can do the project management and architecture, but anybody can design and program”
- But, the above are all wrong!!
  - SE education is needed to improve quality of all software, and all development phases

Challenge 1 (distinctive relationships)...

- I suggest we need to **gradually** move to an environment where people are educated in proper proportion to what *should be* the employment demand
  - Software mostly developed by software engineers
    - 30-40% of industry jobs
  - Computer scientists doing the research and new technology development
    - 15-25% of jobs
  - Computer engineers working mostly in hardware
    - 15-25% of jobs
  - Community college graduates helping with simpler aspects of all the above
    - 20-30% of jobs

Challenge 2: Ensuring the wider community understands what SE is

The general public understands *roughly* what other engineers do

Software engineering is a term many don't even know!

Parents and guidance counselors often advise high school students to enter CS or CE if they are interested in computers.

Challenge 2 (the wider community)…

We must all be ambassadors for our field!

- Tell people *proudly* you are a software engineer, especially when speaking to journalists, managers, teachers, regulators, other engineers, etc.
- Be ready with a good explanation of the differences among related disciplines (see Challenge 1)

In particular, professional engineering societies need a better appreciation of SE

- … more later

Challenge 3: Developing our standard body of knowledge

In the SE community, we have a lot of knowledge

- There has been much research
- We have powerful techniques and representation languages
- Many companies have developed methodologies and tools
- Huge numbers of people have a high level of skill

Challenge 3 (body of knowledge) ...

We are building

- SWEBOK
- Standardized curricula
- Many IEEE and other standards

But there are still important areas where

- A consensus is lacking about:
  - The ‘right’ way to do many things
    - E.g. formal vs. informal methods
  - The knowledge all SEs should know
- We need far more empirical data about what works and what doesn’t

Challenge 4: Adopting better tools more widely to become more efficient

Other branches of engineering make far more use of high-quality tools

Many commercial SE tools are poor

- Poor user interfaces, functionality and interoperability

Developers do not perceive the value in tools

- Too high cost and/or perceived learning time for perceived benefit

Developers are often too entrenched in their ways

- Or lack time to try out tools

Challenge 5: Building better education and continuing education programs

Many universities now have well-designed software engineering programs

Although some scope for improvement remains:

- Some omit aspects of the body of knowledge
- There is a lack of professors who have industrial SE experience in the core of SE
- There is a lack of good textbooks in specialized areas of SE
  - E.g. large system development
  - Textbooks need problem sets & should fit curricula

But what about after you graduate?

Challenge 5 (education) ...

Learning by doing is not enough

- You need to read good publications to learn new techniques and expand your horizons
- IEEE Software, Computer + CACM are good choices

Many professions require continuing education

- E.g. the medical profession
- The IEEE now has a certification program that requires re-testing every few years

Corporations also must also invest in their employees

- But not just in the latest languages and products!!!

Challenge 6: Improving ethical, legal and business foundations in our field

We have good codes of ethics

- ACM / IEEE

But there is a key problem that give us a bad reputation

- We market off-the-shelf software *full of bugs* and with *no guarantee*
- Consumers have to purchase and discover the problems later
- Big companies are trying to entrench this even more strongly through laws such as UCITA
- Some people say, “such software can never be made reliable economically” (which I believe is false)

Challenge 6 (ethics, legal, business) ...

Solutions:

- We need to promote a business environment where the following slogan will be a competitive advantage
  - **“Reliable software – guaranteed!!”**
- Consumer rights need to be protected better
- To achieve the above we need
  - Software engineers who are capable of developing software to a higher level of quality (Challenge 5)
  - Tools to help them do this efficiently (Challenge 4)

Challenge 7: Confronting the ongoing controversy about licensing

The original intention of engineering licensing was to protect the public from dangerous engineering designs

- Hence all engineers who independently perform engineering work must be licensed
- The license is issued only after strict educational and experience requirements are met.
- You do not need a license if you are working as an employee, and not signing contracts and or vouching for the safety of design

Challenge 7 (licensing) ...

Fallacy 1: We need licensing for *all* software developers

- Even most electrical and computer engineers are not licensed
- Much software is not safety-critical

Challenge 7 (licensing) ...

Fallacy 2: Software engineering education should be primarily for those who will be licensed

- I.e. for those developing critical systems
  - Hence the only reason for accrediting programs would be to protect public safety
- But the core educational needs of those developing large business systems are very much the same
  - Only domain-specific details differ
  - But details differ in *all* application domains
- Other non safety oriented fields also require stringent certification/licensing and program accreditation
  - E.g. accounting, financial analysis

Challenge 7 (licensing) ...

So:

- We should educate software engineers in the same way irrespective of whether they will be licensed
- All software engineering graduates should be eligible for licensing
  - If their programs are accredited to ensure they are taught how to develop quality software efficiently
  - So as to protect public safety
- And, in time, most software developers would have a software engineering education
- So the quality of non-safety-critical software should also be excellent

Conclusion

Being a graduate engineer gives you several benefits:

- Prestige of being able to call yourself an engineer
- Discipline of an engineering education
- Eligibility to become licensed
- Pride of being in the worldwide community of engineers

In time (10-25 years) I hope that most new software developers choose an SE educaton:

We need to encourage a better ethic of quality by:

- Improving our body of knowledge and our tools
- Improving the knowledge and skills of the workforce
- Guaranteeing the quality of shrink-wrapped software