Designing software for a million users – It's not about the technology

Professor Ian Sommerville

School of Computer Science, University of St Andrews, St Andrews, Scotland

Scotland was probably the first country in the world to deploy a national digital learning environment in 2006. This was an innovative and pioneering system but suffered from two fatal flaws. There were problems with the system's usability and it was a closed system. It offered a set of applications that were designed in the early years of the 21st century. Much more effective and usable applications had become freely available on the Internet before the system was deployed.

Our goal was to specify a replacement for this original system that avoided the flaws in its design and which provided a range of opportunities for teachers to use digital technologies to support learning in their classrooms. We wanted to create an open system that could integrate the web services and applications that teachers and students really wanted to use.

Potentially, this system has a large user base. There are more than 3000 schools in which it will be deployed ranging from tiny rural schools with a handful of pupils and a couple of teachers to huge city school campuses, with thousands of students. There are more than 50,000 teachers and almost potential student users. As an aim was to use the system to involve parents in their childrens' education, there are more than a million potential users of the system.

The key challenges that we faced were not technical problems. Issues we had to address were a very wide range of user abilities, complex system governance, a hostile user base, heterogeneous hardware and a desire to avoid political risk in deploying the system.

For many types of user-facing system, the technology is not the problem – it's the socio-technical stuff that causes the real difficulties. Socio-technical factors are the reason why many technically excellent systems don't really work that well. They're the reason why so many software-intensive systems projects are late, over-budget and fail to realise their objectives. As engineers, I think it's time that we faced up to these issues and thought about how we can better understand how they affect the systems that we are creating.

©Copyright held by the author(s)

Requirements for a digital learning environment

As part of the development process, I reckoned that we could use a methodical approach to deriving the requirements by using and adapting standard requirements engineering methods. These methods all involve:

- Understanding the business need and the environment where the system will be deployed.
- Scoping the system and establishing its boundaries
- Engaging with stakeholders to talk about the requirements for the system. These engagements may be facilitated with models of the existing system or the system that is being proposed.
- Documenting the requirements and checking these, to some extent at least, meet the business need and stakeholder requirements.

In our initial discussions, it became clear that there would be problems with activities that are fundamental to requirements engineering methods:

- Scoping the system. Some people thought the system should simply be a portal to generic tools; others believed that it would be more appropriate to orient the system around some model of pedagogy and yet others thought that the system should be the core of a whole school automation system.
- Engaging with stakeholders. The previous system was not well received by teachers and there was considerable political interference in its operation. This meant that many stakeholders were disenchanted and unwilling to spend time in engaging with us.
- System modelling. Whilst engineers are comfortable with abstractions and models, these are actually quite alien to many people, especially if their education is in arts and humanities rather than science. They find it very difficult to relate models to reality.

These problems meant that conventional requirements engineering methods were effectively useless so we turned to an approach developed as part of agile development – user stories.
User stories were originally developed as part of so-called Extreme Programming (XP), an agile method of software engineering. In that method, a user story was a description of an interaction that a user might have with a system and this description was used to decide on the

system features that could be implemented. We used higher-level stories that stimulated discussion about the system.
User stories were a great success. All of the members of the group could relate to them and they were effective in communicating our vision to a wider community. An unexpected bonus of using these stories was that politicians and civil servants with no technical or educational background could relate to them. The Scottish Secretary for Education commented how the stories brought the system to life for him and how refreshing it was to receive a report on a technical topic that he could understand.
The advantage of the user stories for the teaching community was that we made a point of developing stories for all stages of use of the system – from early years to young adults. We did not therefore require teachers to imagine how some scenario might be translated to their needs.

1 System architecture

Our final system architecture was designed around four fundamental principles:

1. Everything should be provided as a service. This means that everything is replaceable and distributable.
2. We would require the implementation of as few services as possible. For political reasons, we needed to have an authentication service so that the system could track, to some extent, who is doing what. We needed a set of configuration services to create different versions of the DLE and, again for political reasons, we needed storage services, to allow for local control.
3. Users could configure their own version of the environment. This configuration would not require detailed technical knowledge. Configuration could be at a local authority level, a school level or an individual teacher level.
4. Services could be loosely or tightly integrated. Tightly integrated services could use the system's authentication and storage services; loosely integrated services used their own authentication (which could be Google or Facebook authentication) and managed their own storage.

These led to the development of a layered architecture for the system as shown in Figure 1.

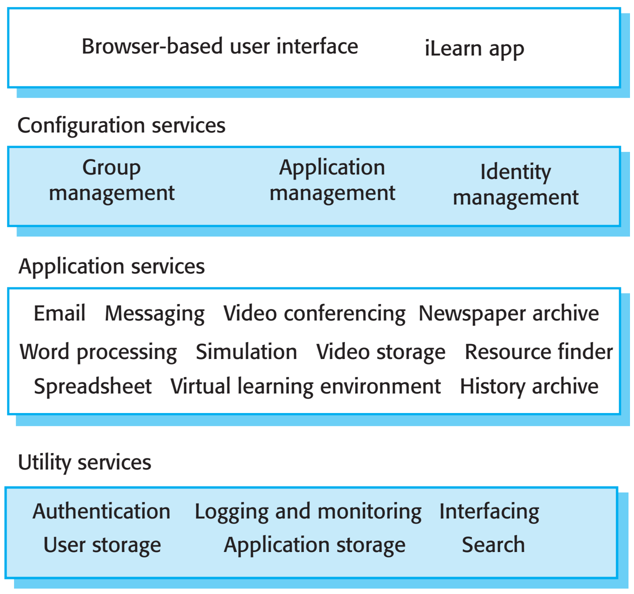

Figure 1. A layered architecture for a DLE

Lessons learned

There are four 'take-away' messages that reflect some of the socio-technical issues that I've discussed here:

- Methods may be useful where systems are being developed for a clearly defined purpose in a single managed organisation but in more complex organisations, they don't work.
- Most users don't care about new systems. The key challenges for engineers are finding ways to communicate with these users and building systems that are flexible enough to adapt to different styles of use.
- Governance is critical – if at all possible, reduce governance complexity as much as you can.
- User stories work – are by far the best way that I have found to communicate with a wide user community that has diverse backgrounds and experience.