Software deployment and maintenance

Nigel Goddard

School of Informatics
University of Edinburgh

What is deployment?

Getting software out of the hands of the developers into the hands of the users.

More than 50% of commissioned software is not used, mostly because it fails at deployment stage.

80% of the cost of (commissioned) software comes at and after deployment.

What are the issues that make it hard?

Is deployment the problem?

Not always.

Often, problems *show up* at deployment which are actually failures of requirements analysis. Disaster – such problems can be very hard or impossible to fix, in a large system. e.g. NPfIT...

However, there are also genuine transition issues.

Key issues around deployment

- Business processes. Most large software systems require the customer to change the way they work.
- Training. No point in deploying software if the customers can't use it.
- Support. The need goes on, and on, and on.
- Deployment itself. How physically to get the software installed.
- Equipment. Is the customer's hardware up to the job?
- Expertise. Does the customer have the IT expertise to install the software?
- Upgrades. Can't avoid them!
- Integration with *other* systems of the customer.
- Performance.

Deployment itself

Many people will sell you tools to help deploy software. Such systems help you to:

- package the software
- make it available (nowadays over Internet or on DVD)
- give the customer turn-key installers, which will:
- check the system for missing dependencies or drivers etc.
- install the software on the system
- set up any necessary licence managers
- ...

A relatively simple system for Java programs is Java Web Start. See suggested reading. But not all it's cracked up to be...

Good commercial installation-builder: install4j.
Best free installer-builder: izpack.org

Software evolution and release management

Discipline in the evolution of software is (at least) as important as in its development.

- gather change requirements: new features, adapting to system/business change, bug reports
- evaluate each; produce proposed list of changes
- go through normal development cycle to implement changes – *ensuring that you understand the software*, which may be non-trivial.
- issue new release

Unfortunately, emergencies happen, and things have to be done with urgency. If at all possible, go through the normal process afterwards.

Maintenance

Software has bugs. New features are required. Circumstances change. Therefore software is changed. *Who changes it?* Development team broken up – maintenance may be done by different company!

Repeated change leads to architectural degradation. Old systems may have been degraded from the start!

Software rots. Even with no code changes, the systems change, and eventually you can't compile the software.

Tracking bugs: Trac / Bugzilla / Jira / Mantis...

Re-engineering

Re-engineering is the process of taking an old or unmaintainable system and transforming it until it's maintainable. This *may* be considerably less risky and much cheaper than re-implementing from scratch.

Re-engineering may involve:

- Source code translation e.g. from obsolete language, or assembly, to modern language.
- Reverse engineering i.e. analysing the program, possibly in the absence of source code.
- Structure improvement, especially Modularization.
- Data re-engineering.

Issues: what is the specification? Which bugs do you deliberately preserve?

Reading

Suggested: The Java webstart page

Suggested: The izpack.org website

(see above or the web page for URLs)