What is deployment?

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

Some stats:

- 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.

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. Has this been properly thought through?
- **Training.** No point in deploying software if the customers can’t use it.
- **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?
- **Integration** with *other* systems of the customer.
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
  - ...
Maintenance

The process of changing a system after it has been delivered.

Kinds

- **Fixing bugs and vulnerabilities:** not only in code, but also design and requirements

- **Adapting to new platforms and software environments:** e.g. new hardware, new OSes, new support software

- **Supporting new features and requirements:** necessary as operating environments change and in response to competitive pressures
Maintenance challenges

- Often a new team has to understand the software
- Development and maintenance often separate contracts: De-incentivises developers paying attention to maintainability.
- Maintenance work is unpopular: seen as less skilled, can involve obsolete languages
- As programs age, structure degrades and are harder to change: Not only software itself, also compilers, documentation.
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.
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*, **architectural refactoring**
- **Data re-engineering**, reformatting and cleaning up data.
- **Adding adaptor interfaces** to users and newer other software

**Issues:**

- What is the specification?
- Which bugs do you deliberately preserve?
Reading

**Suggested:** Sommerville on software evolution and maintenance