After system delivery

Systems do not stand still after delivery
Three different kinds of change

- Software Maintenance Changed requirements are implemented but structure unchanged
- Architectural Transformation changes to system architecture are made *e.g.*, from centralised architecture to Client/Server
- Software re-engineering functionality not changed, but internal structure modernised

Maintenance

Three kinds of maintenance:

- Repair of Software Faults 17%
- Adaptation to different operating environment 18%
- Add Functionality 65%

Maintenance major item in the budget for software use
comparable to development costs

Reasons for high maintenance costs

More expensive to add functionality later. Additional (organisational) reasons:

- Team stability Maintainers different from developers
- Contractual Responsibility Maintenance contract often different from development contract
  ⇒ no incentive for development team to ease maintenance
- Staff skills Maintenance often assigned to most junior, inexperienced staff

Program age and structure can be outdated as well
Big problem: often quick fixes required
Proper remedy postponed (and abandoned eventually)

Architectural Evolution

Normal maintenance not enough for old legacy systems
Reasons:

- Hardware costs: many PC's cheaper than mainframe
- User interface expectation: GUI rather than forms
- Remote access must be supported

not possible via normal maintenance: requires substantial design and implementation ⇒ need very good business case to make it worthwhile
If structure is too complicated: transform client requests via middleware
⇒ can rebuilt the system slowly without change to user

Software Re-engineering

Architectural Evolution often too risky
⇒ legacy systems re-implemented without change in functionality or architecture
Aims:

- Redocumenting
- translating to more modern programming language
- modifying and updating structure and value of system data
- make system more maintainable

Activities in re-engineering

- Source code translation
- Reverse engineering
- Program structure engineering
- Modularisation
- Data re-engineering

Depends on availability of good CASE-tools
cannot replace architectural evolution

Year 2000 problem as example

To save space, two digits for year used
⇒ code and data had to analysed for use of year data
Apart from using four digits, other alternatives were used:

- Scrapping old legacy systems
- Re-interpretation of data (00 means 1950!)

Correct re-implementation too diffcult!