After Deployment

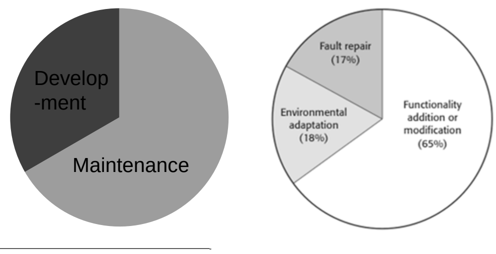
Evolution - Maintenance 9.1-9.3

Configuration management 25

Legacy systems 9.4

Re-engineering 9.3.2

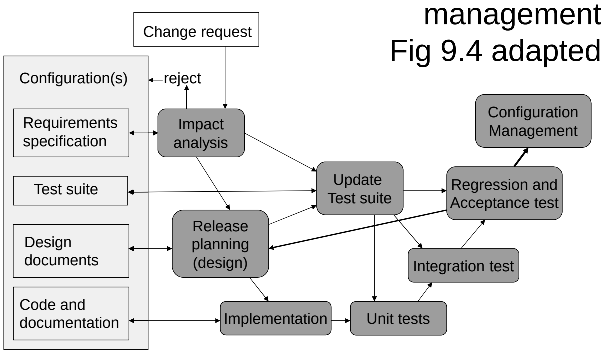
Figure 9.8 Maintenance effort distribution



Emergency code repair Fig 9.6

- Leave change request open
- Follow normal change request routine

Maintenance / evolution / configuration management
Fig 9.4 adapted



Change Request Form

Project: SICSA/AppProcessing **Number:** 23/02
Change requester: I. Sommerville **Date:** 20/01/09
Requested change: The status of applicants (rejected, accepted, etc.) should be shown visually in the displayed list of applicants.

Change analyzer: R. Looek **Analysis date:** 25/01/09
Components affected: ApplicantListDisplay, StatusUpdater

Associated components: StudentDatabase

Change assessment: Relatively simple to implement by changing the display color according to status. A table must be added to relate status to colors. No changes to associated components are required.

Impact analysis

- Change Control Board
  - benefits of the change
  - number of users affected
  - what if no change?
  - cost
- If change:
  - priority
  - fit in release cycle

Change Request Form

Project: SICSA/AppProcessing **Number:** 23/02
Change requester: I. Sommerville **Date:** 20/01/09
Requested change: The status of applicants (rejected, accepted, etc.) should be shown visually in the displayed list of applicants.

Change analyzer: R. Looek **Analysis date:** 25/01/09
Components affected: ApplicantListDisplay, StatusUpdater

Associated components: StudentDatabase

Change assessment: Relatively simple to implement by changing the display color according to status. A table must be added to relate status to colors. No changes to associated components are required.

Change priority: Medium
Change implementation:
Estimated effort: 2 hours
Date to SGA app. team: 28/01/09 **CCB decision date:** 30/01/09
Decision: Accept change. Change to be implemented in Release 1.2
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments:

Configuration items

- requirements
- design documents
- code - modules
- test suites
- documentation
- installation files/routines

Terminology

- Version
  - of an item
  - unique identifier
- Baseline
  - collection that cannot be changed (fall-back)
- Release
  - delivered to customer

Tool support

- Database
- Editing: check out ... check in
- System build
- Regression test
- Change reports, documentation

Figure 25.12 Continuous integration

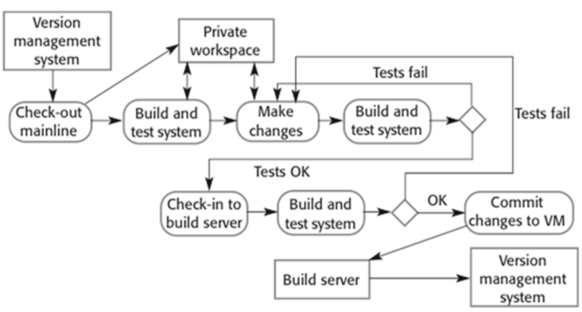
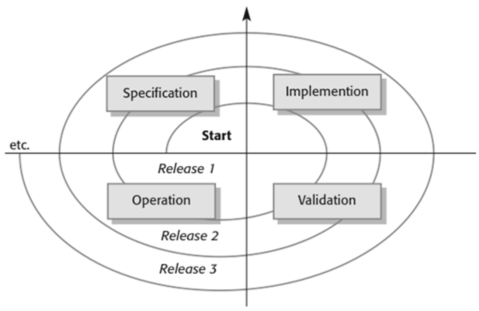


Figure 9.1 A spiral model of development and evolution

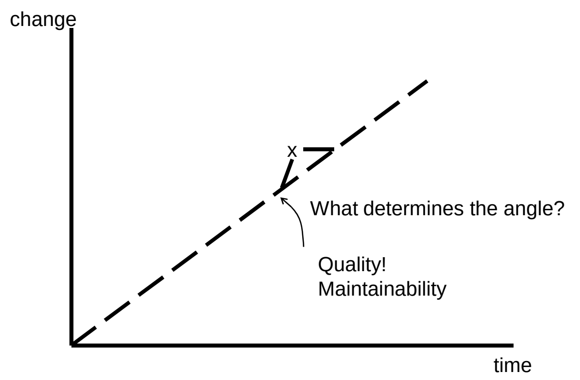


Evolution Dynamics (Lehman)

Law	Description
1. Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
7. Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
6. Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
2. Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.

Constant pace of change

Law	Description
8. Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.
3. Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
4. Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
5. Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.



Maintenance costs

- Maintenance costs more than development
  - loss of information
    - time
    - handovers
  - less skilled people
  - structure gets worse
- It pays to invest in maintainability
  - refactoring

Refactoring 9.3.3

- During development (evolutionary, incremental, agile)
- During maintenance

- ”code smells”
- design patterns
- documentation

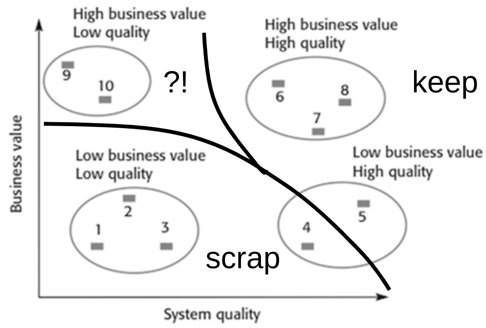
Legacy systems

- Old systems
  - $> 10^{11}$ LOC
  - date back to 70’s
- Hardware no longer available
  - ”don’t touch it” not an option
- Business rules implicit in software
- Data – a lot of it!
  - only accessable through this system

Legacy software

- Documentation lost (not maintained)
- Design – not modular
  – overoptimized
  – user interface (command line)
- Code – source code lost
  – old language
  – unstructured
  – badly patched

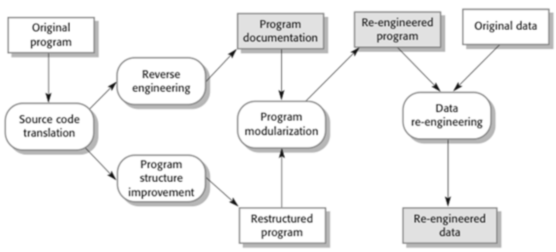
Figure 9.13 An example of a legacy system assessment



Re-engineering

- Goal:
  extract what we must / can reuse:
  – knowledge: business rules
  – data: conversion
  – design, code?
- Why?
  – reduce risk
  – reduce cost

Figure 9.11 The reengineering process



Legacy system wrapper

- Even if you keep the legacy system, ...
- how to interface with new systems