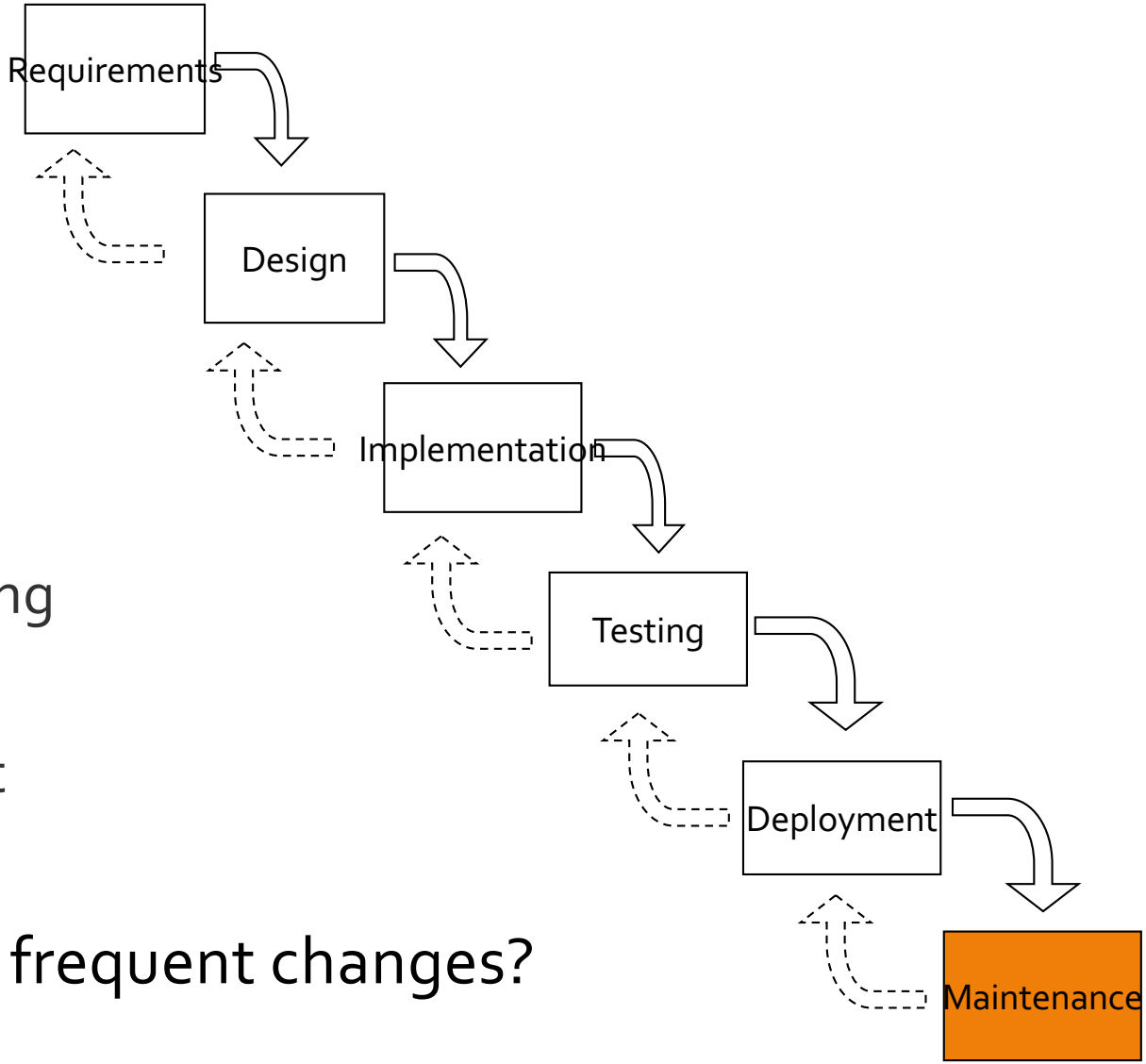
Software Evolution

- Maintenance vs. evolution
- Architectural erosion
- Laws of Software Evolution

CE202 Software Engineering, Autumn term
Dr Michael Gardner, School of Computer Science and Electronic Engineering, University of Essex

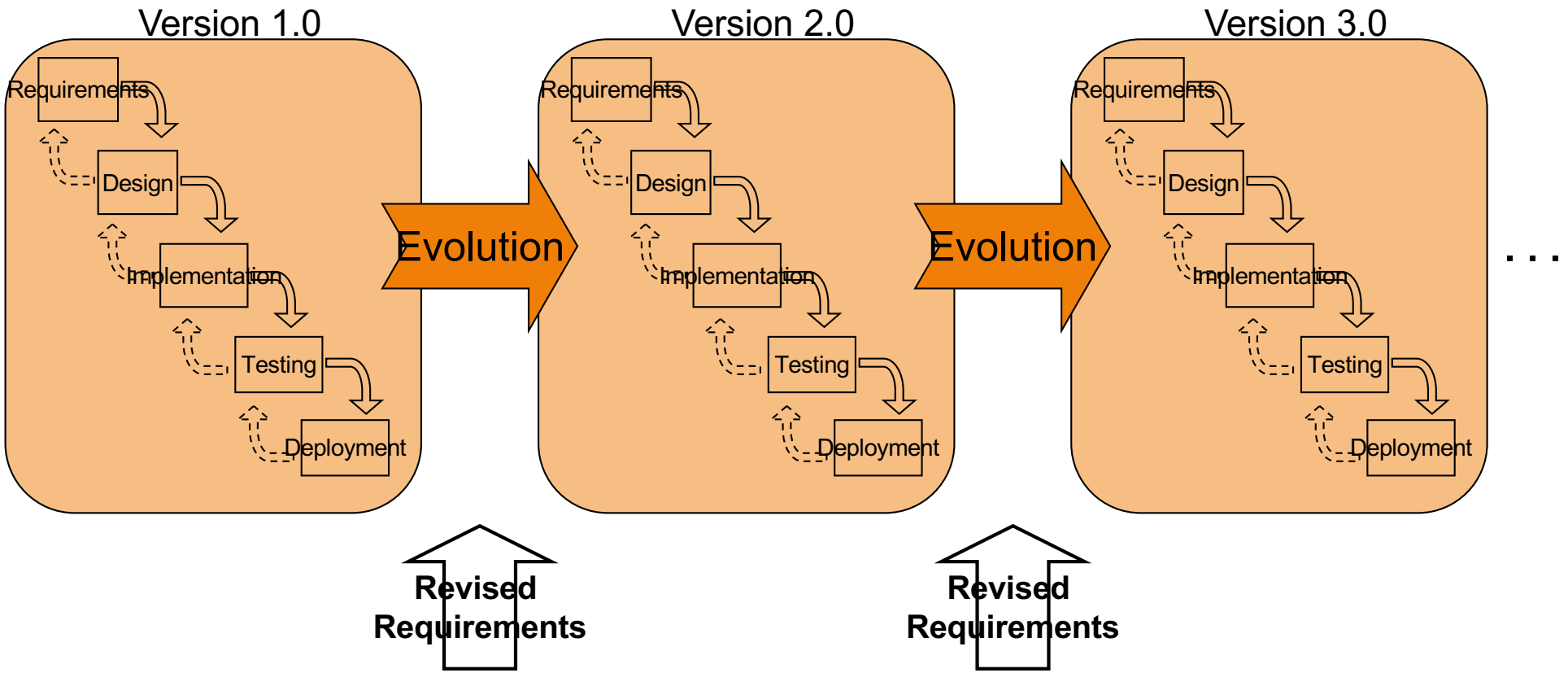
Reminder: Non-evolutionary lifecycle model: Waterfall Model

- Assumption: the entire project can be done in broad phases
  - Specify everything
  - Design everything
  - Implement everything
  - Test everything
  - Deploy final product
  - "**Maintain**"
- What about radical, frequent changes?



Reminder: Maintenance or evolution?

"Evolutionary" model: Reflects reality better!



Software maintenance

- A phase in the traditional waterfall model:
  - The "last" stage in the process
  - "Maintenance": Modifying a program after it has been put into use
  - Not geared towards major changes
    - E.g.: Changes in the system's architecture

→ Conclusion:

→ The waterfall model (and the term "maintenance") is suitable for small, "fixed" projects

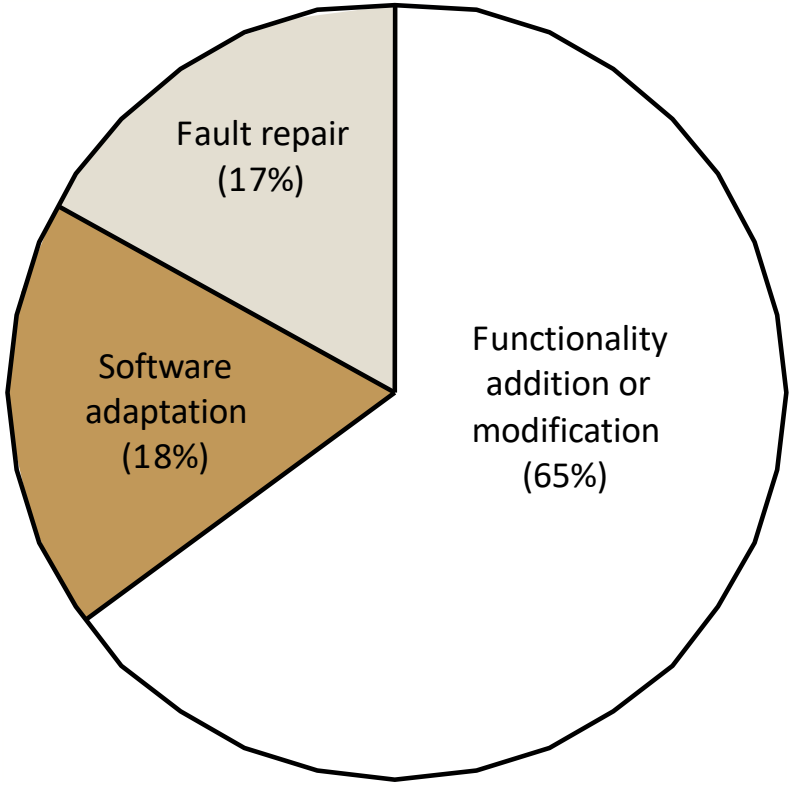
→ The waterfall model is also suited to very large projects which require rigorous project management

Types of changes

- **Bug fixing**: Faults discovered & must be fixed
- **Changes in the "operational environment"**
  - New hardware (computers, networks) added
  - Other components and software change
    - E.g., new operating system
- **Changes in customer's requirements**
  - The business environment changes
    - Example for new requirement: Streaming video to mobile phones
  - Clients' expectations increase
    - Performance, reliability & safety requirements may change
    - Competition: Change or lose!

Distribution of maintenance effort

From:
Sommerville, 2004



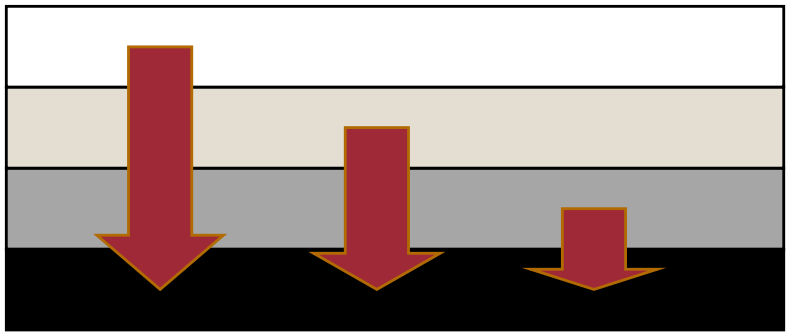
→ Conclusion: Software change is frequent, unexpected, and very expensive

Why is evolution so expensive?

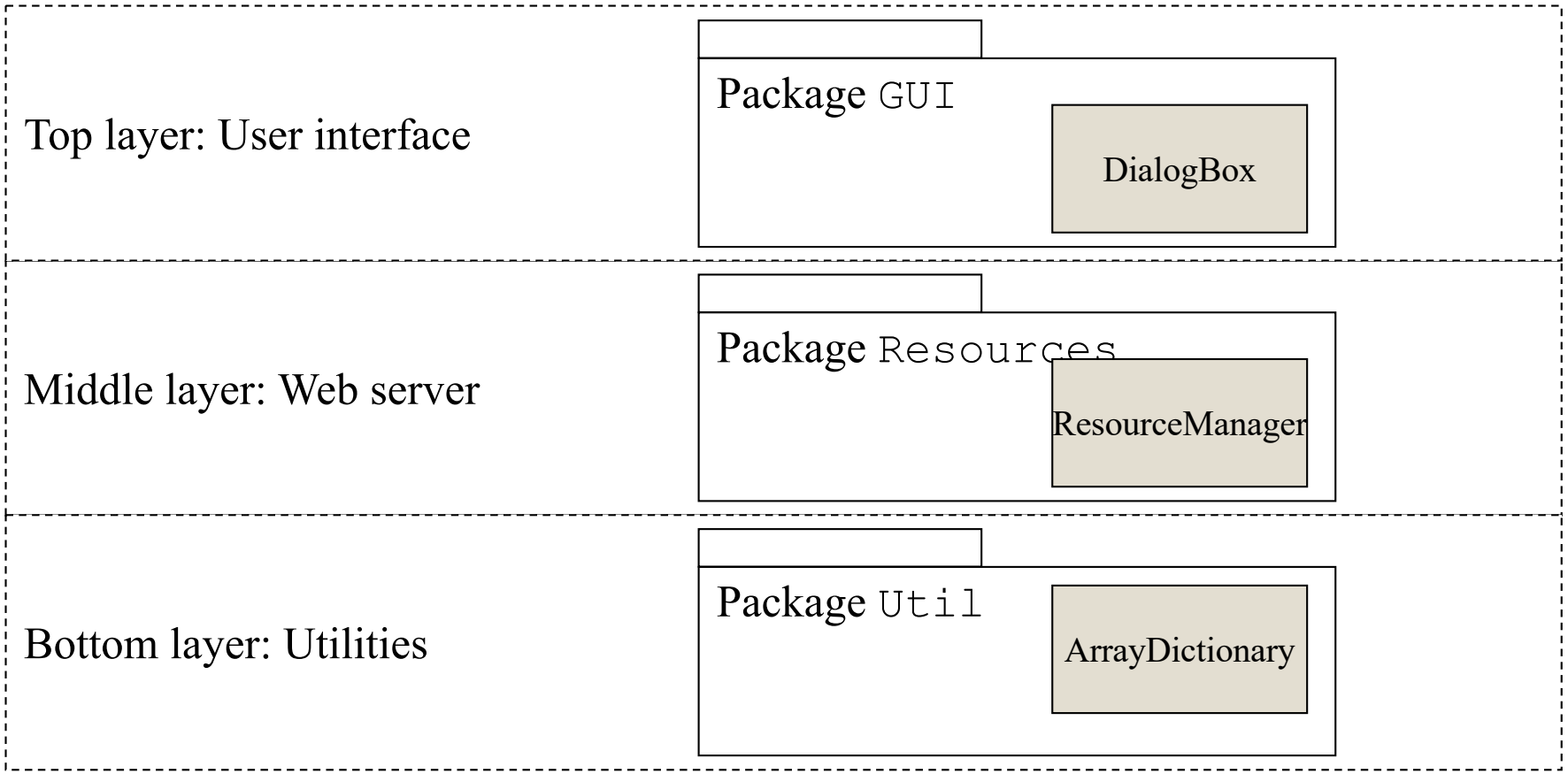
- Change prediction is very difficult
  - Which operating system will be used tomorrow?
  - Which hardware will replace the existing? How is it going to be different?
- Teams unstable
  - Short-term contracts with programmers
  - New programmers need be trained [Brooks 1975]
- Architectural erosion/drift [Perry & Wolf 92]
  - As programs age, their structure is degraded and they become harder to understand and change.
- Contractual responsibility
  - SW suppliers (usually) have no contractual responsibility for maintenance – no incentive to design for future change.

Example: Architectural erosion in a layered architecture

- Principle: Prevent cycles of dependencies between every part of the system
  - Each module belongs to a "layer"
  - And
  - Each layer may access only "lower" layers



Architectural erosion: example (cont.)



Architectural erosion: example (cont.)

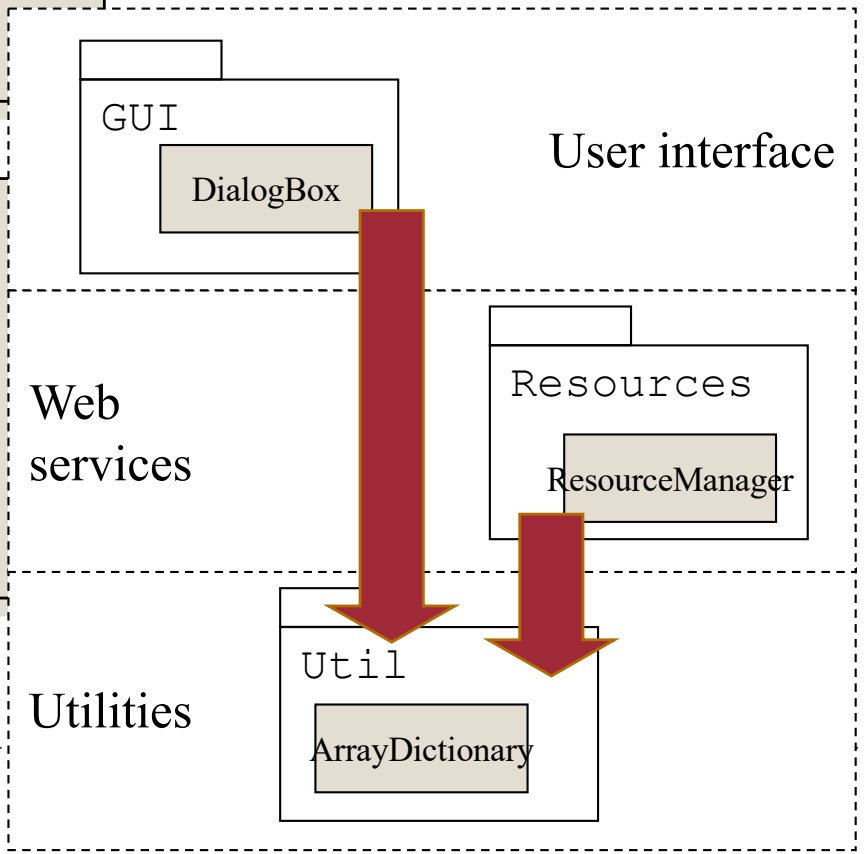
- Correct dependency: From upper to lower layer

```
package org.w3c.resource;

public class ResourceManager {
  private ArrayDictionary allResources;
...
}
```

```
package org.w3c.GUI;

public class DialogueBox {
  private ArrayDictionary
    Labels_to_Widgets;
...
}
```



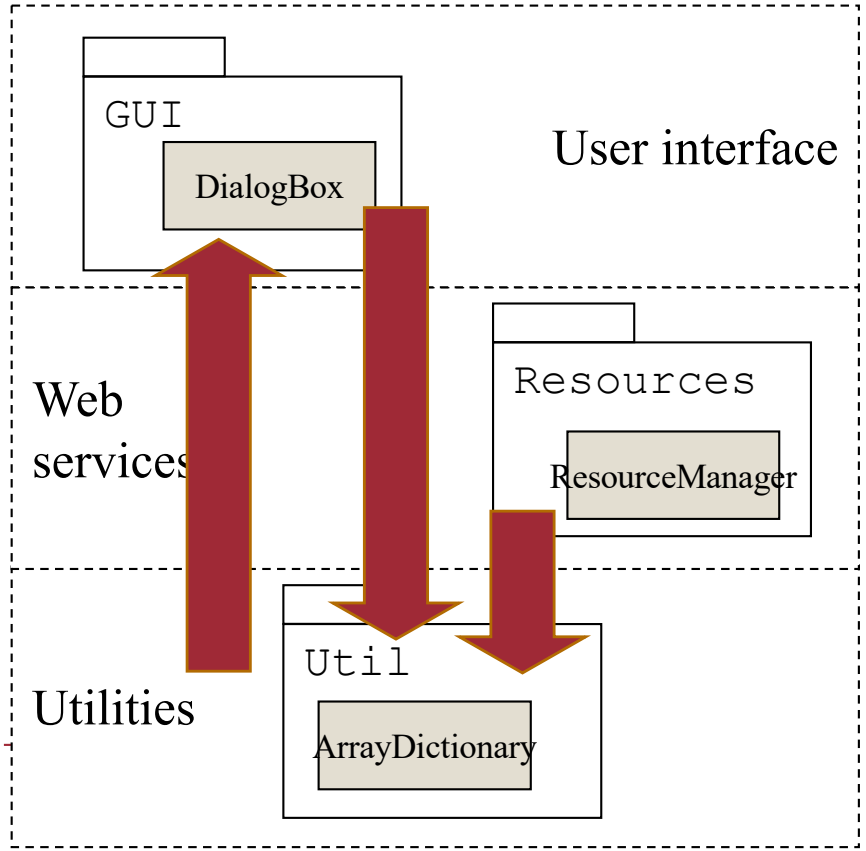
Architectural erosion: example (cont.)

- Change creating an incorrect dependency: From lower to upper layer

```
package org.w3c.util;

public class ArrayDictionary {
  public Object get(Object key){
  if (cannot-find-object)
     // Report error:

org.w3c.GUI.DialogBox.report("error..."
);
  ...
}
```



Outcomes of architectural erosion

- The “architectural model” is incorrect
  - Maintenance is difficult: Programmers cannot find their way in the program
- The Program is “brittle”
  - Breaks with the simplest change
  - The “domino effect”

The Laws of Software Evolution

- **Manny Lehman**, the "Father of Software Evolution", wrote many papers from the mid 70s onwards, proposing "Laws of Software Evolution" for "E-type systems".
- Systems classified into:
  - S-type: formally specified and verified; static by definition
  - **E-type**: real-world system
- "Laws": Reflect the cooperative activity of many individual and organisational behaviours.
  - Influenced by thermodynamics
  - Reflect established observations and empirical evidence
- "An emerging theory of software process and software evolution."
  - i.e., work in progress

Lehman's laws

I. **Continuing Change**
   - An E-type system must be continually adapted else it becomes progressively less satisfactory in use

II. **Increasing Complexity**
   - As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it

III. **Self regulation**
   - Global E-type system evolution processes are self- regulating and supports the ability to react to change. (eg. an E-type systems growth inevitably slows as it grows older)

IV. **Conservation of Organisational Stability**
   - Average activity rate in an E-type process tends to remain constant over system lifetime or segments of that lifetime. Managers have no power?

V. **Conservation of Familiarity**
   - In order for E-type systems to continue to evolve, its maintainers must possess and maintain a mastery of its subject matter and implementation.

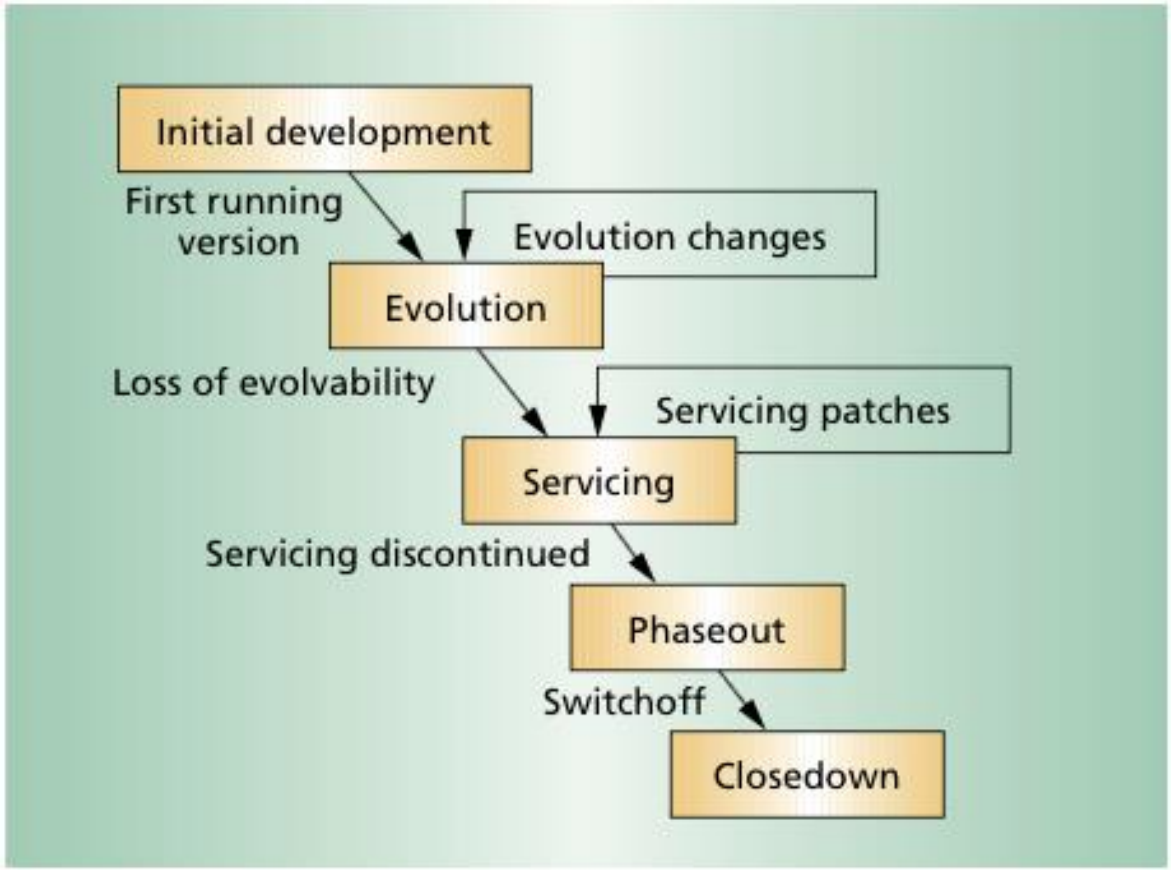
VI. **Continuing Growth**
   - The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over the system lifetime

VII. **Declining Quality**
   - Unless rigorously adapted to take into account changes in the operational environment, the quality of an E-type system will appear to be declining as it is evolved

VIII. **Feedback System**
   - In order to sustain continuous change, or evolution of a system, there must be a means to monitor the performance it will have.

I	Continuing Change	An E-type system must be continually adapted else it becomes progressively less satisfactory in use
II	Increasing Complexity	As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it
III	Self regulation	Global E-type system evolution processes are self-regulating and supports the ability to react to change. (eg. an E-type systems growth inevitably slows as it grows older)
IV	Conservation of Organisational Stability	Average activity rate in an E-type process tends to re-main constant over system lifetime or segments of that lifetime. Managers have no power?
V	Conservation of Familiarity	In order for E-type systems to continue to evolve, its maintainers must possess and maintain a mastery of its subject matter and implementation.
VI	Continuing Growth	The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over the system lifetime
VII	Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of an E-type system will appear to be declining as it is evolved
VIII	Feedback System	In order to sustain continuous change, or evolution of a system, there must be a means to monitor the performance it will have.

Staged model of software evolution*



* Bennett, K. H.; Rajlich, V. T.; Mazrul, R. Mohamad (1995). "Legacy System: Coping with success". *IEEE Software*. pp. 19–23.

Summary

- Maintenance as part of the waterfall process
- Evolutionary models reflect reality better
- Software change is frequent, unexpected, and very expensive
- Problems caused by architectural erosion
- The laws of software evolution
- The staged model of software evolution

Class exercises

Law no. I: *Continuing Change*

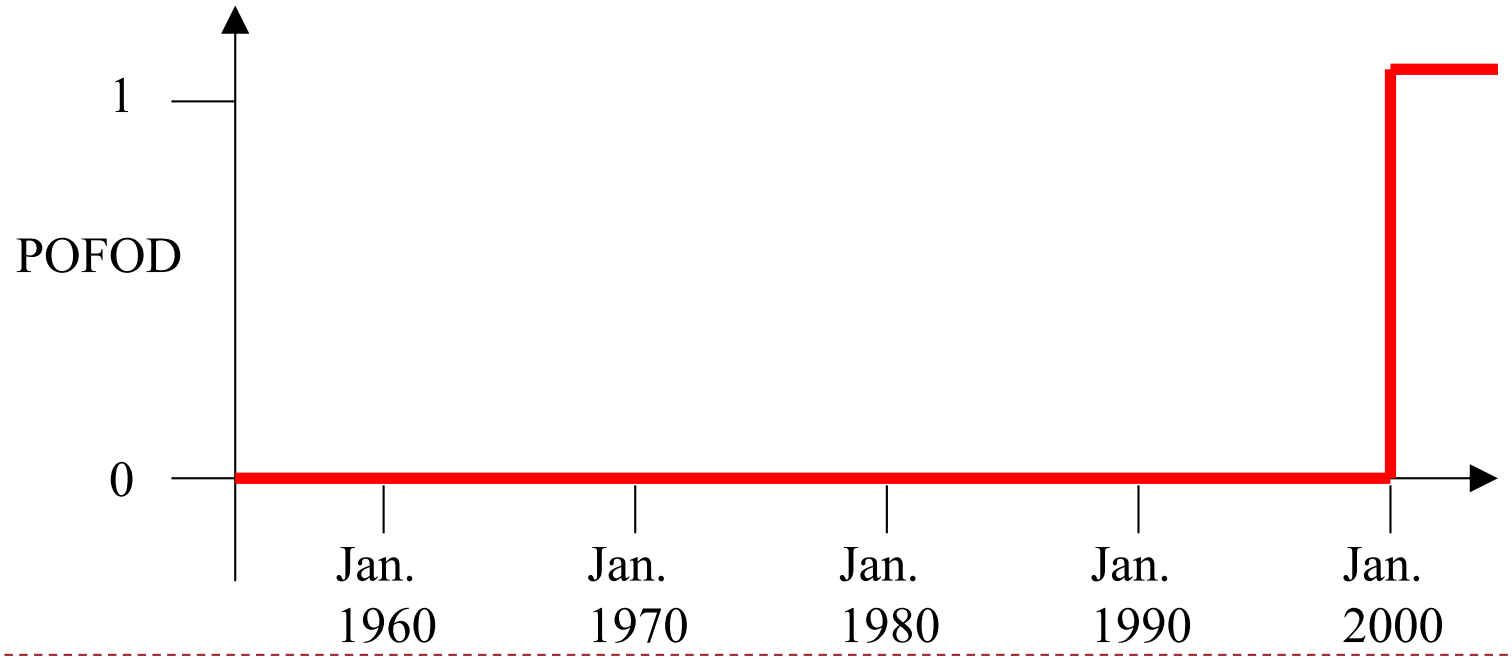
An E-type program that is used must be continually adapted else it becomes progressively less satisfactory

What are the reasons for this?

- Growth of the environment ("operational domain")
- Hence, increasing mismatch between the system and its environment

Exercise: *Continuing Change*

- Sketch the reliability of a program that is not Y2K-compliant as a function of time before and after January 2000:



Law no. II - *Increasing Complexity*

As a program is evolved its complexity increases unless work is done to maintain or reduce it.

What are the reasons for this?

- Small changes are applied in a step-wise process
  - Each 'patch' makes sense locally, not globally
- Effort needed to reconcile accumulated changes with the overall performance goals

Law no. VI: *Continuing Growth*

Functional content of a program must be continually increased to maintain user satisfaction over its lifetime

- Implied by the first law, except with focus on *functional requirements*
- What is the motivation for this?
  - Competition
  - Customer demand
  - User dissatisfaction
  - "Omitted attributes will become the bottlenecks and irritants in usage as the user has to replace automated operation with human intervention. Hence they also lead to demand for change"

Exercise

- Find a solution to the architectural erosion problem in slide 11
  - How can a dialogue box be created without introducing a dependency of class util.ArrayDictionary on class gui.DialogBox?

Bibliography

- M. Lehman. "Laws of Software Evolution Revisited". Lecture Notes in Computer Science 1149 (Proc. 5th European Workshop on Software Process Technology), pp. 108–124. Berlin: Springer-Verlag, 1996.
- F. P. Brooks. The Mythical Man-Month. Reading: Addison-Wesley, 1975.
- D. E. Perry, A. L. Wolf. “Foundation for the Study of Software Architecture”. ACM SIGSOFT Software Engineering Notes, Vol. 17, No. 4 (1992), pp. 40–52.

Bibliography II

- Ch. 32 'Software Maintenance' in: I. Sommerville. "Software Engineering" 7th Edition. Reading: Addison-Wesley, 2004.
- Tom Mens, Lecture Notes in Software Evolution, Université de Mons-Hainaut, Service de Génie Logiciel