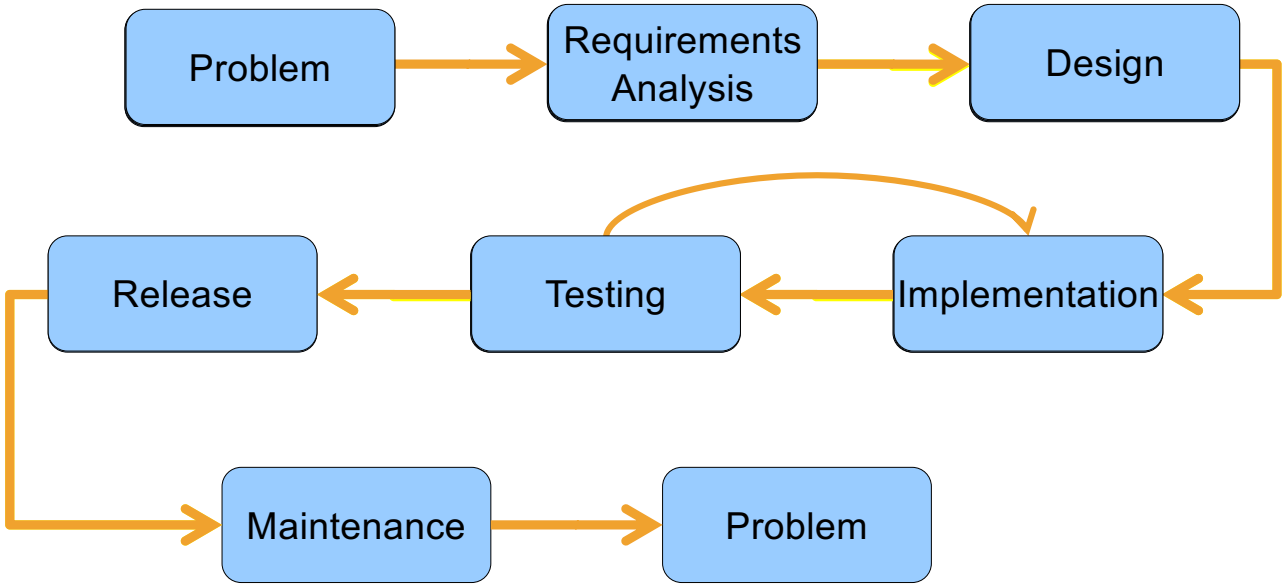
CS 4320 / 7320 Software Engineering

Maintenance

What is the SDLC?
Where does Maintenance fit?



Why Maintenance?

Why isn't *Release* the end?

- Defects are uncovered
- Operating environments change
  - Hardware *and* software
- New user requirements surface

Software Maintenance …

…is the **totality** of activities required to provide **cost-effective support** to software.

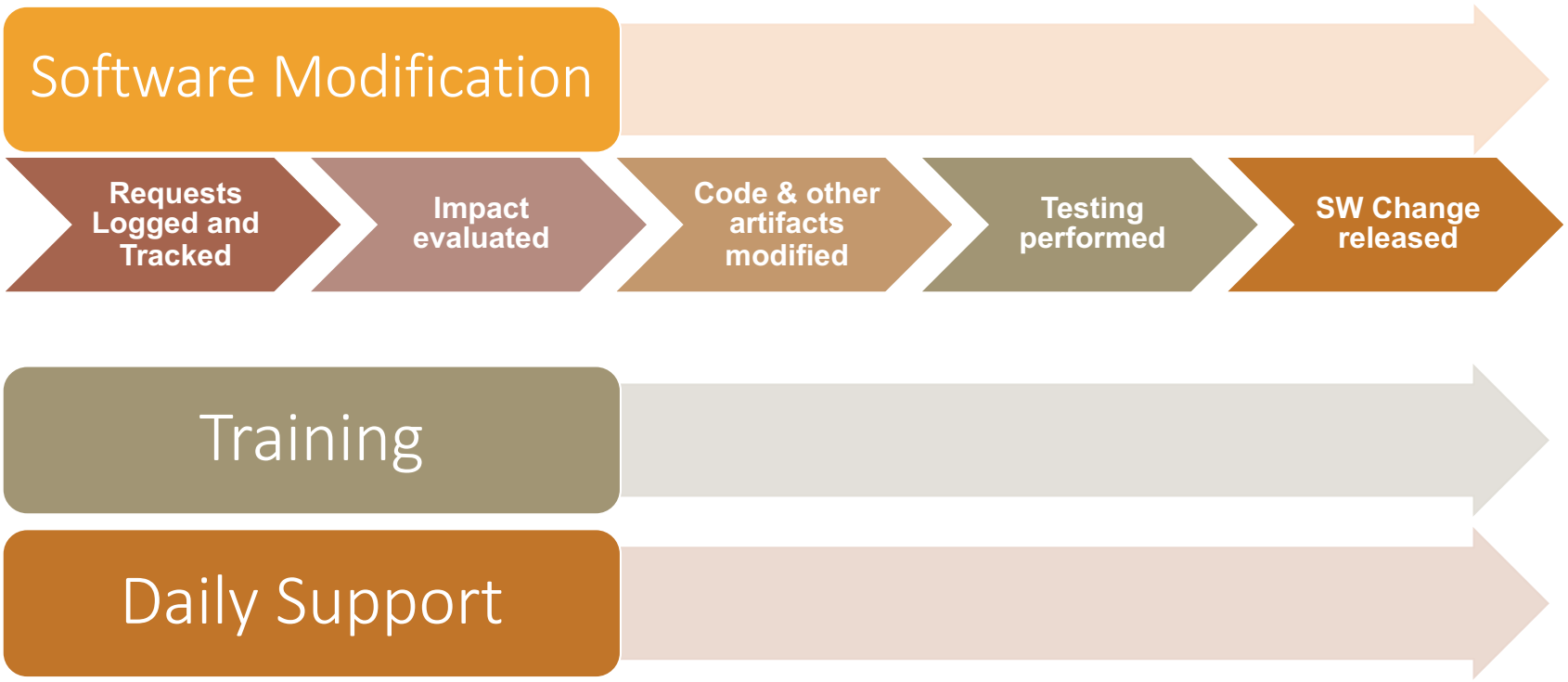
…occurs during:

Predelivery

Planning for operations, maintainability, transition

Postdelivery

Software modification, training, interfacing with help desk


Software Modification
Requests Logged and Tracked
Impact evaluated
Code & other artifacts modified
Testing performed
SW Change released
Training
Daily Support

Software modification process: SLDC+

Requests logged & tracked

Requirements Analysis
& impact analysis
& understanding code
& update req. docs

Design & update design docs

Implement

Testing & update tests

Release

The Objective of Software Maintenance*

*i.e., the software modification part of it

..is to modify existing software
to fix defects,
to adapt to changing the environment,
and to add new features
while preserving software integrity.

Categories of Maintenance

	CORRECTION	ENHANCEMENT
PROACTIVE	Preventative	Perfective
REACTIVE	Corrective	Adaptive

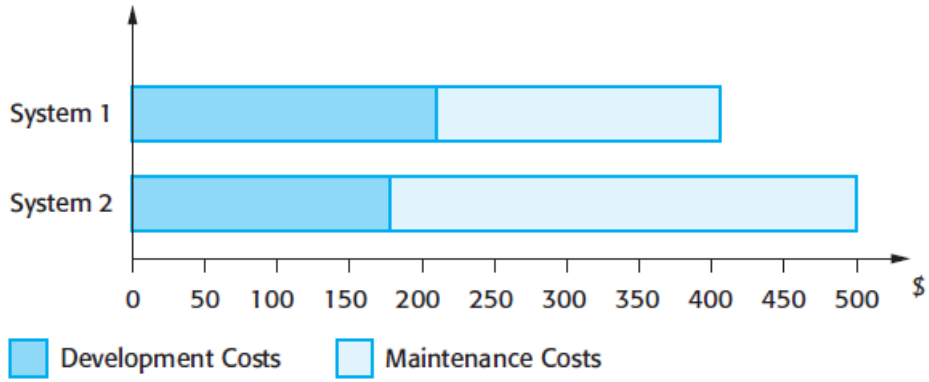
- Bug fixes: Corrective Maintenance
- Fix latent faults before they become operational: Preventative Maintenance
- New Features: Perfective Maintenance
- Adapt to changing environment: Adaptive Maintenance

Maintenance Costs…

Typical: Development 1/3, Maintenance 2/3

Overall lifetime costs may decrease as more effort is expended during system development to produce a maintainable system.

For System 1, extra development costs of $25,000 are invested in making the system more maintainable. This results in a savings of $100,000 in maintenance costs.



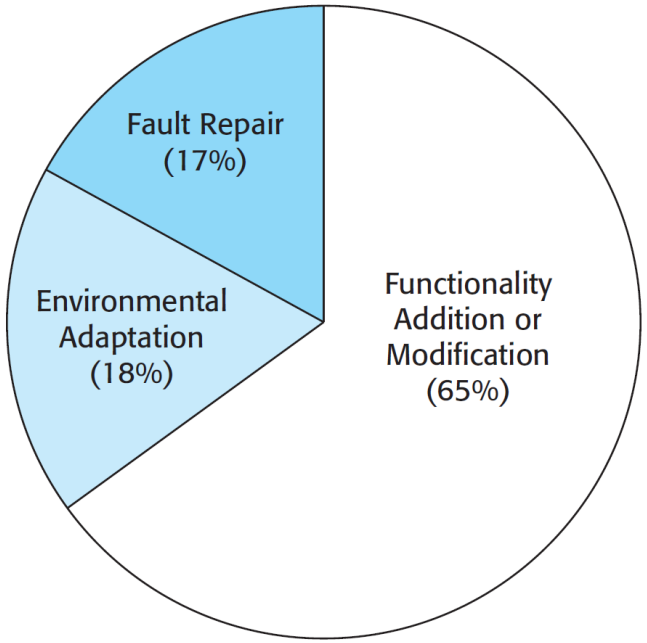
Sommerville, Software Engineering, 9th edition

Software modification costs...

Most cost is NOT in bug fixes

Most management reports don't distinguish

Sommerville, Software Engineering, 9th edition



Software Evolution: Lehman's Laws

Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
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.
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.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Sommerville, Software Engineering, 9th edition

Key Issues in Software Maintenance

Limited Understanding: 1/2 total maintenance effort is understanding the software to be modified. Developers leave. Text (code) is hard to understand. Changes need to be documented or the documentation lies.

Testing: Takes time and money. Regression testing plan and automation help reduce time and cost.

Impact Analysis: Identify all systems affected by a change request. Estimate resources needed. Evaluate risk. Take severity of the problem into account. Plan course of action.

Maintainability: Software modifications need to be designed to maximize maintainability. Often neglected.

Measures of maintainability

Analyzability: measure of effort to understand the issue

Changeability: measure of effort to implement a modification

Stability: measure of unexpected behavior, including during testing

Testability: Measure of effort to test the modified software

Size of the software

Complexity of the software

Understandability

Refactoring is…

The restructuring of code to:

- Increase reliability
- Increase maintainability
- Increase extensibility

Refactoring itself does NOT increase functionality

- It facilitates cleaner evolution

Refactoring - how

Reverse engineer original design decisions

Re-design with understanding of:

- New operational environment
- New functionality requirements

The re-design is usually a high-level abstraction

Refactoring – a few code “smells”

Duplicate code: replace with a single method or function.

Long methods, large classes: low coherence. Replace with smaller coherent methods.

Switch (case) statements: Often have duplication. In OOP can use polymorphism.

Data clumping: groups of data items reoccur. Encapsulate in an object.

Speculative generality: (YAGNI)

Refactoring references

THE book on the subject:

M. Fowler and K. Beck, Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, 1999.

Recommended website:

Contains summary of all refactorings in the book.

https://refactoring.com/