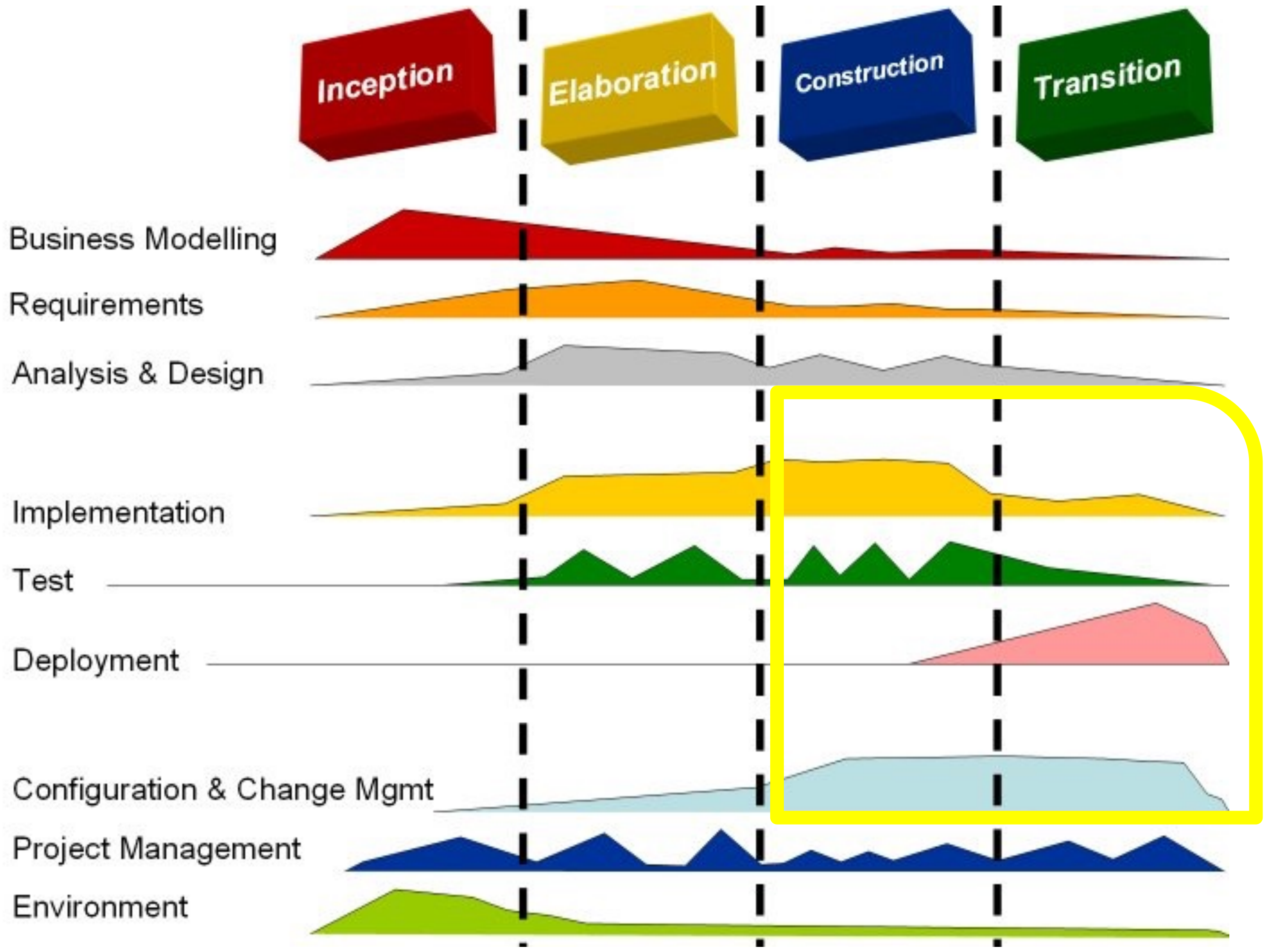
Software Engineering

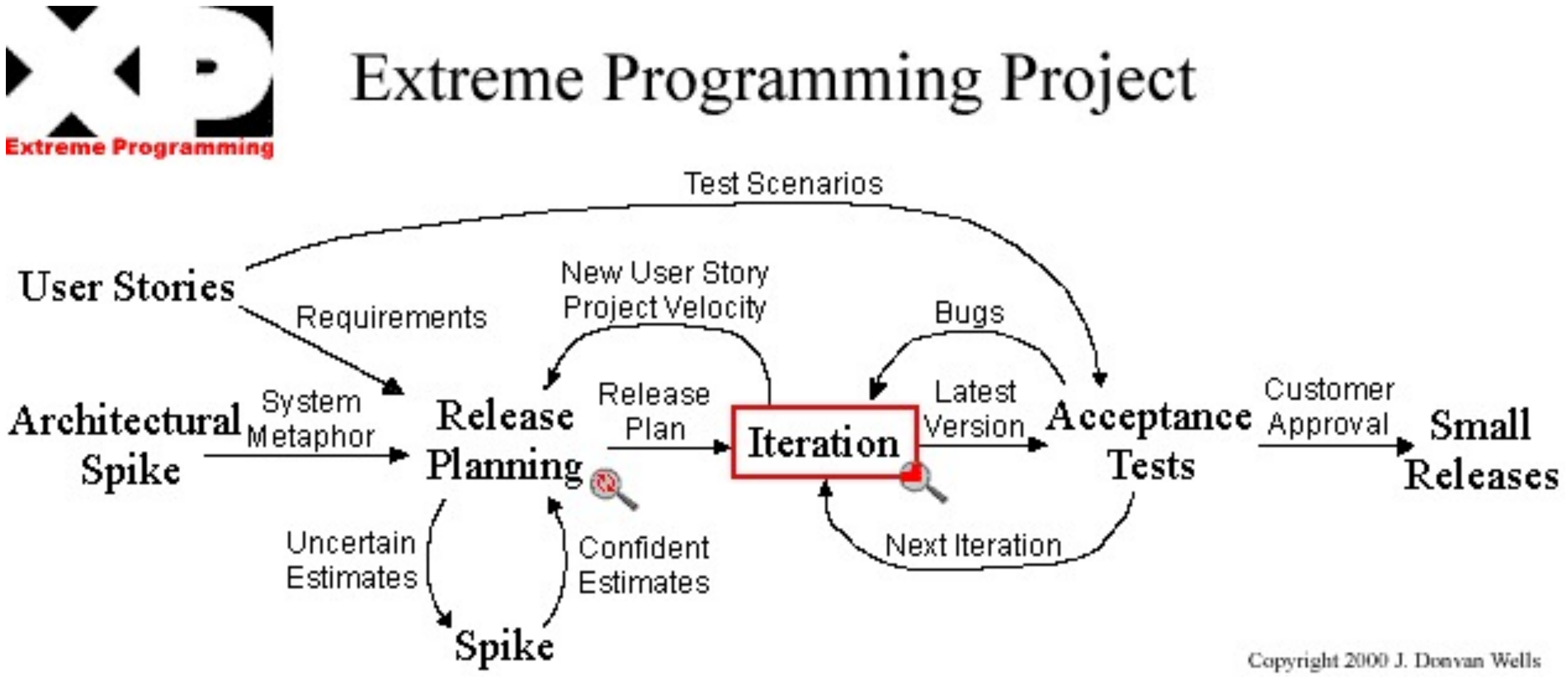
L.EIC-ES-2021-22

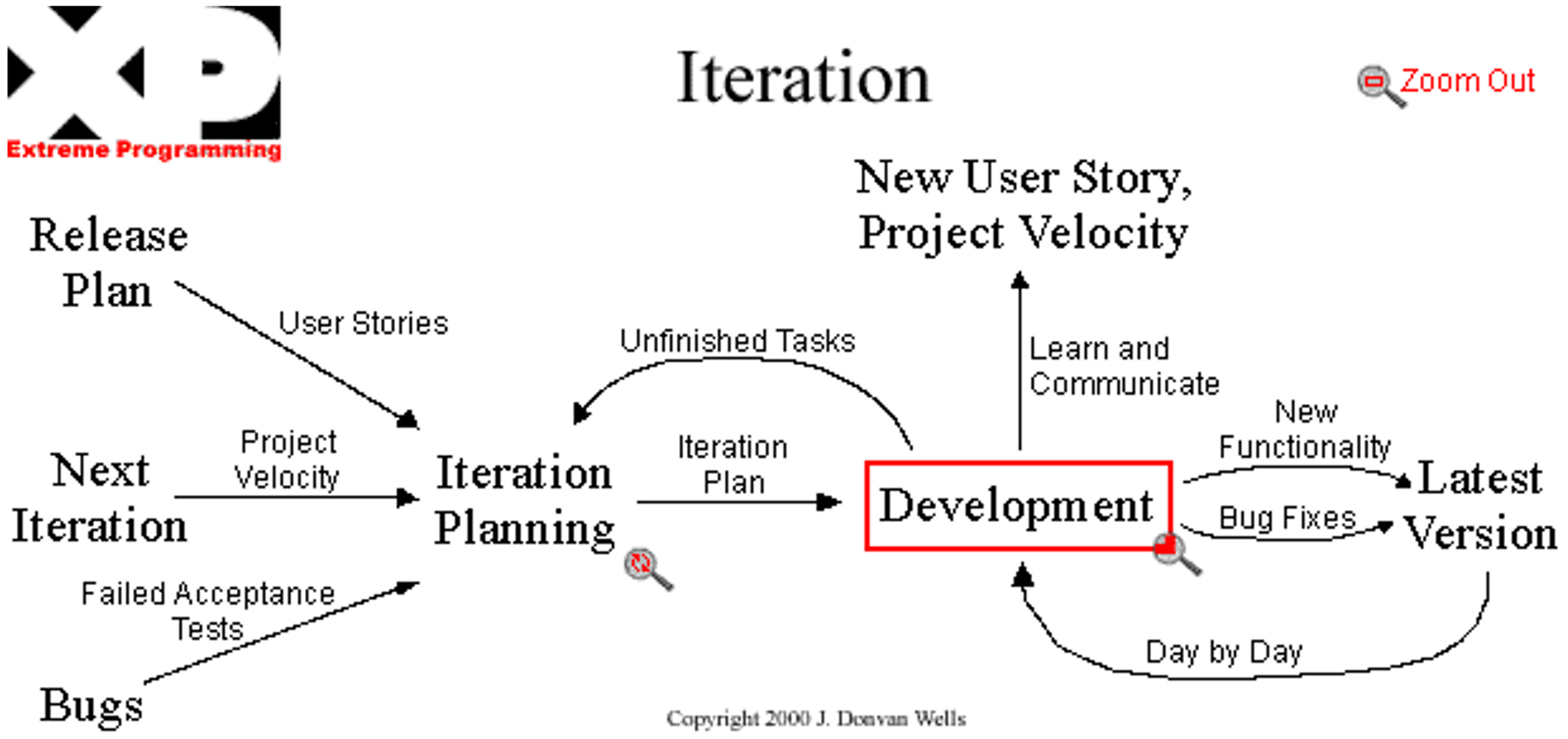
Ademar Aguiar, J. Pascoal Faria

U. PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Rational Unified Process




Extreme Programming
Extreme Programming Project
Test Scenarios
User Stories
Requirements
New User Story
Project Velocity
Bugs
Architectural Spike
System Metaphor
Release Planning
Release Plan
Iteration
Latest Version
Acceptance Tests
Customer Approval
Small Releases
Uncertain Estimates
Confident Estimates
Next Iteration
Spike
Copyright 2000 J. Donvan Wells



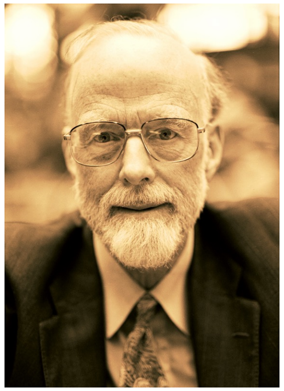
Release planning: Release content & date

Simplicity

- "Do the simplest thing that could possibly work" (DTSTTCPW) principle
  - Elsewhere known as KISS (Keep It Simple, Stupid)
- A coach may say DTSTTCPW when he sees an XP developer doing something needlessly complicated
- "You ain't gonna need it'' (YAGNI) principle
- Simplicity and communication support each other (how?)

Simple Design

- Use the simplest possible design that gets the job done
- The requirements will change tomorrow, so only do what's needed to meet today's requirements (remember YAGNI)
- Avoid big up-front design


Tony Hoare, Turing Award Lecture, 1980

I conclude there are two ways of constructing a software design:

one way is to make it so simple that there are obviously no deficiencies,

and the other way is to make it so complicated that there are no obvious deficiencies.

Pair Programming

- How it works:
  - Two programmers work together at one machine
  - Driver enters code, while navigator critiques it
  - Periodically switch roles and pairs
  - Requires proximity in lab or work environment
- Advantages:
  - Serves as an informal review process
  - Helps developing collective ownership and spread knowledge
  - Improves quality (less defects, better design), whilst maintaining (or improving) productivity
- Note: If you don’t do XP/PP, at least do peer reviews

Refactoring aka Design Improvement

- Refactoring = improve the structure of the code without changing externally visible behavior
- e.g., refactor out any duplicate code generated in a coding session
- Refactoring and automated tests go hand-in-hand (why?)
- Simple design and continuous design improvement (with refactoring) go hand in hand (why?)

Test-driven Development

- Test first: before adding a feature, write a test for it!
  - If code has no automated tests, it’s assumed it doesn’t work
  - Also create test for bugs discovered, before fixing them
- Unit Tests (or developer tests): testing of small pieces of functionality as developers write them
  - Usually for testing single methods, classes or scenarios
  - Usually automated with a unit testing framework
  - Experiments show that TDD reduces debugging time
- Acceptance Tests (or customer tests): specified by the customer to check overall system functioning
  - A user story is complete when all its acceptance tests pass
  - Usually specified as scripts of UI actions & expected results
  - Ideally automated with a UT or AT framework

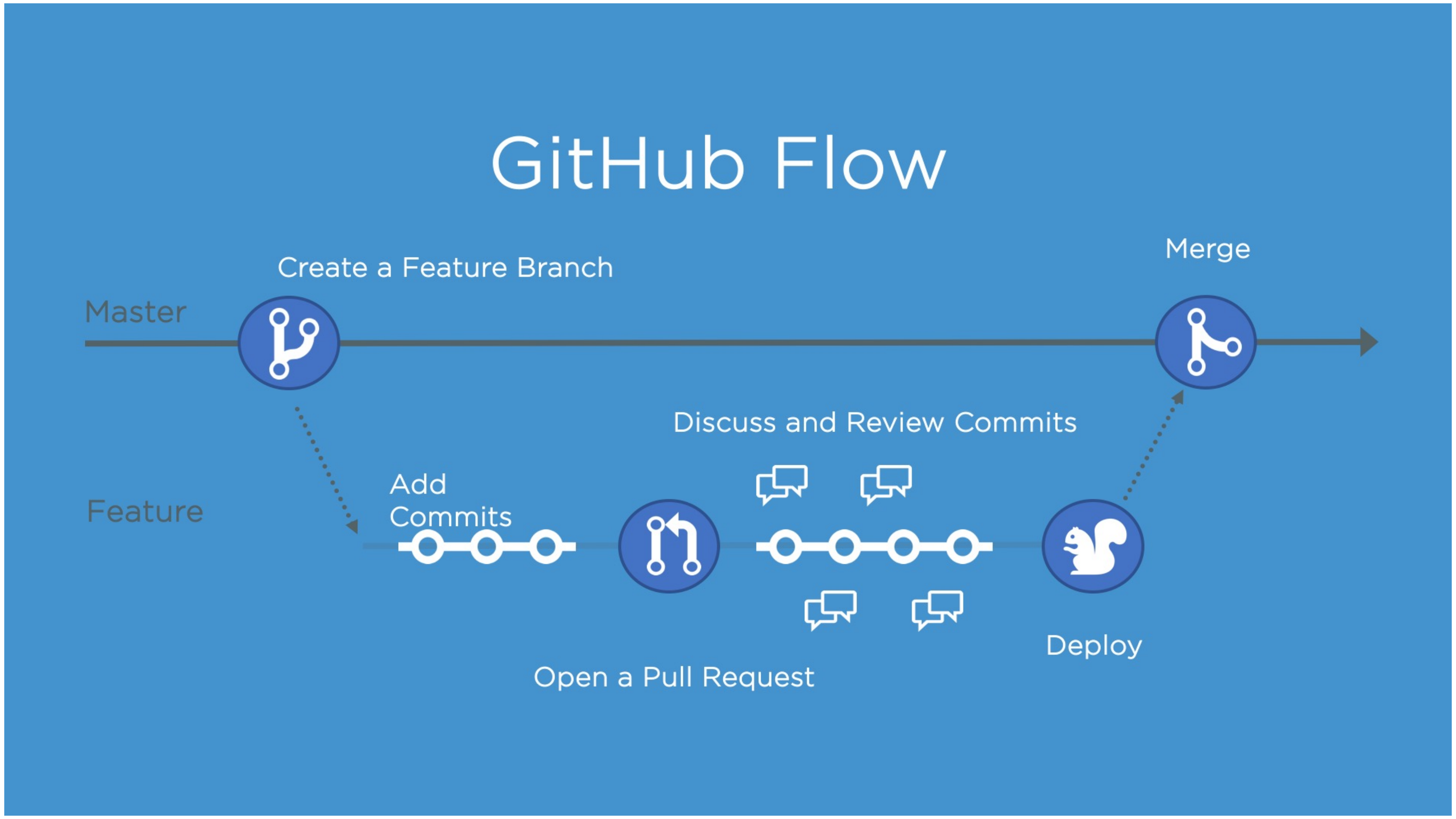
Collective Code Ownership

- What it means:
  - No single person "owns" a module
  - Any developer can work on any part of the code base at any time
- Advantages:
  - No islands of expertise develop
  - All the developers take responsibility for all of the code
  - Pressure to create better quality code
  - Change of team members is less of a problem

Continuous Integration

- All changes are integrated into the code base at least daily, as opposed to “big-bang integration”
- Tests have to run 100% before & after integration
- Enables frequent releases

GitHub Flow


GitHub Flow
Create a Feature Branch
Merge
Master
Discuss and Review Commits
Add Commits
Feature
Deploy
Open a Pull Request

Software Evolution

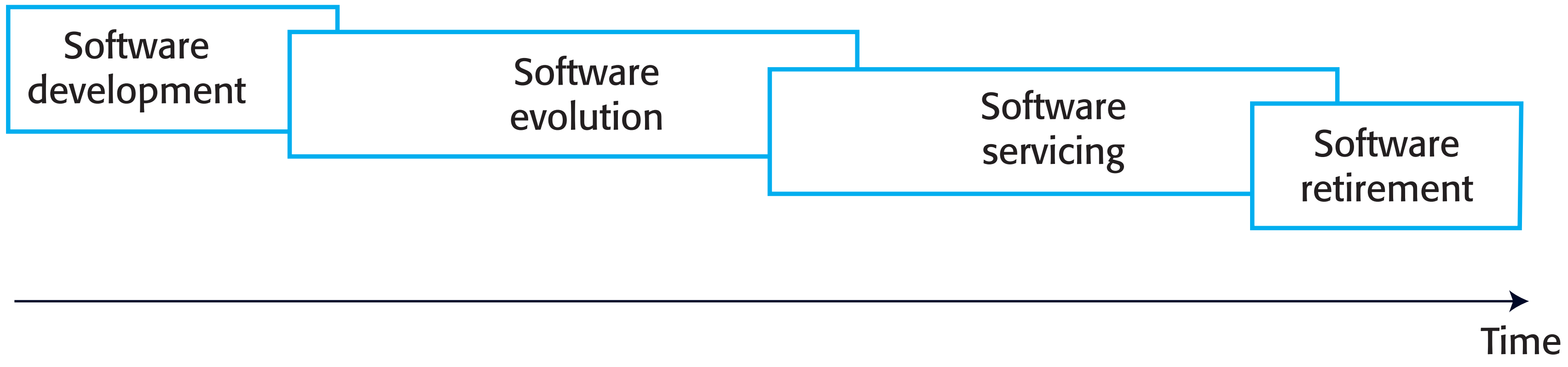
Software change

- Software change is inevitable
  - New requirements emerge when the software is used;
  - The business environment changes;
  - Errors must be repaired;
  - New computers and equipment is added to the system;
  - The performance or reliability of the system may have to be improved.
- A key problem for all organizations is implementing and managing change to their existing software systems.

Importance of evolution

- Organisations have huge investments in their software systems - they are critical business assets.
- To maintain the value of these assets to the business, they must be changed and updated.
- The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

Evolution and servicing



Evolution and servicing

- Evolution
  - The stage in a software system’s life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.
- Servicing
  - At this stage, the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software’s environment. No new functionality is added.
- Phase-out
  - The software may still be used but no further changes are made to it.

Agile methods and evolution

- Agile methods are based on incremental development so the transition from development to evolution is a seamless one.
- Evolution is simply a continuation of the development process based on frequent system releases.
- Automated regression testing is particularly valuable when changes are made to a system.
- Changes may be expressed as additional user stories.

Software maintenance

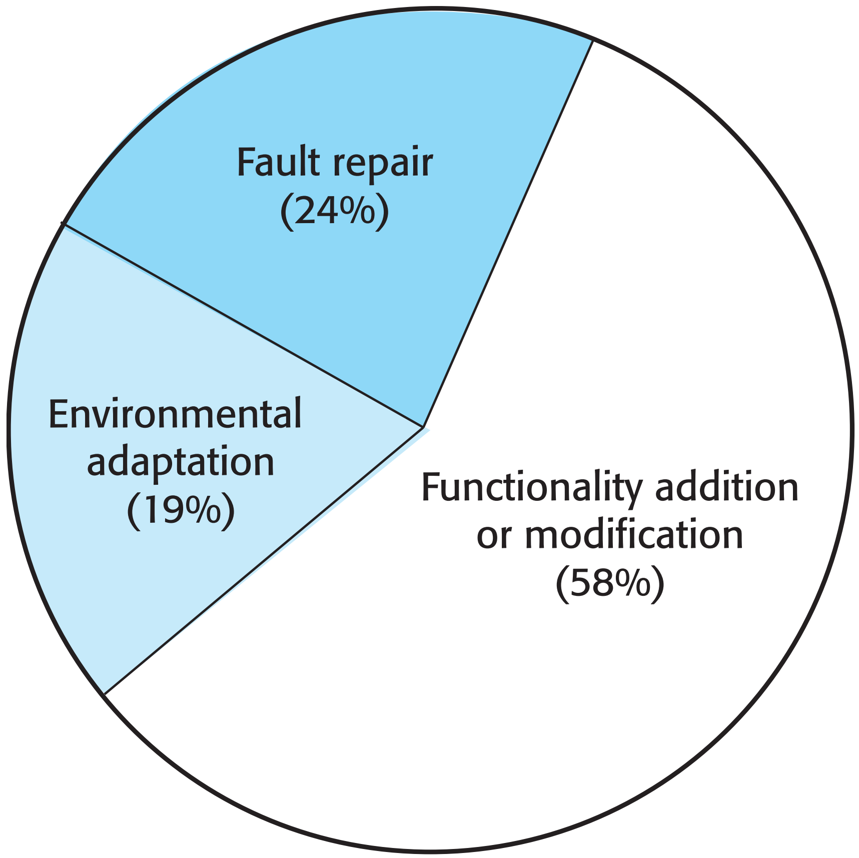
Software maintenance

- Modifying a program after it has been put into use.
- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- Maintenance does not normally involve major changes to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

Types of maintenance

- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
- Environmental adaptation
  - Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Functionality addition and modification
  - Modifying the system to satisfy new requirements.

Maintenance effort distribution



Key points

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- For custom systems, the costs of software maintenance usually exceed the software development costs.
- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.
- It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.
- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.
- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
- Refactoring, making program changes that preserve functionality, is a form of preventative maintenance.

ademar.aguiar@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO