Testing of Object-Oriented Software

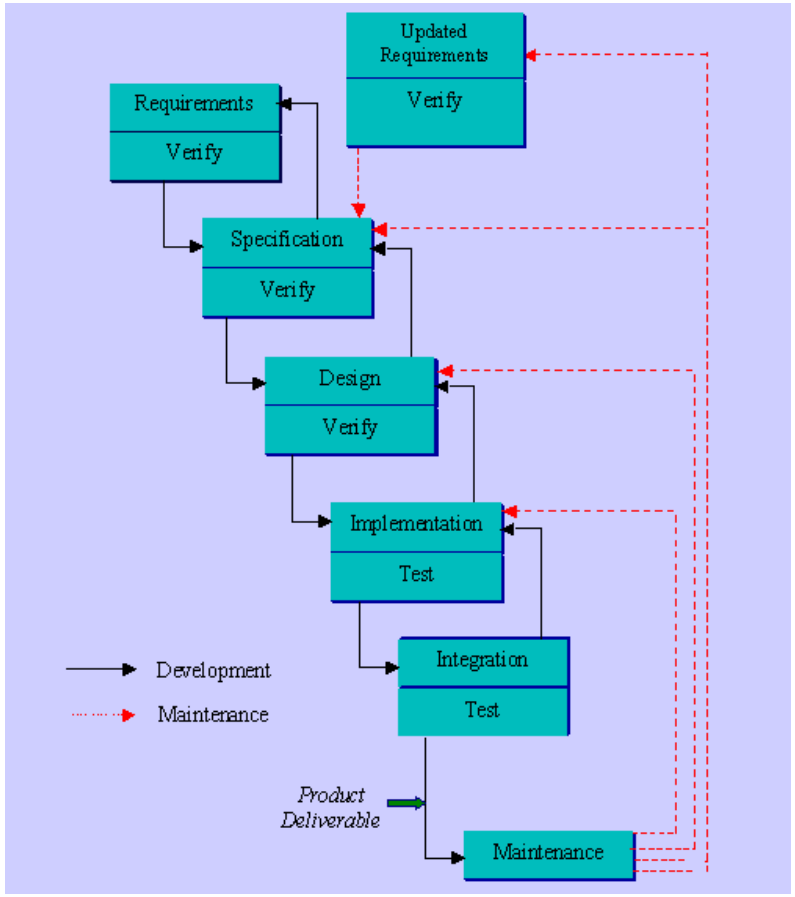
Hans Vangheluwe

Modelling, Simulation and Design Lab (MSDL)
School of Computer Science, McGill University, Montréal, Canada

Overview

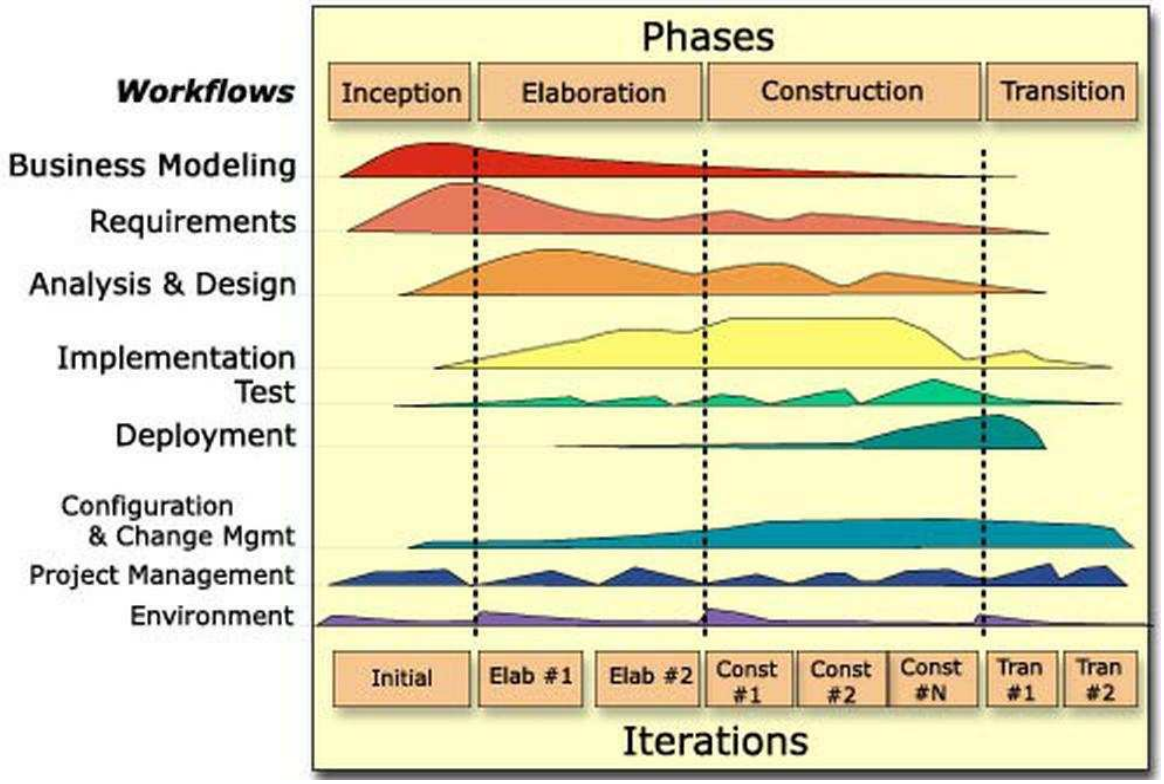
1. Testing background
   - Testing as part of the Software **Process**
   - **Terminology**
   - **Types** of Testing
2. **Test-Driven** development: Unit Testing
   - Test for **Success/Failure/Sanity**
   - The `Roman` **example** from
     Mark Pilgrim's "Dive into Python"
     `http://diveintopython.org/`
   - in Python using the `pyunit` Unit Testing Framework

Testing as part of the Software Process: Waterfall



http://scitec.uwichill.edu.bb/cmp/online/cs22l/waterfall_model.htm

Testing as part of the Software Process: RUP

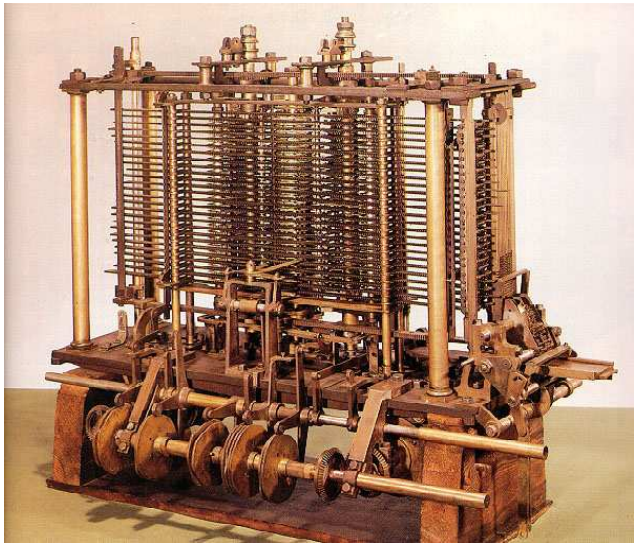


Test-centric (test first), Iterative, Regression Testing

Terminology

- **Validation**: a process designed to increase our **confidence** that a program satisfies the **requirements** it implements.
  No certainty (cfr. Carl Popper)!
- **Verification**: a formal or informal argument that a program works (correctly) on **all** possible inputs.
- **Testing**: a process of running a program on a **limited set** of inputs and comparing the actual results with expected results (from the requirements).
- **Test Oracle**: a **source of expected results** for a test case.
- **Debugging**: a process designed to determine **why** a program is not working correctly.
- **Defensive programming**: the practice of writing a program in a way designed specifically to **avoid** making errors and to **ease** validation and debugging.

Errors in Software?



The concept that *software might contain errors* dates back to 1842 in Ada Byron's notes on the analytical engine in which she speaks of the difficulty of preparing program 'cards' for Charles Babbage's Analytical engine: *. . . an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders.*

Software "bug" ?

Myth: *In 1946, when Grace Hopper was released from active duty, she joined the Harvard Faculty at the Computation Laboratory where she continued her work on the Mark II and Mark III. Operators traced an error in the Mark II to a* moth *trapped in a relay, coining the term bug. This bug was carefully removed and taped to the log book September 9th 1945. Stemming from the first bug, today we call errors or glitch's in a program a bug.*

Software "bug" ?

Usage of the term "bug" to describe inexplicable defects has been a part of **engineering jargon** for many decades and predates computers and computer software; it may have originally been used in hardware engineering to describe mechanical malfunctions. For instance, Thomas Edison wrote the following words in a letter to an associate in 1878:

It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arisethis thing gives out and it is then that "Bugs"-as such little faults and difficulties are called-show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.

Designing Test Cases

Exhaustive testing is usually impossible:
A program with three inputs ranging from 1 to 1000
would take 1 000 000 000 test cases !
With a speed of 1 test per second, it would take 31 years.

How to define a **limited** set of good test cases ?

- **Black-box** testing : testing from specification without knowing or taking into account implementation or internal structure.
- **Glass-box** testing : augments black-box testing by looking at the implementation.

Black-box testing

- Advantages:
  - not influenced by assumptions about implementation details
  - robust with respect to changes in implementation
  - allows observers with no internal knowledge of the program to interpret the results of the test
- Disadvantages:
  - unlikely to test ("cover") all parts of a program

Reducing the number of cases to test . . . intelligently

- A program should test typical input values:
  - Arrays or sets are not empty.
  - Integers are between smallest and largest values.
- Boundary conditions usually reveal:
  - Logical errors where the path to a special case is absent.
  - Conditions which cause the underlying hardware or system to raise an exception (e.g., arithmetic overflow).
- Test data should cover all combinations of largest and smallest values:
  - Arrays of 0 and 1 element
  - Empty strings and strings of one character

Glass-box/White-box testing

Glass-box tests complement Black-box testing by adding a test for **each possible path** through the program's implementation.

Good (Unit) Testing:

A test case should be able to:

- run completely by itself, without any human *input*. Unit testing is about **automation**.
- determine by itself whether the function it is testing has **passed or failed**, without a human *interpreting* the results.
- run in isolation, **separate from any other test cases** (even if they test the same functions).
  $\Rightarrow$ Determine *cause* uniquely!

Types of tests:

- test for **Success**
- test for **Failure**
- test for **Sanity**

Illustration: Roman Numerals conversion

`http://diveintopython.org/roman_divein.html`