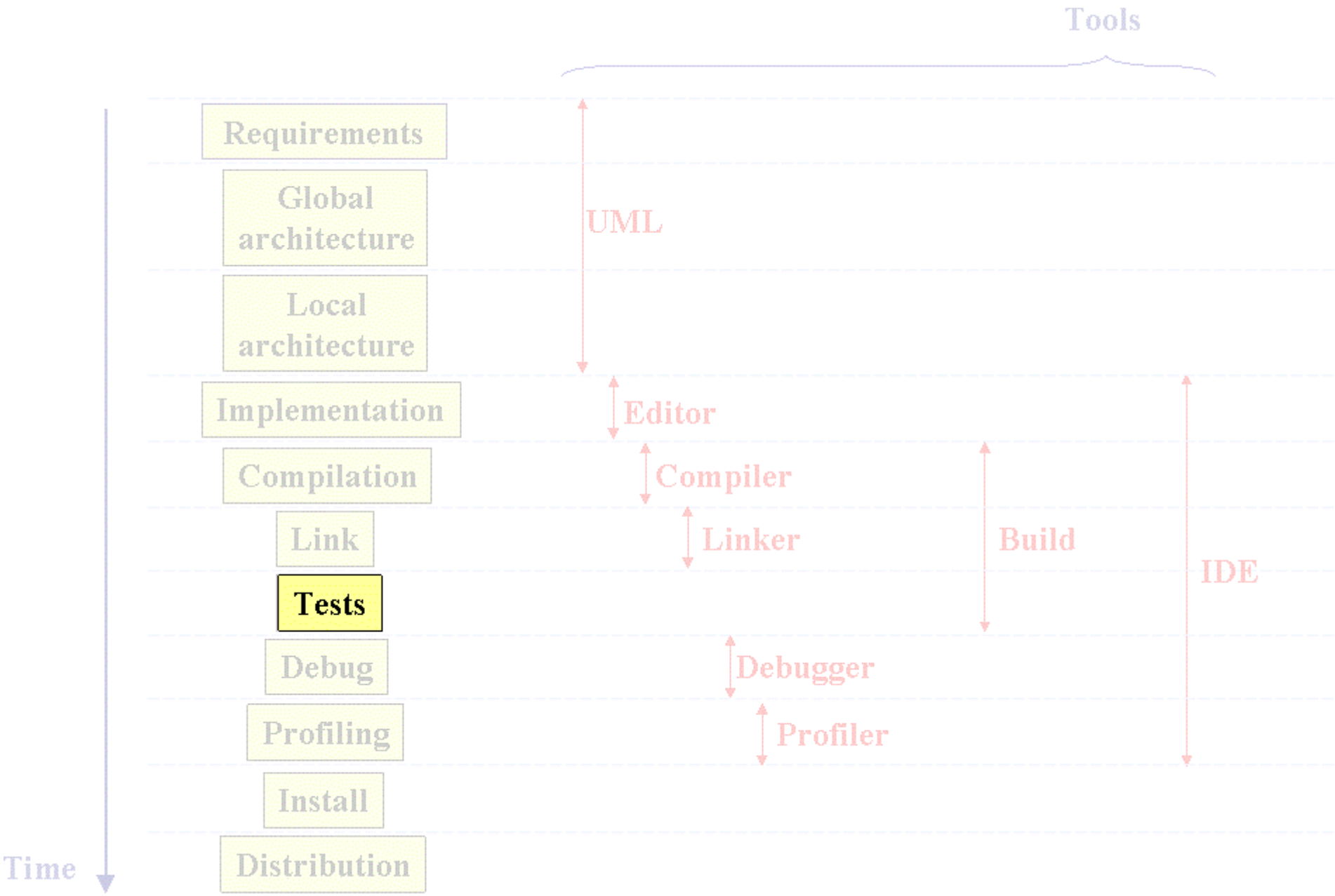
Testing

Mathieu Lacage - DREAM

INRIA
SOPHIA ANTIPOLIS


Tools
Requirements
Global architecture
Local architecture
Implementation
Compilation
Link
Tests
Debug
Profiling
Install
Distribution
Time
UML
Editor
Compiler
Linker
Build
IDE
Debugger
Profiler

Outline

- What is testing ?
- The need for testing
- Different approaches to testing
- What really matters

What is testing ?

- A component or function: takes input, generates output.
- Test:
  - For each input, verify generated output against expected output.
  - For sequences of input, verify generated output against expected output.

The need for testing

- Do you want your users to be beta testers ?
  - Make sure released code is not too buggy

- New features: break existing features
  - Regression tests automatically verify that existing features are not broken
  - It is thus faster to implement a new feature

Different types of testing

- Unit testing vs system testing:
  - Unit testing: test independent components separately.
  - System testing: test the system as a whole
- Regression testing:
  - Test existing functionality: detect regressions
  - New functionality = new test
  - New bug detected = new test to expose the bug

Different types of testing

- Black box vs white box testing:
  - Black box: use only the public API, do not assume anything about implementation.
  - White box: can use private API, can assume it knows the details of the implementation
- White box:
  - often easier to write
  - catches a lot of early bugs because you know where the implementation is weak

Code coverage analysis

- Code coverage analysis tools can be very useful for test case writing: they can tell you which pieces of code are not exercised by the tests.
- Then, you know what kind of tests you need to write to cover a larger set of your software
- See: “gcov” and “lcov”:
  - Gcov: gcc manual
  - http://ltp.sourceforge.net/coverage/lcov.php
    Sample output:
    http://www-sop.inria.fr/dream/personnel/Mathieu.Lacage/ns3-lcov/

Test automation

- Goal: run the tests as often as possible.
  - Tests must be easy to run
  - Test reports must be easy to visualize

- Tools:
  - Use the tool used in your project.
  - Simple solution:
    - A single test function for each unit
    - A global function: invokes all test functions, print FAIL and PASS strings.
    - Invoke global test function often (ideally, each run of the program).
  - Fancy test automation and report generation tools: Junit and CppUnit.

Conclusion

- Prefer white-box testing to black-box testing
- Test boundary conditions first (the common case is tested by the rest of the code)
- Test the easy stuff first:
  - Input combinations can be easily generated
  - Number of input combinations is small
- Use system or unit testing depending on which is the easiest to get started with
- Start with simple test automation, use more fancy tools later if you need them.