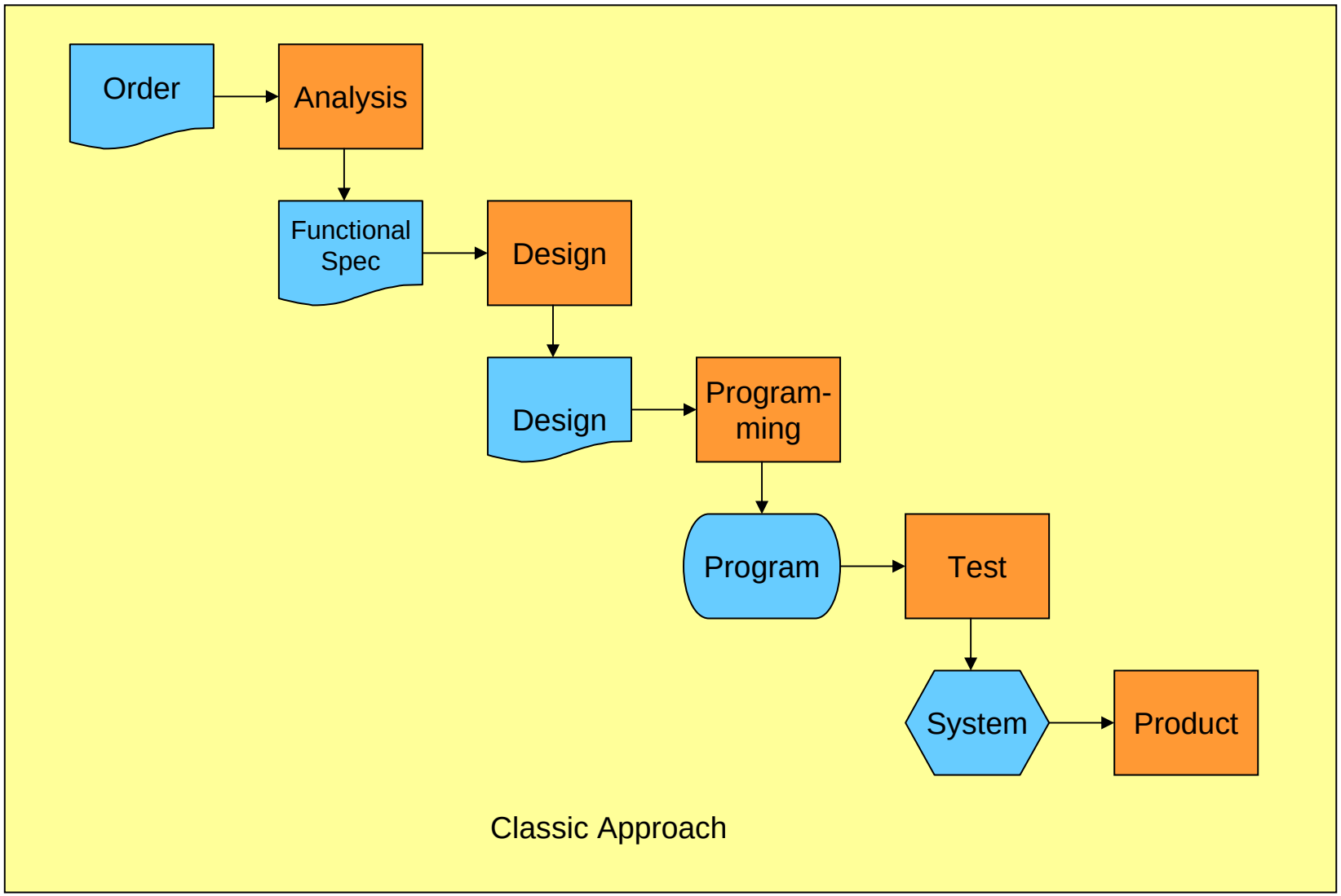
The Drawbacks of Model driven Software Evolution

A Contribution from
Harry Sneed
ANECON GmbH, Vienna
for the CSMR-2007 Special Session
Model driven Software Evolution
Amsterdam, March 2007

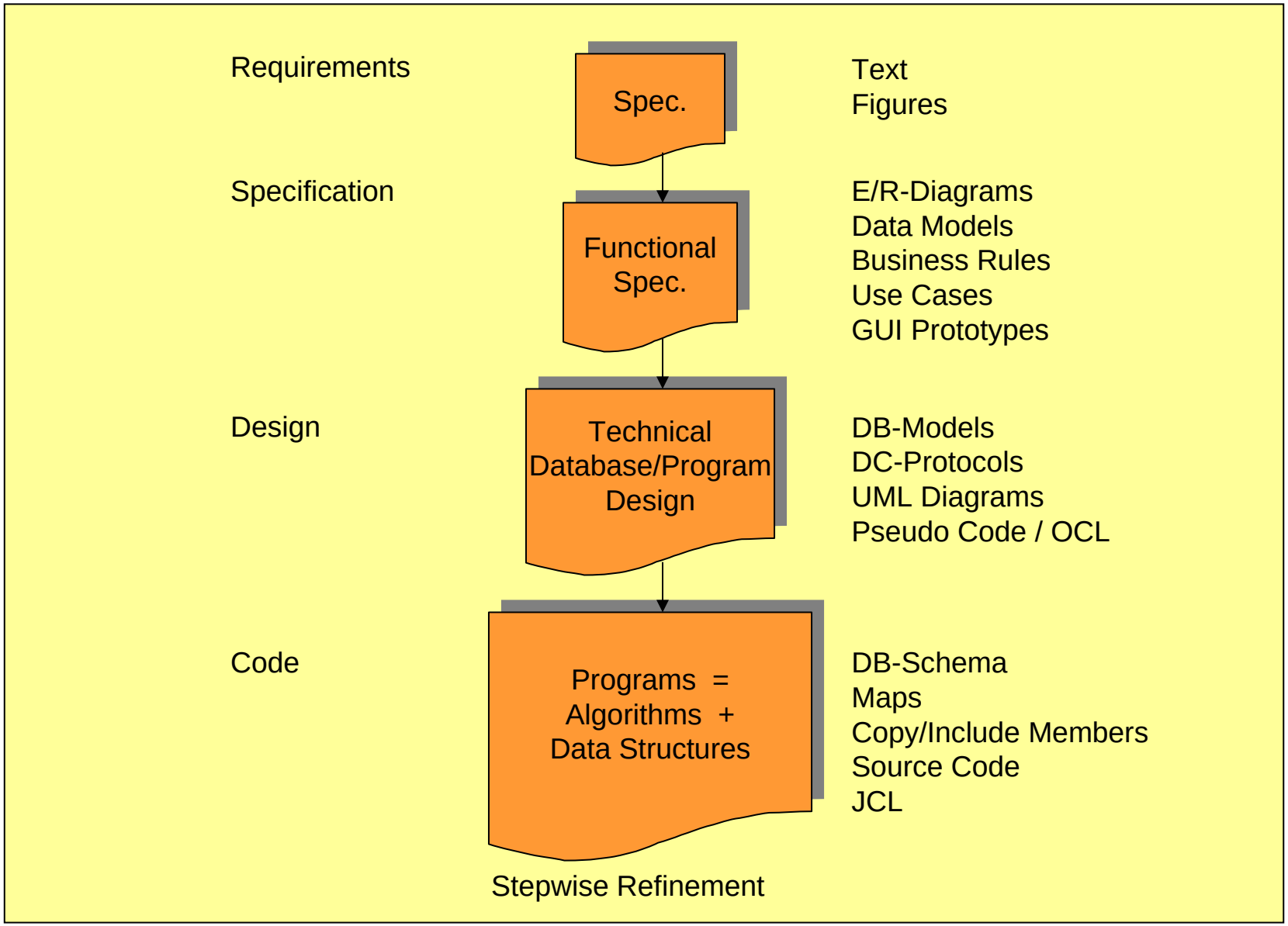
Software System Development according to the V-Model

Development Activities	Development Results
• Requirement Analysis	• Requirement Specs
• System Design	• System Architecture
• Component Design	• Component Specs
• Programming	• Code
• Unit Test	• Unit Test Cases
• Integration Test	• Integration Test Cases
• System Test	• System Test Cases

The Waterfall Model



TOP-DOWN Step wise Development



Criticism of the Top-Down-Approach

- Mc Cracken/Jackson: Life-Cycle Concept Considered Harmful.
  „System requirement can never be stated fully in advance, not even in principle, because the user doesn`t know them in advance - not even in principle“.

- Gladden: Conventional Life-Cycle Approach Exacerbates Maintenance Problem.
  „Each modification to the requirements adversely effects the system by impacting each subsequent task ... the result is a vicious cycle compounding the maintenance problem ... requirements are always incomplete when development begins“.

- Balzer: Specification and Implementation are Intertwined.
  „In actual practice developments steps are not refinements of the original specification, but instead redefine the specification itself ... there is a much more intertwined relationship between specification and implementation than the standard rhetoric would have us believe“.

- Rich/Waters: „Writing a complete specification in a general-purpose specification language is seldom easier and often incredibly harder than writing a program. Furthermore, there has been little success in developing automatic systems that compile efficient programs form specifications“.

Test of the TOP-DOWN Method

- Demillo, Perles, Lipton: „If a formal program is transformed from an informal specification then the transformation process itself must necessarily be informal ... in the end, the program itself is the only complete and accurate description of what the program will do“.
- Fetzer: „From a methodological point of view programs are mere conjectures and testing is an attempted and all too frequent successful refutation ... reasoning about programs tends to be non demonstrative, implicative and non additive ...“.
- Perles: „People must plunge into activities that they do not understand and people cannot create perfect mechanism...“.
- Sneed: „The only way to make the specification a complete and accurate description of the program is to reduce it to the semantic level of the program. However, in so doing, the specification becomes semantically equivalent to the program“.

Model driven Development, i.e. Maintenance, brings up the same issues as with CASE in the 1980‘s

Assumption: „If I can do all the front-end analysis and design with a CASE tool then pump it into a code generator that spits out code, the software problem is solved.“

Refutation: „All programs are written twice, once for the garbage can and once for the computer“
(Donald Knuth)

Problem: „Production and testing is a multi step process with CASE, first you do the design, then the program generation, then the compilation, then a link edit, and then you test. If an error occurs, it occurs in the program and not in the design. To correct it, you have to start over again from the top.“
(Adam Rin, Father of CA-IDEAL 4GL)

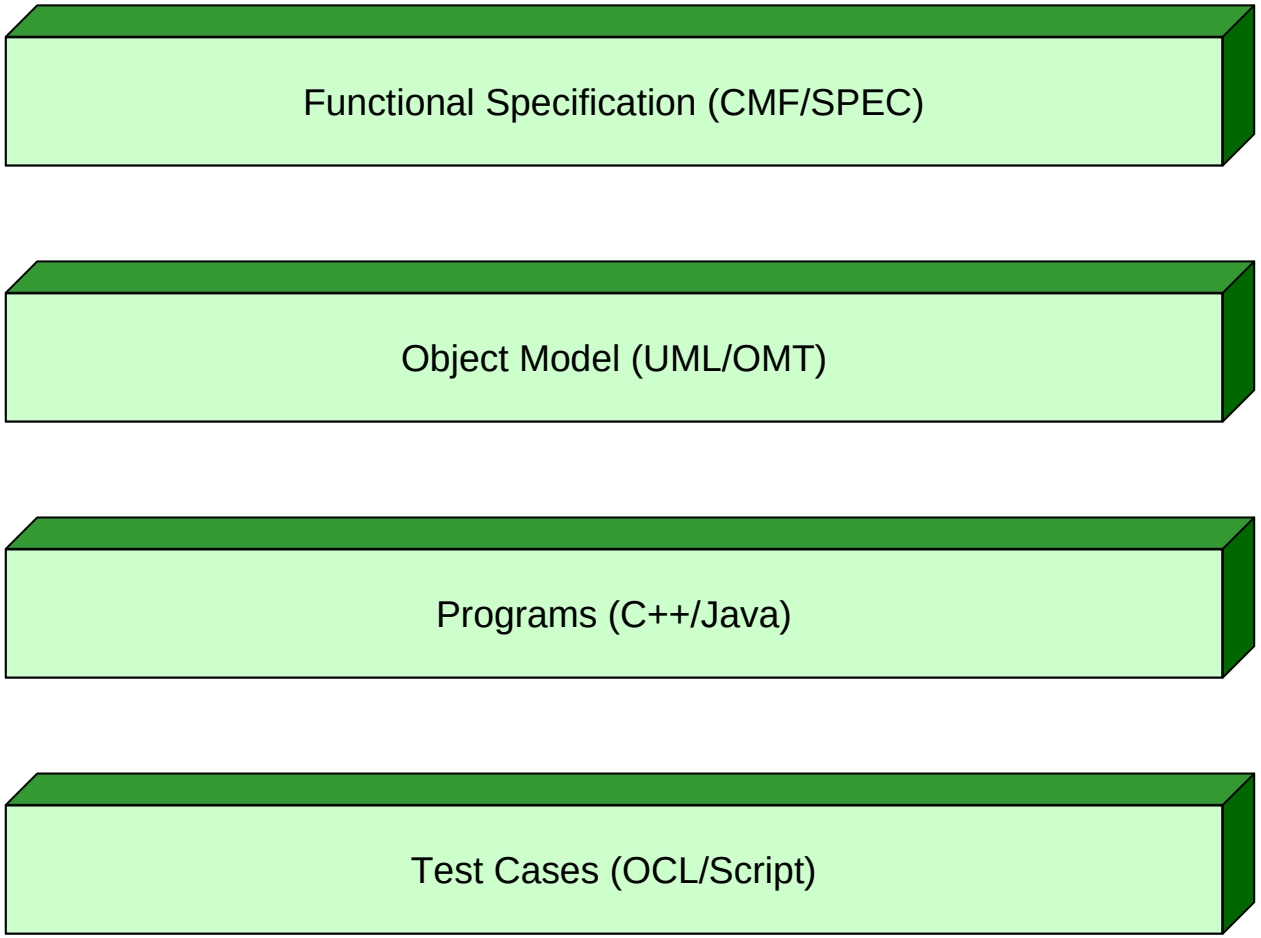
Model driven considered harmful

- Model-driven tools magnify the mistakes made in the problem definition
- Model-driven tools create an additional semantic level to be maintained
- Model-driven tools distort the image of what the program is really like
- The model cannot be directly executed. It must first be transformed into code which may behave other than expected.
- Model driven tools complicate the maintenance process by creating redundant descriptions which have to be maintained in parallel
- Model driven tools are designed for top-down development.
- Top-down functional decomposition creates maintenance problems

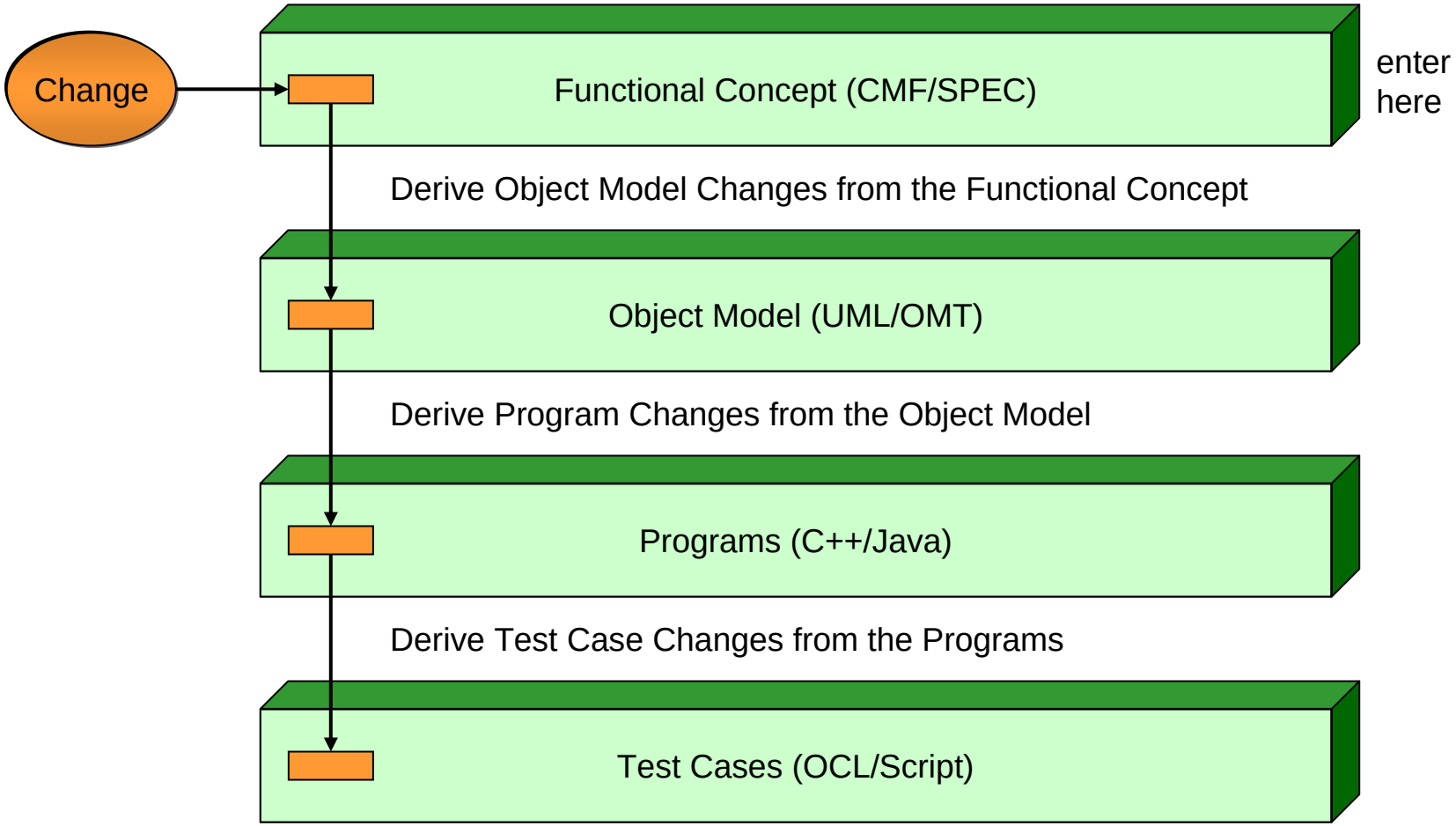
What should be maintained ?

- Most IT-users only maintain the code, the other semantic levels are soon obsolete.
- More progressive users also maintain the test cases and some even maintain the requirements, but hardly any maintain the design. Reverse Engineering was introduced to reproduce the design from code.
- Model driven Evolution would have them maintain the design and regenerate the code for each new release.

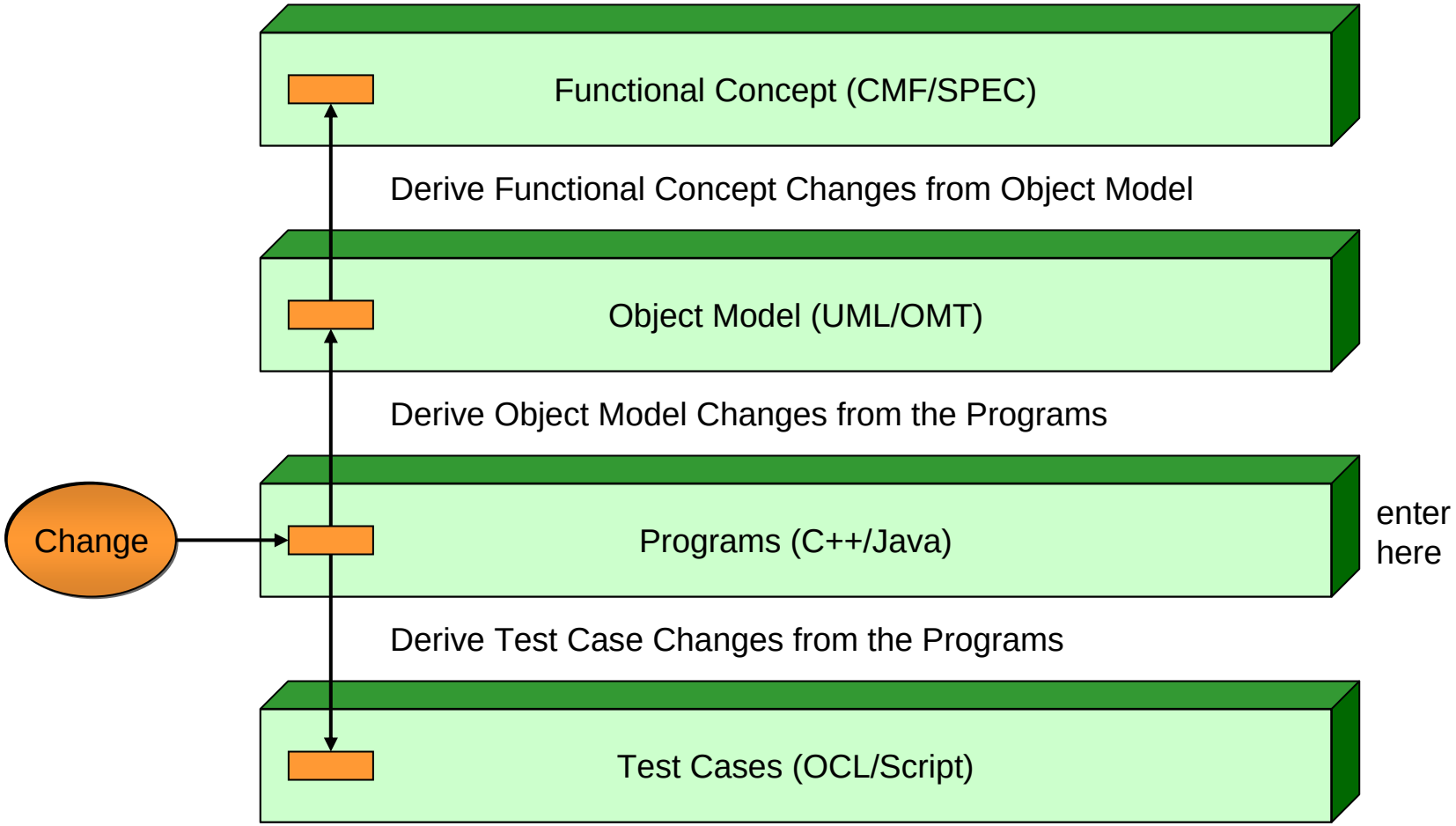
Methods of Software Evolution



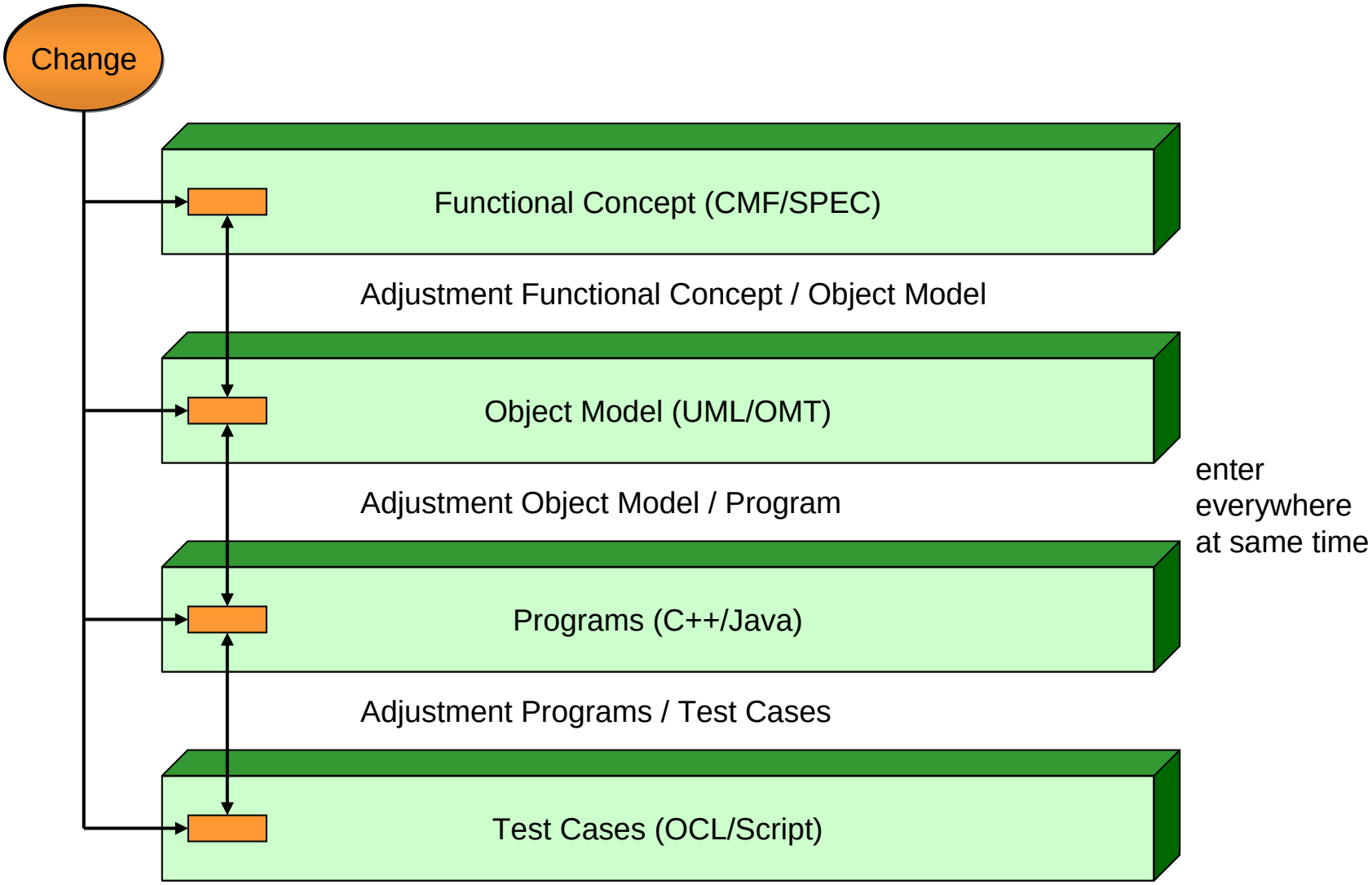
TOP DOWN Change Method

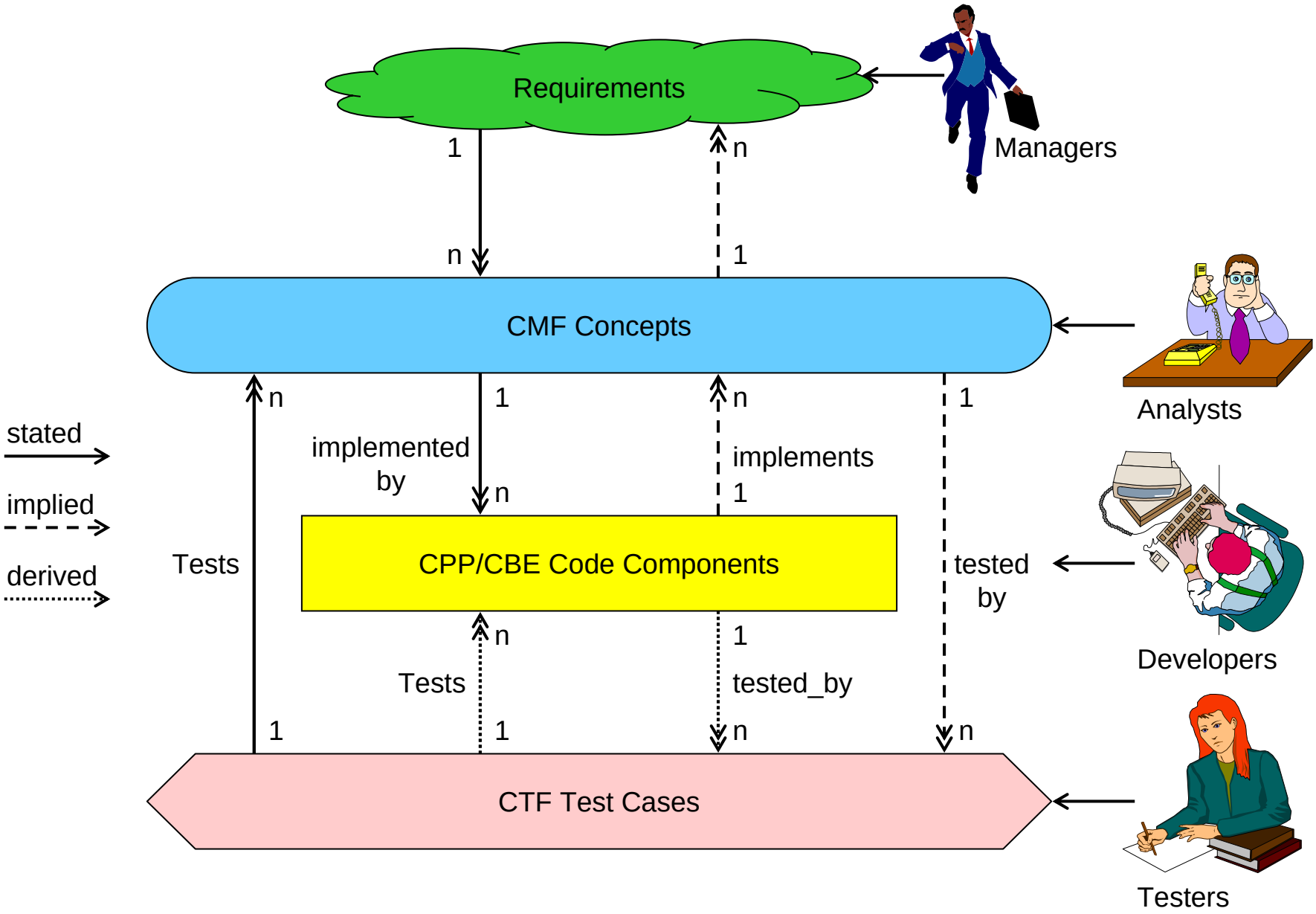


BOTTOM UP Change Method



Parallel Change Method





4-Layered GEOS Product Structure

Alternate Approaches to Software Evolution

- The Top-Down Model driven Approach
- The Bottom-Up Code driven Approach
- The Dual Approach
- The Requirement driven Approach
- The Test Driven Approach

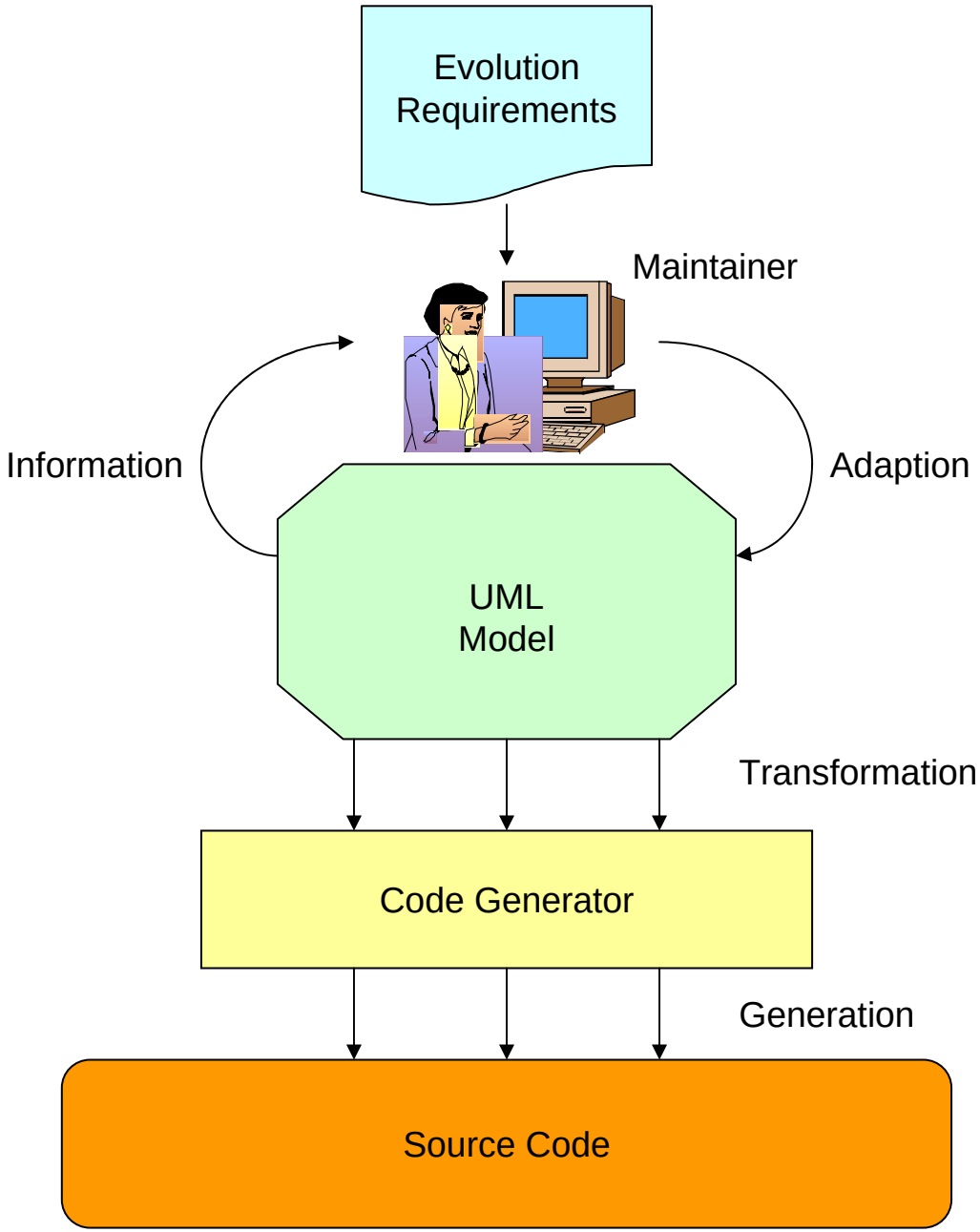


Figure 1: **Top Down Model-driven Approach**

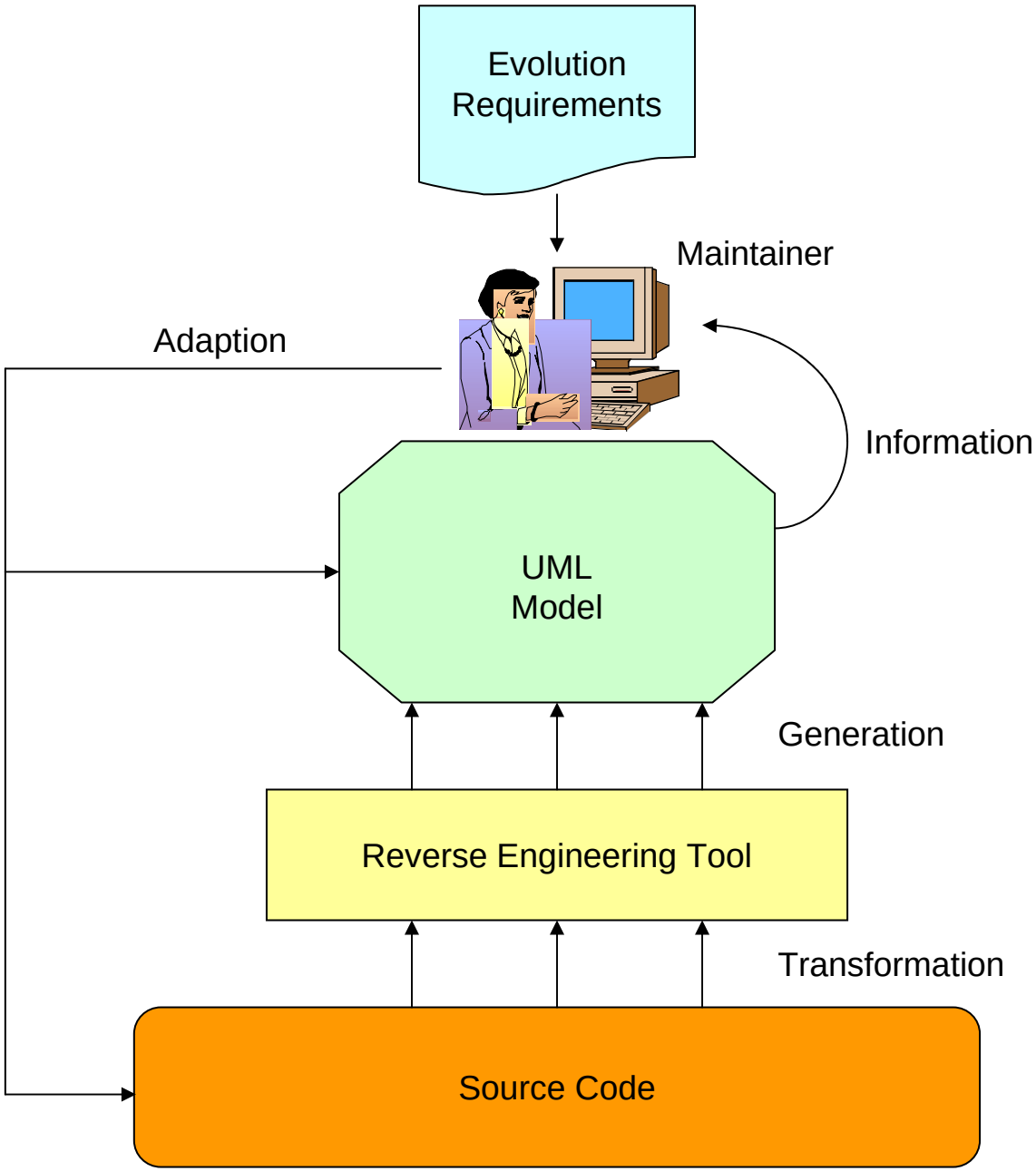


Figure 2: **Bottum-Up Code-driven Approach**

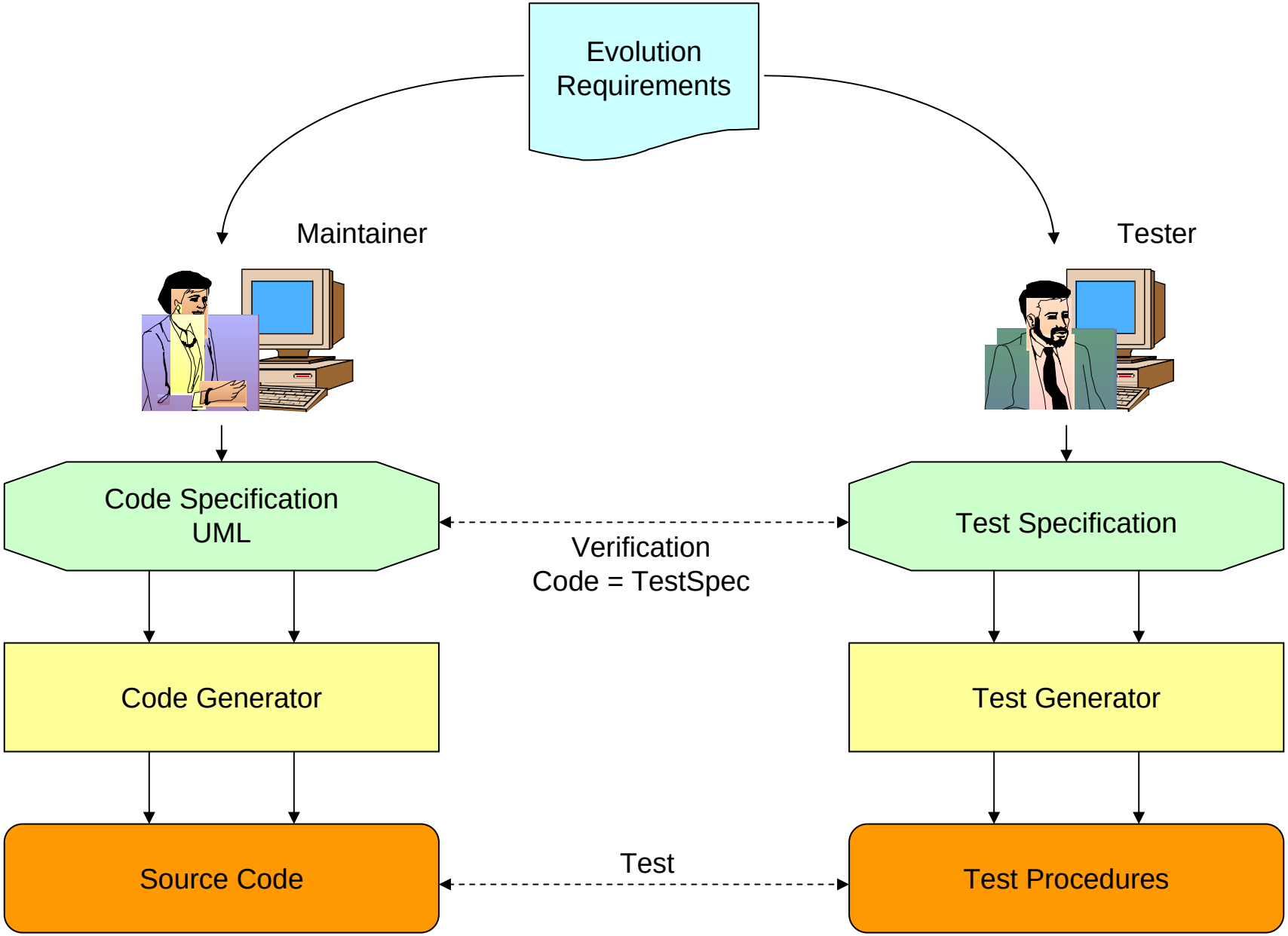


Figure 3: **Dual Approach**

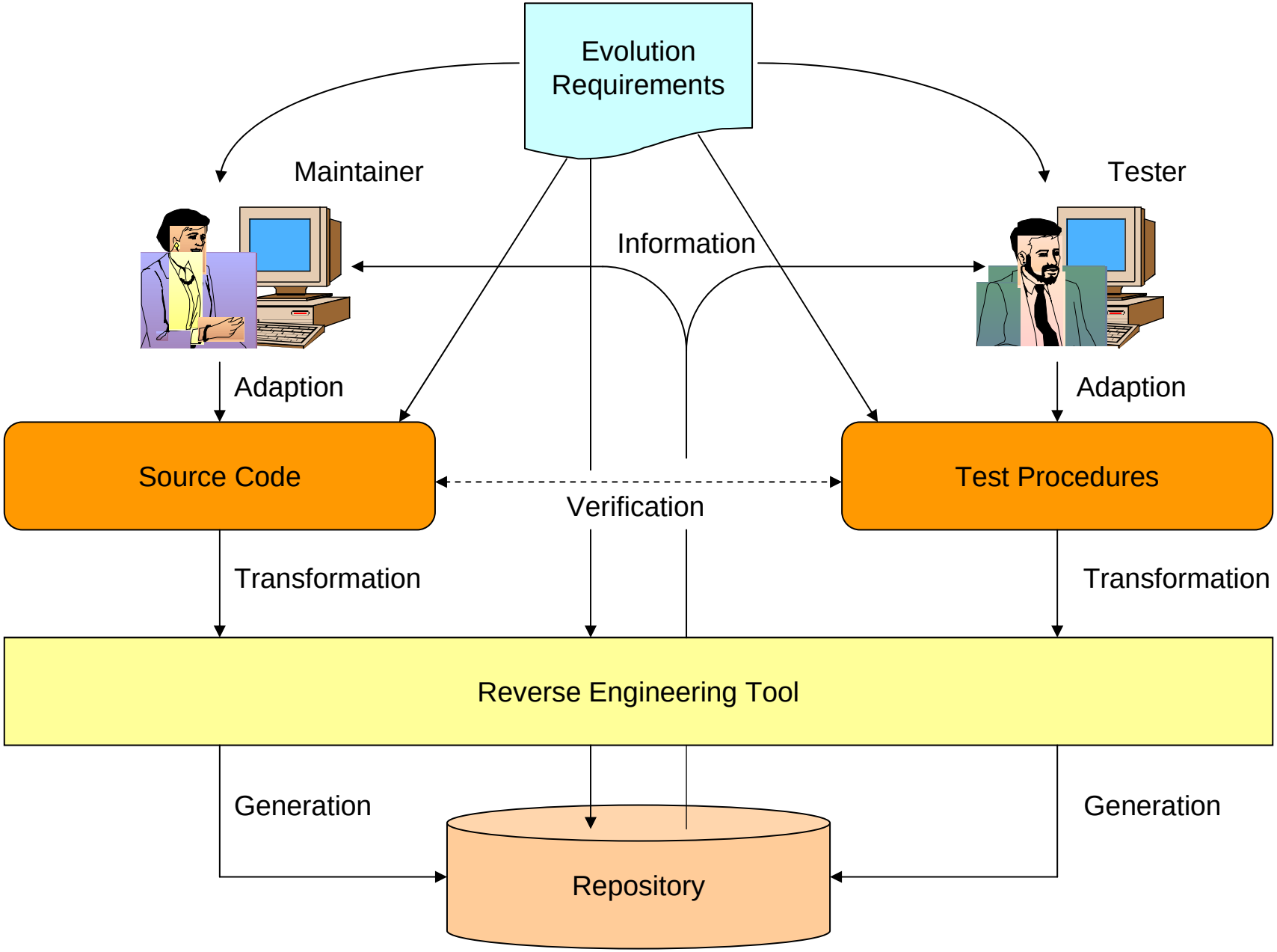


Figure 4: **Requirements-driven Approach**

Software is Communication

- Each semantic software level of abstraction serves some communication purpose.
- Requirements serve the communication between developers and users.
- Code serves the communication between developers and the machine.
- Test cases serve the communication between testers, users and developers.
- Design models serve the communication between developers.

The Role of Communication in Software Evolution

- The essential communication is that between humans and the computer, i.e. the code. It must be maintained in any case.
- The next most important communication is that between users and developers. It should be maintained in the requirements documents. There should be no change requests, instead the requirements should be evolved.
- The third most important means of communication is the test cases. They should be maintained to ensure the quality of new releases. Test cases are also an excellent means of communicating with the users.
- The least important media of communication is the design documentation. It is nice to have one, but it is not essential. Most IT users have managed to live without it for years. Besides it can always be reproduced from the code.
- Of course, it would be nice to maintain and evolve all the semantic levels of a software product, but in view of the costs this is seldom possible. So if any has to be sacrificed than it had best be the design documentation.

Summary

- If a UML design can really replace the programming code as envisioned by Jacobson in his paper „UML all the way down“, then it becomes just another programming language.
- The question then comes up as to what is easier to change
  - The design documents or
  - The programming language
- This depends on the nature of the problem and the people trying to solve it. If they are more comfortable with diagrams, they can use diagrams. If they are more comfortable with text, they should write text.
- Diagrams are not always the best means of modelling a solution. A solution can also be described in words. The important thing is that one model is enough – either the code or the diagrams. They should be reproducible from one another.