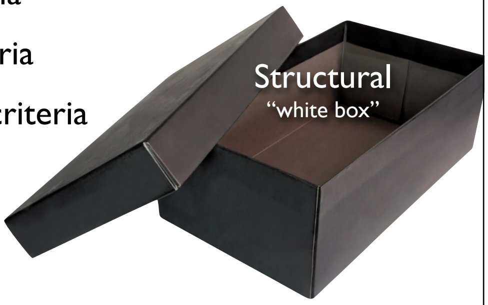
Structural Testing

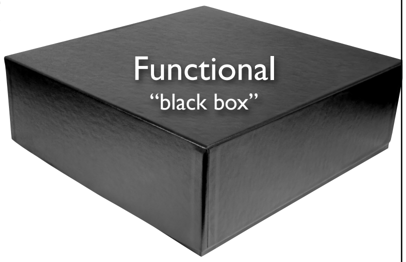
- Path coverage criteria
- Logic coverage criteria
- Dataflow coverage criteria
- Mutation testing



Structural testing takes a look at the internals of a program, and aims to exercise the code as thoroughly as possible. The main systematic approaches to structural testing use different coverage criteria or mutants as test goals.

Functional Testing

- Boundary Value Testing
- Equivalence Class Testing
- Decision Table-Based Testing
- Combinatorial Testing
- Grammar-based Testing
- Model-based Testing

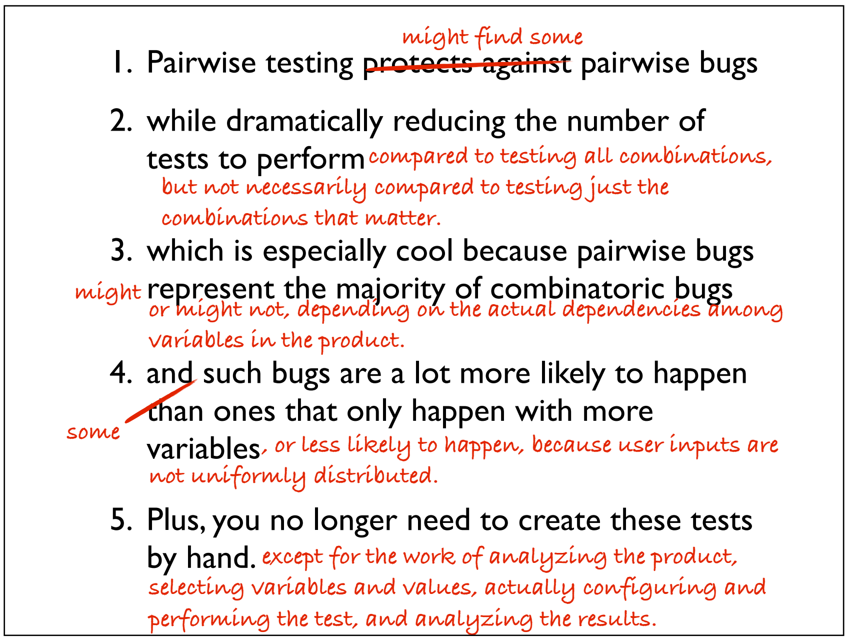


Functional testing considers the system under test as a black box of which only the inputs and outputs are known - this is complementary to structural testing. Functional testing uses specifications to derive test cases - a specification can range from an interface definition, informal specification, or even formal specification.

IPO Strategy

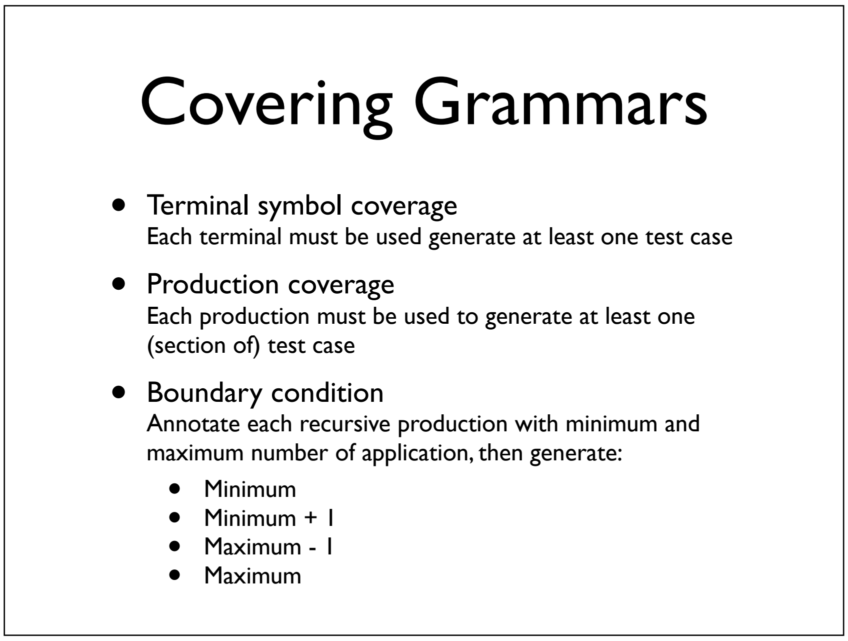
- Builds a t-way test set in an incremental manner
  - A t-way test set is first constructed for the first t parameters,
  - Then, the test set is extended to generate a t-way test set for the first t + 1 parameters
  - The test set is repeatedly extended for each additional parameter.
- Two steps involved in each extension for a new parameter:
  - Horizontal growth: extends each existing test by adding one value of the new parameter
  - Vertical growth: adds new tests, if necessary

Combinatorial testing is an effective way to reduce the number of test cases necessary while still detecting the majority of faults caused by interactions between parameters. IPO (In Parameter Order) is one of many heuristics to generate a covering array, which represents a t-wise test suite.



1. Pairwise testing ~~protects against~~ might find some pairwise bugs
2. while dramatically reducing the number of tests to perform compared to testing all combinations, but not necessarily compared to testing just the combinations that matter.
3. which is especially cool because pairwise bugs might represent the majority of combinatoric bugs or might not, depending on the actual dependencies among variables in the product.
4. ~~and~~ some such bugs are a lot more likely to happen than ones that only happen with more variables, or less likely to happen, because user inputs are not uniformly distributed.
5. Plus, you no longer need to create these tests by hand. except for the work of analyzing the product, selecting variables and values, actually configuring and performing the test, and analyzing the results.

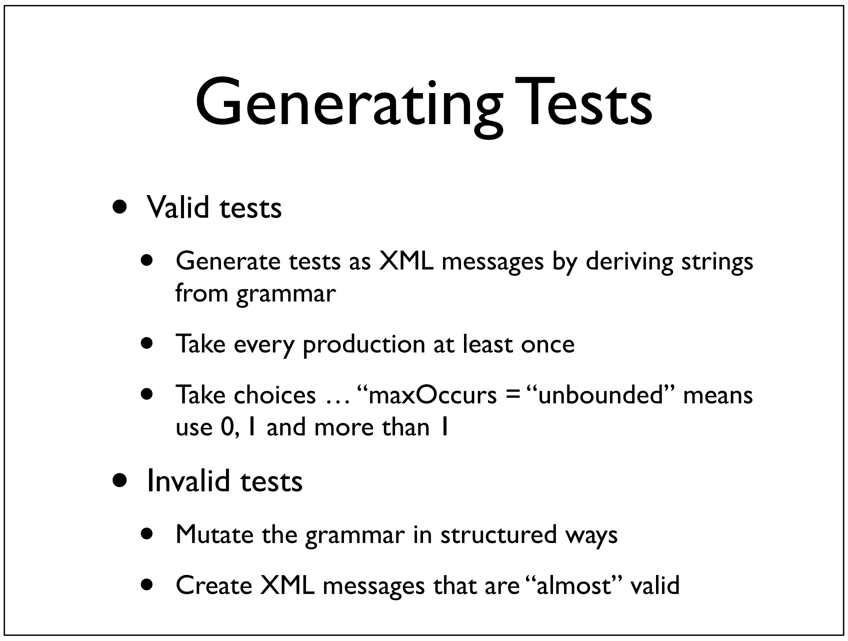
Combinatorial testing reduces the number of tests by only requiring combinations of all parameters up to a certain number – 2-way (pairwise), 3-way, etc. It is a very popular technique that has been shown to be effective at fault finding (e.g., 90% of all faults have been reported to be caused by interaction of two parameters). Despite such impressive data, one should not blindly adopt a technique but understand the testing problem at hands.



Covering Grammars

- Terminal symbol coverage
  Each terminal must be used generate at least one test case
- Production coverage
  Each production must be used to generate at least one (section of) test case
- Boundary condition
  Annotate each recursive production with minimum and maximum number of application, then generate:
  - Minimum
  - Minimum + 1
  - Maximum - 1
  - Maximum

Grammar-based testing is one of the oldest types of automated testing, originally conceived to test compilers. A grammar can be used to generate complex (textual) test input. As a grammar potentially defines an infinite number of possible data, coverage criteria are once again applied.



Generating Tests

- Valid tests
  - Generate tests as XML messages by deriving strings from grammar
  - Take every production at least once
  - Take choices … "maxOccurs = "unbounded" means use 0, 1 and more than 1
- Invalid tests
  - Mutate the grammar in structured ways
  - Create XML messages that are "almost" valid

Invalid test data can be produced by mutating a grammar. Mutation can be applied before or during test generation with the grammar.