



Specification Based Testing – Part 2

Combinatorial Coverage as an Aspect
of Test Quality

Objective



Objective

Apply
combinatorial test
coverage to
assess test
quality

Assessing Test Quality



| Numerous approaches exist for assessing test quality including:

- Combinatorial coverage
- Mutation testing

| Combinatorial coverage looks at how combinations of parameter values are tested together

| Various studies show that most failures can be detected with combinations of a small number of parameters

https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=917352

Total t-way Coverage

| For a given test set of “n” variables and values, proportion of t-way variable-value combinations that are executed

| E.g. Assume we are testing a function with 3 variables:

- Variable a: has values 0 and 1
- Variable b: has value 0 and 1
- Variable c: has values 0 and 1

| What is the total 2-way variable-value configuration coverage achieved by the following tests:

a=0; b=0; c=0

a=1; b=1; c=1

a=1; b=0; c=0

Summary





Specification Based Testing – Part 2

Design of Experiments

Objective



Objective

Apply Design of
Experiments to
develop tests

Background



| **Design of Experiments (DOE) is a systematic approach for evaluating a system or process**

| **DOE is heavily utilized in manufacturing and quality engineering**

| **DOE enables efficient investigation of the behavior of a system**

Traditional Experimentation



| Traditional evaluation of the behavior of a system involves designing an experiment in which one factor is modified and the behavior on the system is assessed

| For example, consider varying oven temperature on the impact of the quality of a pizza

Weakness of Traditional Approach



| The behavior of most systems is impacted by many factors

| Factors may also combine to create interactions

| In the pizza case, additional factors include:

- Rack positioning
- Cook time

DOE Advantages



| DOE enables examination of the impact of a single factor as well as combinations of factors

| Values / ranges must be determined for each factor to investigate

| Experiments (runs) are made with combinations of the factors being considered and their impact on the system

Pizza Example Factors



| Cook time

– Low / Med / High

| Rack position

– 1 / 2 / 3 / 4 / 5

| Temperature

– 350 / 375 / 400

Full DOE Combinations



| 45 Runs

DOE Classification



| Full factorial design

- Tests for every factor value combination
- Pizza example

| Fractional factorial design

- Only a fraction of all combinations are addressed
- Orthogonal arrays often used to address limited combinations of factors

Design of Experiments Pairwise Combinations



- 1. Identify the parameters that define each configuration**
- 2. Partition each of the parameters**
- 3. Specify constraints prohibiting particular combinations of configuration partitions**

Design of Experiments Pairwise Combinations



- 4. Specify configurations to test which cover all pairwise combinations of configuration parameter partitions satisfying the constraint**
 - “For any two parameters P1 and P2 and for any partition value V1 for P1 and V2 for P2, there is a specified configuration where P1 has the value V1 and P2 has the value V2. “

Pizza Example

1	Med	350
2	Low	350
3	High	350
4	Low	350
5	Low	350
1	Low	375
2	High	375
3	Med	375
4	Med	375
5	Med	375
1	High	400
2	Med	400
3	Low	400
4	High	400
5	High	400

Experiences with DOE in Software Testing



| Several companies have used DOE in software testing and have reported good results

| DOE has been shown to achieve reasonable code coverage

Warning



| Many software functions contain many parameters and factors

| Pairwise combination testing may leave many functions untested with normal, everyday scenarios

Summary





Specification Based Testing – Part 2

Mutation Testing

Objective



Objective

Understand
Metamorphic
Testing

Test Oracle Problem



- | A set of tests has been developed by an organization

- | The organization executes the tests against a program

- | All of the tests pass

- | What can we conclude?

Mutation Testing



- | Introduce defects (mutants) into program undergoing test

- | Check to see if test cases can detect the mutant

- | Work began in early 70's but was not widely adopted due to cost

- | Today's automated testing environments make mutation testing feasible

Creating Mutants



| **Mutants are typically created via syntactical modifications of source code**

| **Mutation generation tools exist for this**

| **Examples of mutations**

- Modify Boolean expressions (< vs <=)
- Delete Statement
- Modify Variable
- Modify arithmetic operation

Mutation Testing Assumptions



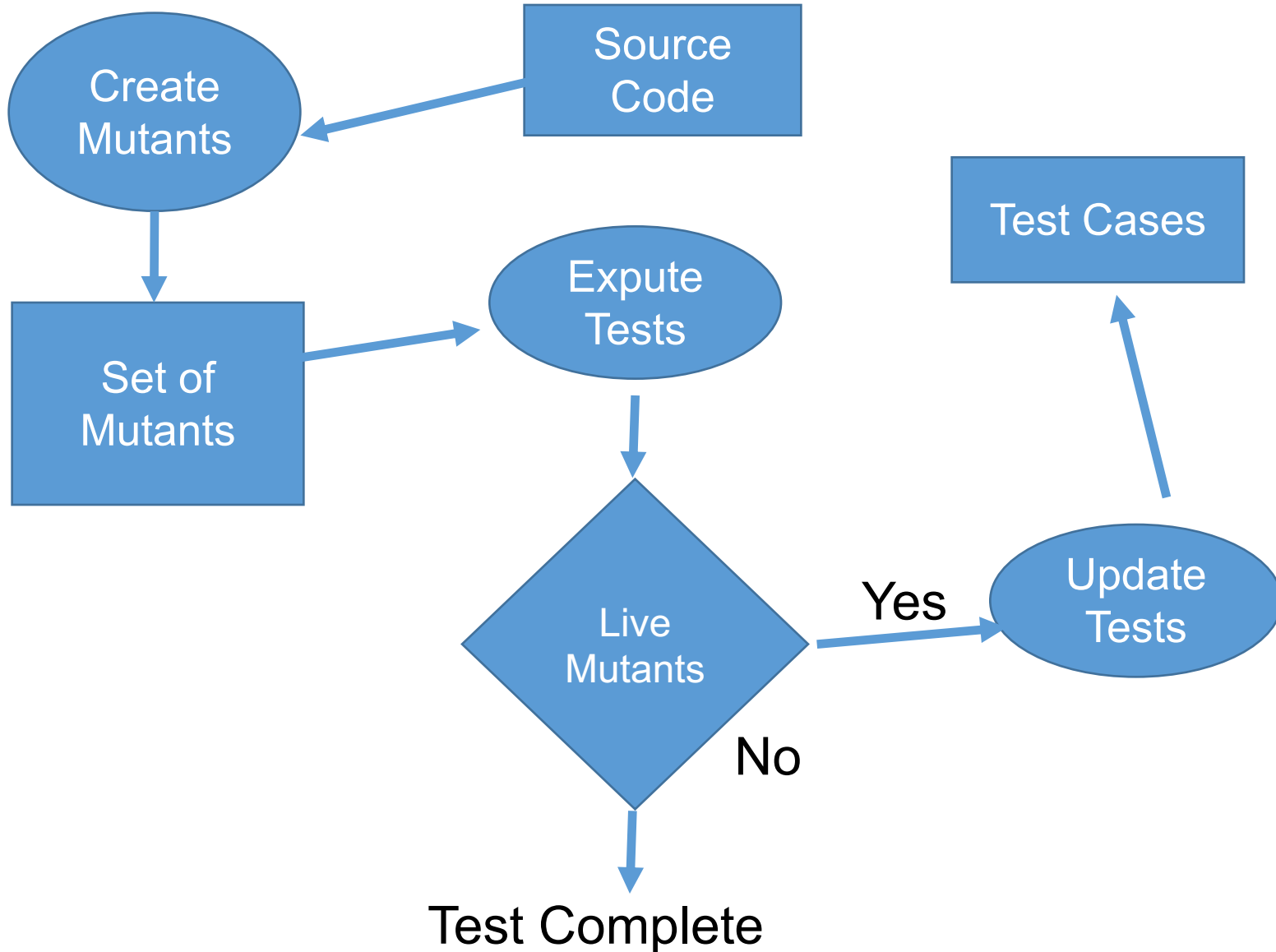
| The Competent Programmer Hypothesis

- Programmers generally create code that is close to being correct reflecting only minor errors

| The Coupling Effect

- Belief that test data that can detect small errors can also detect complex errors

The Mutation Testing Process



Summary





Specification Based Testing – Part 2

Fuzz Testing

Objective



Objective

Understand Fuzz
Testing

Fuzz Testing



- | Approach to testing where invalid, random or unexpected inputs are automatically generated

- | Often used by hackers to find vulnerability of the system

- | Test oracle is not needed

 - Only monitor for crashes or undesirable behavior

- | Fuzzing tool used to generate inputs

Two Types of Test Generators



| **Mutation Based**

| **Generation Based**

Mutation Based Fuzzing



- | Generates test inputs by random modifications of valid test data

- | Doesn't require knowledge of the inputs

- | Modifications may be totally random or follow some pattern tied to frequent error types such as:

- Long or blank strings
- Maximum or minimum values
- Special characters

- | Some tools use “bit flipping” - corrupt input by changing random bits in input

Generation Based Fuzzing



- | Generates random test data based on specification of test input format

- | Anomalies are added to each possible spot in the inputs

- | Knowledge of protocol should give better results than random fuzzing

When to Stop Fuzz Testing?



| Utilize code coverage tools

Summary





Specification Based Testing – Part 2

Metamorphic Testing

Objective



Objective

Understand
Metamorphic
Testing

Test Oracle Problem



**| For many applications
it is difficult to
determine the
expected results**

- Graphics applications
- Complex processing
- Machine Learning
- Big Data

**| Fuzz testing can be
used to detect crashes**

**| Metamorphic testing
can assist**

Assumptions

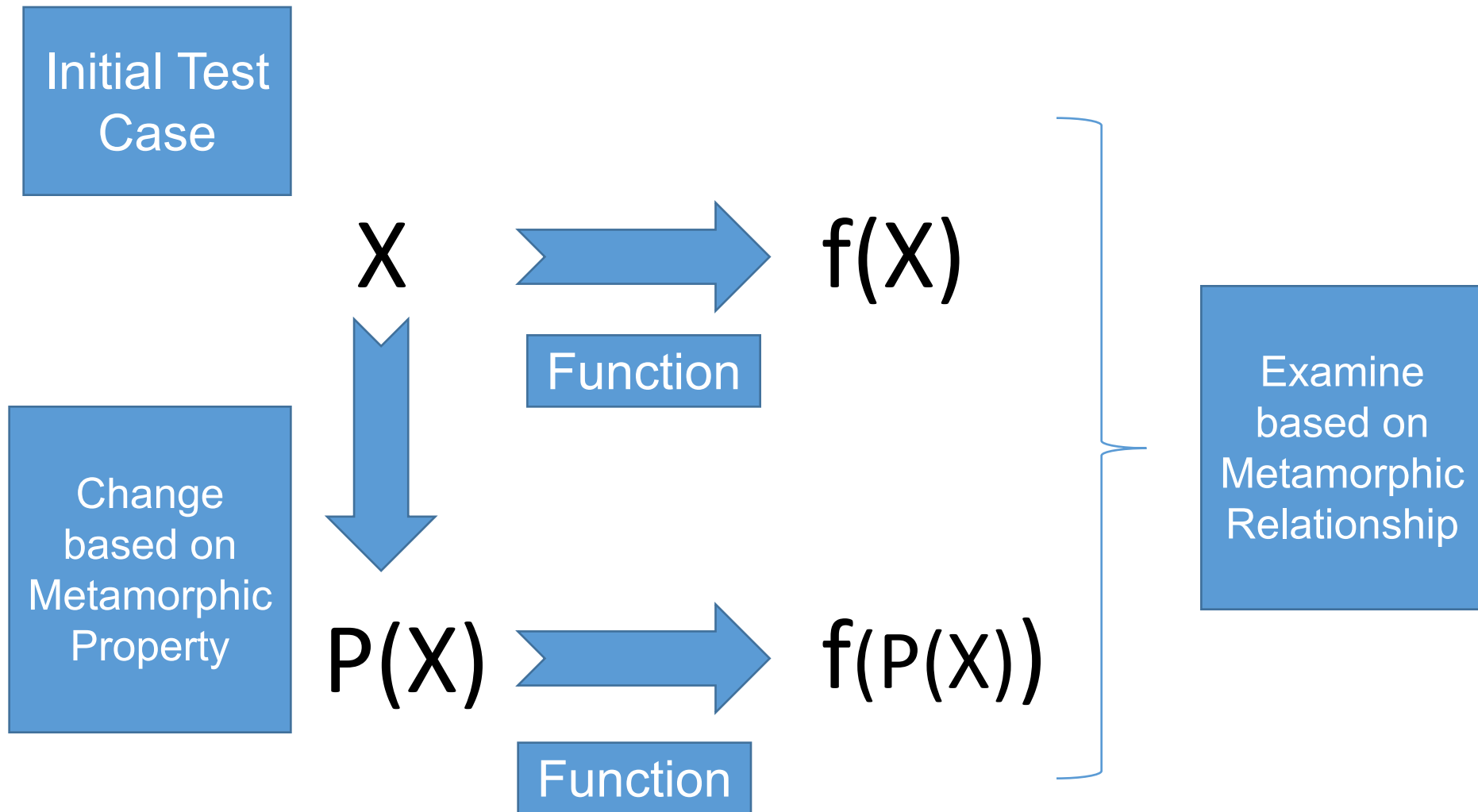


| Some programs have properties such that when changes are made to inputs it is possible to predict changes on outputs (metamorphic properties)

| Consider a service that calculates the variance of a sequence of numbers

- What is the relationship of changing the order?
- What is the relationship of adding 10 to each number?

Metamorphic Testing



Standard Deviation Example



| **Initial Test: 10, 20, 30, 40, 50**

– Result: 14.142

| **Second Test: 20, 30, 40, 50, 60**

– Result: 14.142

| **Third Test: 10, 30, 50, 70, 90**

– Result: 28.284

Summary

