Testing Context
SFWR ENG 3S03: Software Testing

A. Marinache

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Slides adapted from Dr.R.Khedri and Dr.R.Paige.

Objectives

- Understand in what context we perform testing
- Explore different testing techniques and classify them
- View testing from the software lifecycle perspective

https://bit.ly/42otDlj

- Why bother?
  - Include activities that add value

- Why bother?
  - Include activities that add value and address a risk
- Who cares?
  - Include activities that serve someone's interest
- How much?
  - "We will test all combinations of printer features"

Software applications can be:

- Shrinkwrap or COTS
- Internal
- Embedded
- Games
- Throwaway

https://www.joelonsoftware.com/2002/05/06/five-worlds/

What does this mean to me (the tester/developer)?

- Testing varies based on its context
- Know the project, its challenges, and risks

Introducing...

PhonePicture*

- Manage pictures on your phone!
- Send them to your friends!
- Rate photos online!
- Win prizes!
- Your private photos are encrypted!

(* definitely different from any app you've encountered)

Brainsotrm for a minute:

- What kind of software is this?
- Why?
- What does it mean for testing it?

Detect as many defects as possible

- Functionality
- Communication
- Structure
- Missing commands
- Program rigidity
- Performance
- Error handling
- Calculation errors
- etc.

Wait... what IS a defect?

- Defect is a flaw in the system
- Error is a human mistake that introduces the defect in the system
- Failure is the manifestation of the defect when executing the system

Example (The "Meteoro-X" meteorological equipment firmware [Gal04])

The software requirements for "Meteoro-X" meteorological equipment firmware were meant to block the equipment's operation when its internal temperature rose above 50°C. A programmer error resulted in a software fault when the temperature limit was coded as 150°. This fault could cause damage when the equipment was subjected to temperatures higher than 50°. In Northern Europe, the software fault never turned into a software failure. In Southern Europe, an equipment disaster occurred.

SO... Where do we start?

"No matter how many test cases of how many types you've created, you will run out of formally planned tests. You can keep testing. Run new tests as you think of them, without spending much time preparing or explaining the tests. Trust your instincts."

Cem Kaner, *Testing Computer Software*

Definition (Exploratory Testing)

Emphasizes learning, discovery, and adaptability during the testing process

- Goals: Learn and adapt; Uncover usability issues
- When: Requirements are unclear; Deadlines are tight
- How: Define the charter – Explore the system – Document – Share

Pros

- May discover critical defects
- Flexibility towards requirements
- Cost-effective for early detection

Cons

- Difficult to replicate (and document)
- Reliant on tester experience and skill
- Not suitable for compliance-driven indutries

Definition (Specification-based Testing)

Testing against all the explicit claims made about it in the specification

- Goals: Enforce correctness
- When: Specifications are clear and "automate-able"
- How: Using tools that allow pre- and post-conditions

Contracts as specificaton: Eiffel

```
class BANK_ACCOUNT
feature
    withdraw(amount: INTEGER)
        require
            amount > 0
            balance >= amount
        do
            balance := balance - amount
        ensure
            balance = old balance - amount
        end
end
```

Contracts as specificaton: SPARK

```
procedure Increment (X : in out Counter_Type);
--# derives X from X;
--# pre X < Counter_Type'Last;
--# post X = X~ + 1;
```

Source: http://en.wikipedia.org/wiki/SPARK_(programming_language

Pros

- Encourages clear precise requirements
- Early detection of defects
- Synchronization of code and specifications

Cons

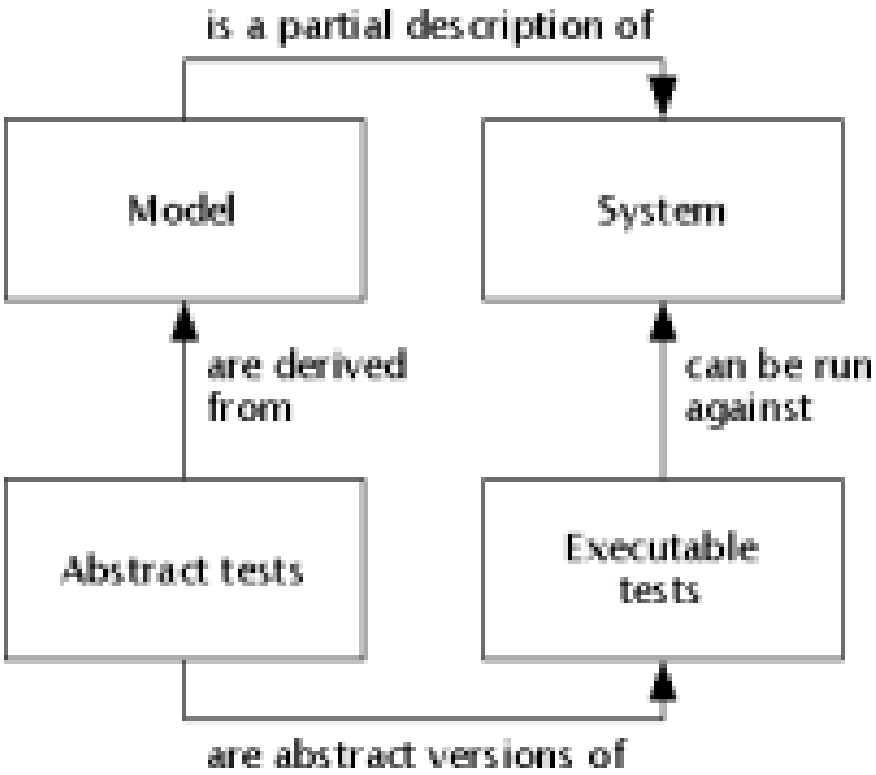
- Requires specific knowledge
- May add overhead
- May need additional tools for automation

Is there something in-between?

Definition (Model-based testing)

Using a formal model of the system or its behavior to (automatically) generate and execute test cases

- Goals: Improve coverage; Bridge gaps between teams
- When: Complex systems; Stable requirements
- How: (next slide)



Source: `https://en.wikipedia.org/wiki/Model-based_testing`

Scenario: Testing a login system

Pros

- Automation
- Consistency
- Reusability

Cons

- Initial cost
- Complexity
- Dependence on model accuracy

Definition (Fuzz Testing)

Providing random, unexpected, or invalid inputs to a program to detect vulnerabilities

- Goals: Improve software robustness
- When: Security scenarios
- How: Setup fuzzer – Generate random inputs – Monitor – Analyze – Refine

Pros

- Scalable
- Automated
- May uncover unusual vulnebaritilies

Cons

- False positives
- Cost of setup
- Hard tp reproduce

Will it be effective on... ?

```
/**
 * Adds a value to draw.
 *
 * @param x the x value.
 * @param y the y value
 */
public void addValue(double x, double y) {
    // if the maximum x is too low we extend it.
    if (maxX<x) {
        maxX=1.2*x;
        nDigitsX=(int)Math.floor(Math.log10(maxX));
    }
    // if the maximum y is too low, we extend it.
    if (maxY<y) {
        maxY=2*y;
        nDigitsY=(int)Math.floor(Math.log10(maxY));
        leftBorder = 20+nDigitsY*7;
    }
    // we add the point to the graph
    series[0].add(x);
    series[1].add(y);
}
```

Will it be effective on... ?

```
/**
 * Static method creating an instance of a given Equation type.
 * The method asks values of parameters through option panes.
 *
 * @param equationType the class of the equation
 * @return the Equation
 */
@SuppressWarnings("unchecked")
public static Equation createEquationFromType(Class equationType) {
    Constructor c =  equationType.getConstructors()[0];
    int n_arguments = c.getParameterTypes().length;
    Object[] arguments = new Double [n_arguments];
    for (int i=0; i<n_arguments;i++) {
        //ask for values
        String s = JOptionPane.showInputDialog(null, ((char)(((byte)'a')+i))+" =",
                "Enter argument", JOptionPane.QUESTION_MESSAGE);
        arguments[i] = Double.parseDouble(s);
    }
    try {
        // we return the new instance
        return (Equation)c.newInstance(arguments);
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "alert", "alert", JOptionPane.ERROR_MESSAGE);
    }
    // if there was no exception
    return null;
}
```

Recall PhonePicture app

Sketch how you could apply each of the following testing types to it

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing

Hint: what do you need for each technique?

Definition (Partition)

Testing performed by dividing the input domain into (ideally) disjoint groups and selecting one representative from each group

How do we partition these domains?

- Int
- Float
- Boolean
- Character
- String
- (Int, Int)
- ...

```
1  private int doSomething(int x) {
     if (x < -10)
3      return someOutput(x);
     if (x < 6)
5      return someOtherOutput(x);
     return yetAnotherOutput(x);
7  }
```

```java
public static double calculateDiscount(String userType, double
    cartValue, String region) {
  double discount = 0.0;

  // Apply discounts based on user type
  switch (userType) {
    case "Customer":
      if (cartValue > 500) discount = 10; // 10% for large orders
      break;
    case "Employee":
      discount = 20; // 20% flat for employees
      break;
    case "VIP":
      discount = 15; // 15% flat for VIPs
      if (cartValue > 500) discount += 5; // Extra 5% for big orders
      break;
    default:
      throw new IllegalArgumentException("Invalid user type");
  }

  // Additional fee/discount for international shipping
  if ("International".equalsIgnoreCase(region)) {
    discount -= 5; // Reduce discount by 5% for international orders
  }

  return discount;
}
```

Does it work for this?

```
    private int increment(int i) {
2     return i+1;
    }
```

Definition (Boundary)

Testing performed by choosing inputs that are on (semantically significant) boundaries

What are boundary values for these domains?

- Int
- Float
- Boolean
- Character
- String
- (Int, Int)
- ...

Does it work for this?

```
1     private int increment(int i) {
        return i+1;
3     }
```

How do we know we have tested ALL of the system?

- As a project manager, what evidence will you accept?
- As a SE, what evidence would you like to produce?

Definition (Coverage-based)

Testing approach that measures the extent the system is tested, based on pre-agreed coverage criteria

Coverage Criteria At least a test that involves every...

- Statement
- Branch
- Condition
- Path

Statement Coverage Examples

```
1    if (COND) then
       F1();
3    F2();
```

```
1    int* ptr = NULL;
     if (i > 20) then
3      ptr = &variable;
     *ptr = 10;
```

- What is a statement?
- How do we achieve Statement Coverage?

Branch Coverage Examples

```
1  if (COND1) then
2    F1();
3  else
4    F2();
5  if (COND2) then
6    F3();
7  else
8    F4();
```

```
1  if (A && (B || F1()))
2    F2();
3  else
4    F3();
```

- What is a branch?
- How do we achieve Branch Coverage?

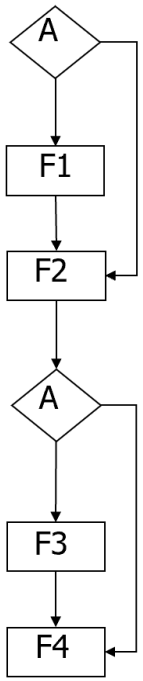
Condition Coverage Examples

```
    if (A && B) then
2     F1();
    else
4     F2();
    if (NOT C) then
6     F3();
    else
8     F4();
```

- What is condition coverage?
- How do we achieve Condition Coverage?

Path Coverage Examples

```
    if (A) then
2     F1();
    else
4     F2();
    if (A) then
6     F3();
    else
8     F4();
```



- What is a path?
- How do we achieve Path Coverage?

Do we really care about all this? Well...

`https://esamultimedia.esa.int/docs/industry/SME/2005-Training/2-SWE-course/ECSS-Q-80B-10October2003.pdf`

What if coverage criteria shows some code is not covered?

- Test set may not be adequate
  - Missing components
  - Missing properties
- Some code may be unreachable
- What can we do...?

Definition (Fault Injection)

Testing by deliberating introducing faults into the system

- Goals: Robustness testing; Safety verification
- When: Critical systems
- How: Introduce faults – Monitor propagation

Variations

- Compile-time: modify source code
- Run-time: use software triggers

```
int max(int x,
    int y) {
  int mx = x;
  if (x > y)
    mx = x;
  else
    mx = y;
  return mx;
}
```

```
int max(int x,
    int y) {
  int mx = x;
  if (x < y)
    mx = x;
  else
    mx = y;
  return mx;
}
```

Mutation Operators

- Operand replacement operators
- Expression modification operators
- Statement modification operators

Why does it work?

- Competent programmer hypothesis
- Coupling hypothesis

```java
public static double calculateDiscount(String userType, double
    cartValue, String region) {
  double discount = 0.0;

  // Apply discounts based on user type
  switch (userType) {
    case "Customer":
      if (cartValue > 500) discount = 10; // 10% for large orders
      break;
    case "Employee":
      discount = 20; // 20% flat for employees
      break;
    case "VIP":
      discount = 15; // 15% flat for VIPs
      if (cartValue > 500) discount += 5; // Extra 5% for big orders
      break;
    default:
      throw new IllegalArgumentException("Invalid user type");
  }

  // Additional fee/discount for international shipping
  if ("International".equalsIgnoreCase(region)) {
    discount -= 5; // Reduce discount by 5% for international orders
  }

  return discount;
}
```

Recall PhonePicture app

Sketch how you could apply each of the following testing types to it

- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

Static or Dynamic?

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

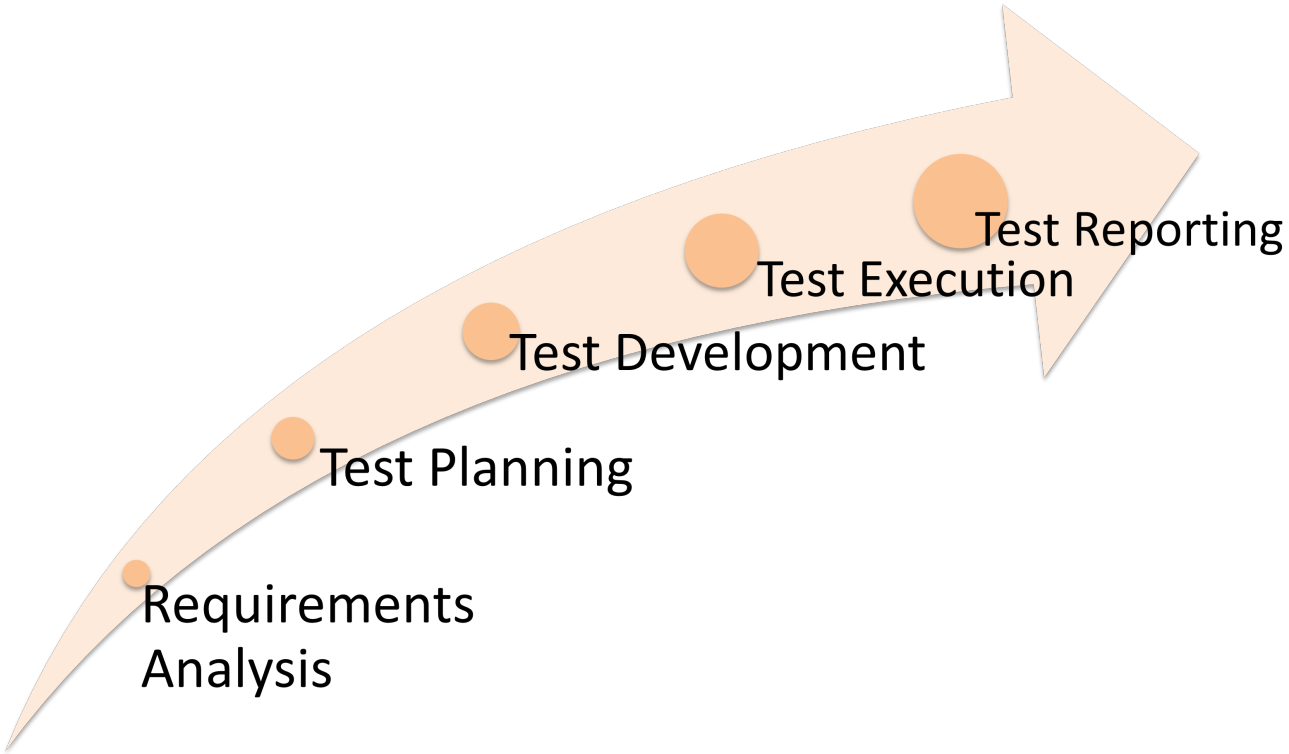
Black-/Grey-/White-Box?

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

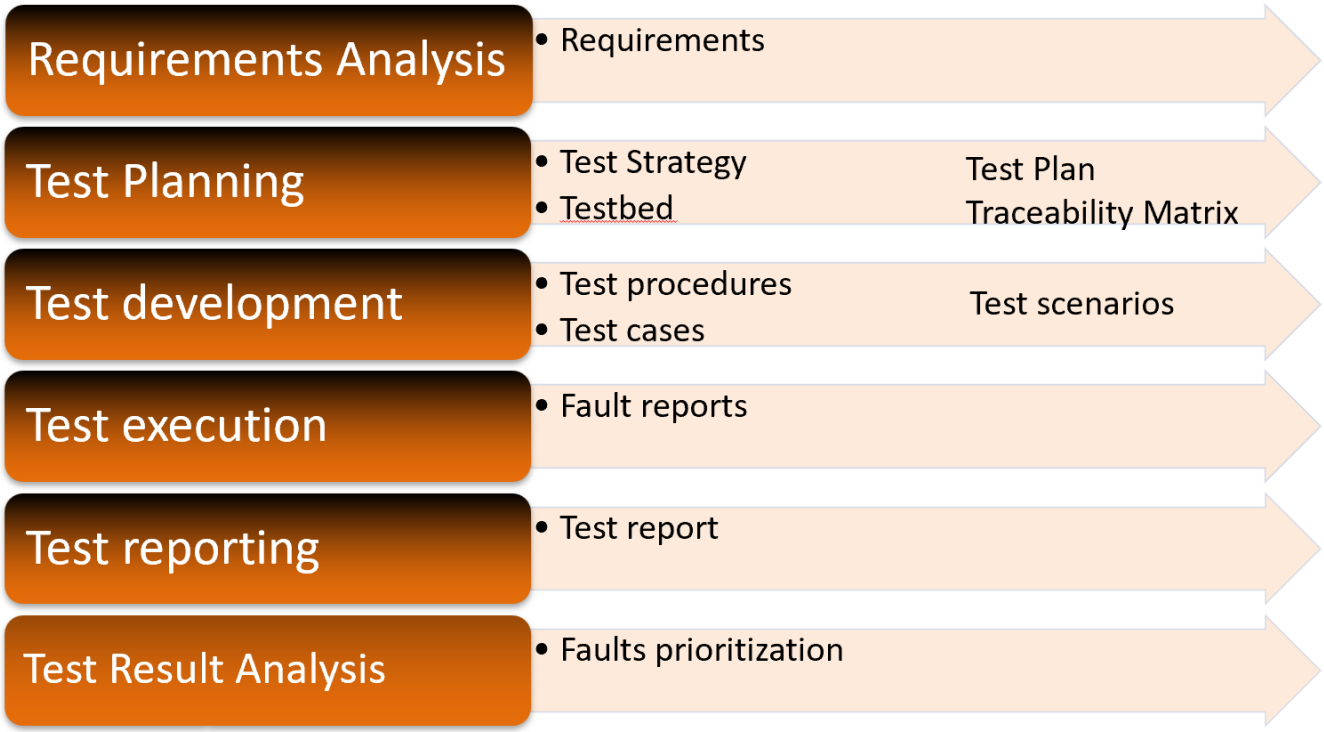
Functional or Non-Functional?

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

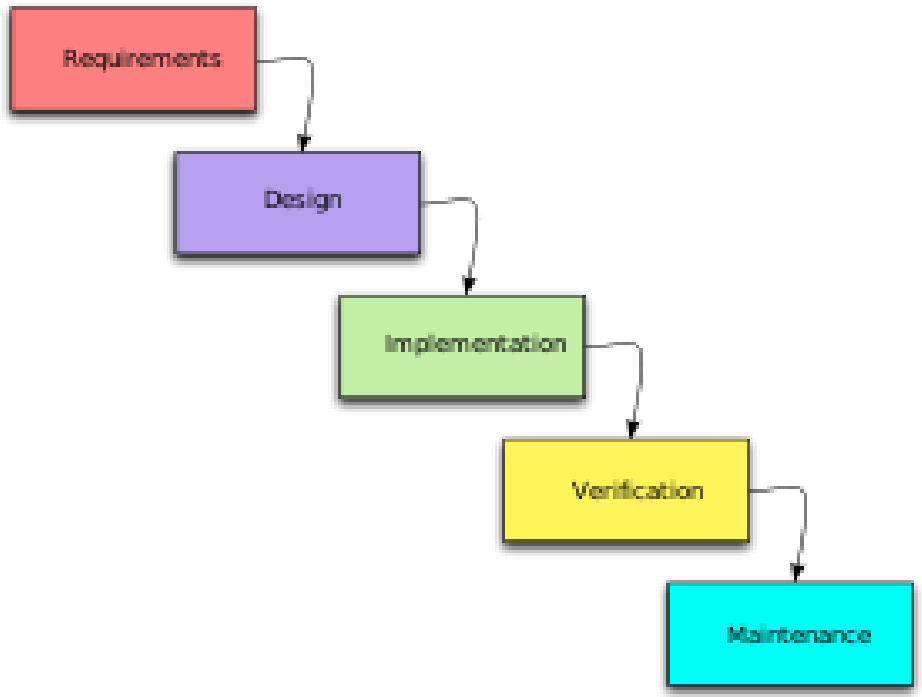
Typical QA Process



QA Artifacts

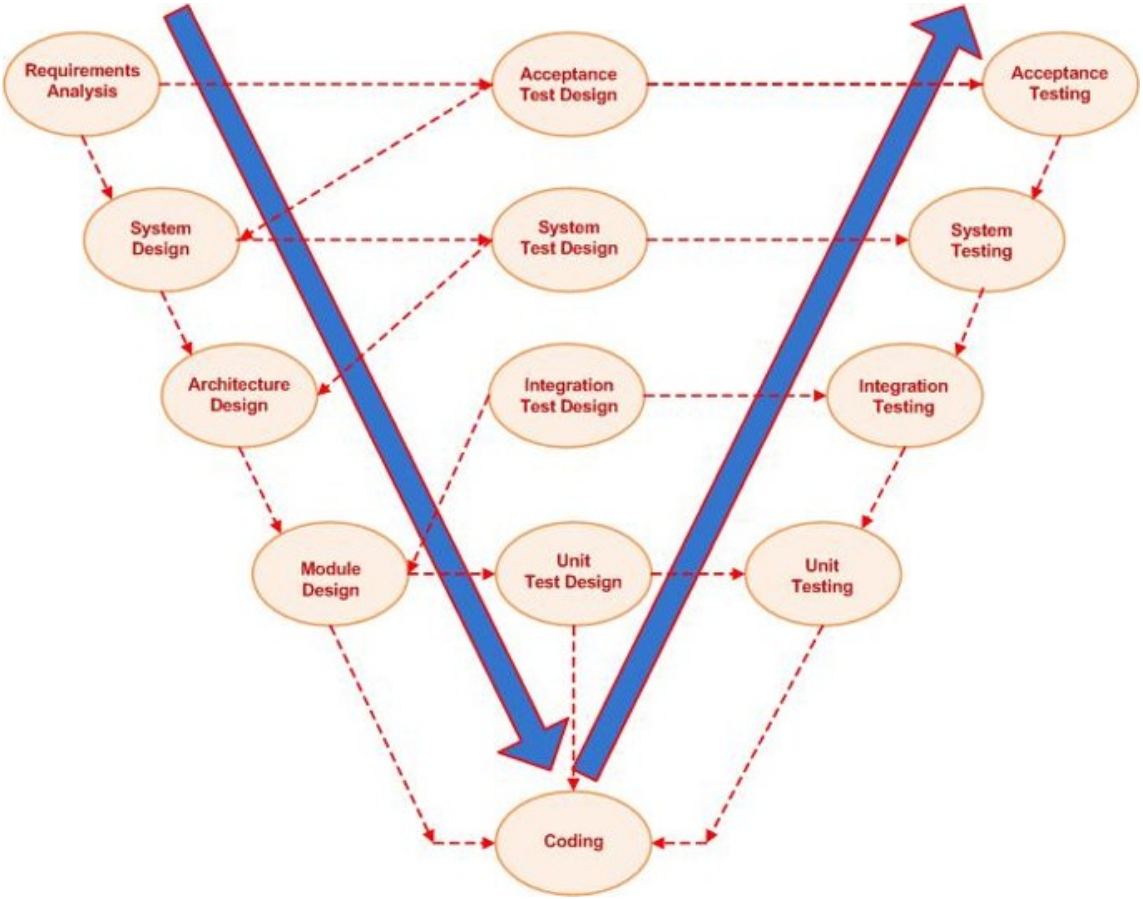


Waterfall model



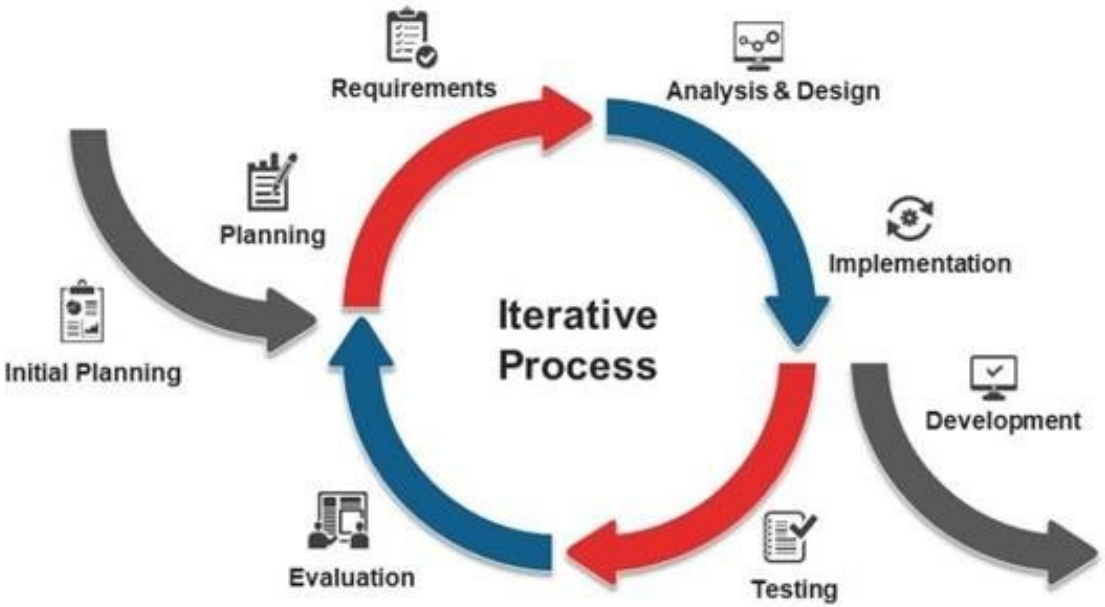
Winston Royce, 1970; Source: waterfall model, wikipedia

V-model



Source: v-model, wikipedia

V-model

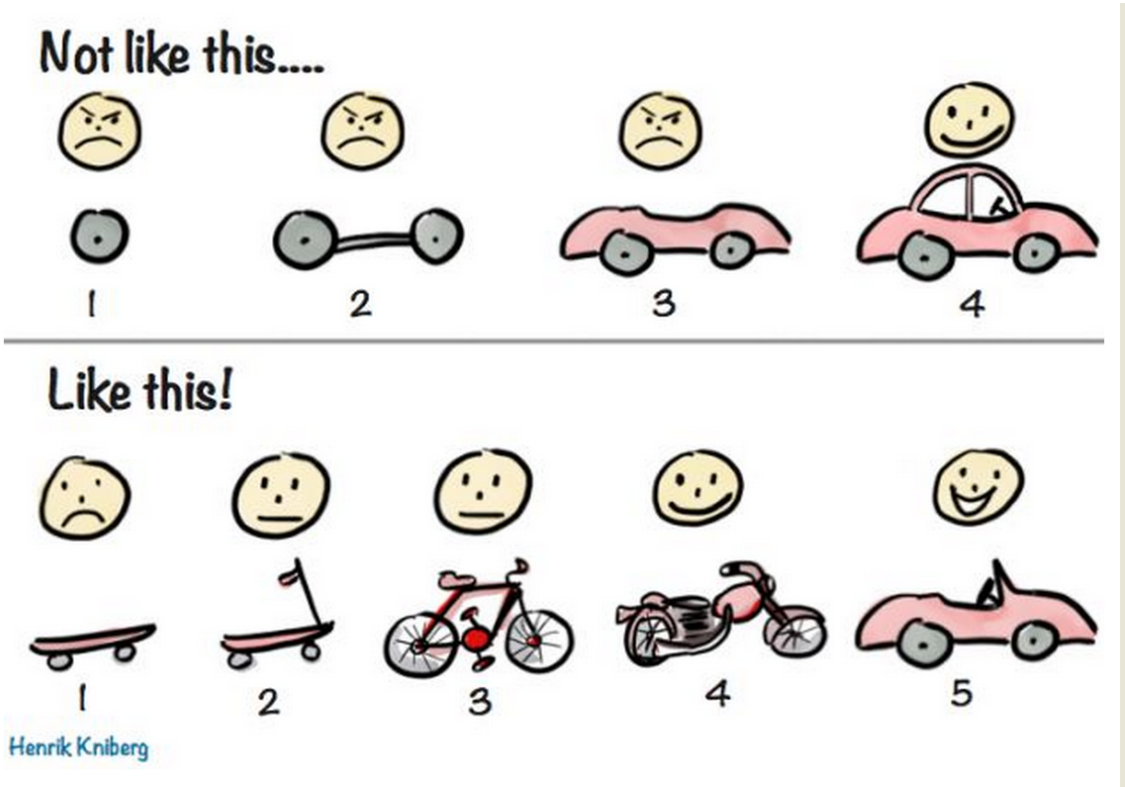


Source [MA]

Agile Framework

Source: `https://www.impactqa.com/`

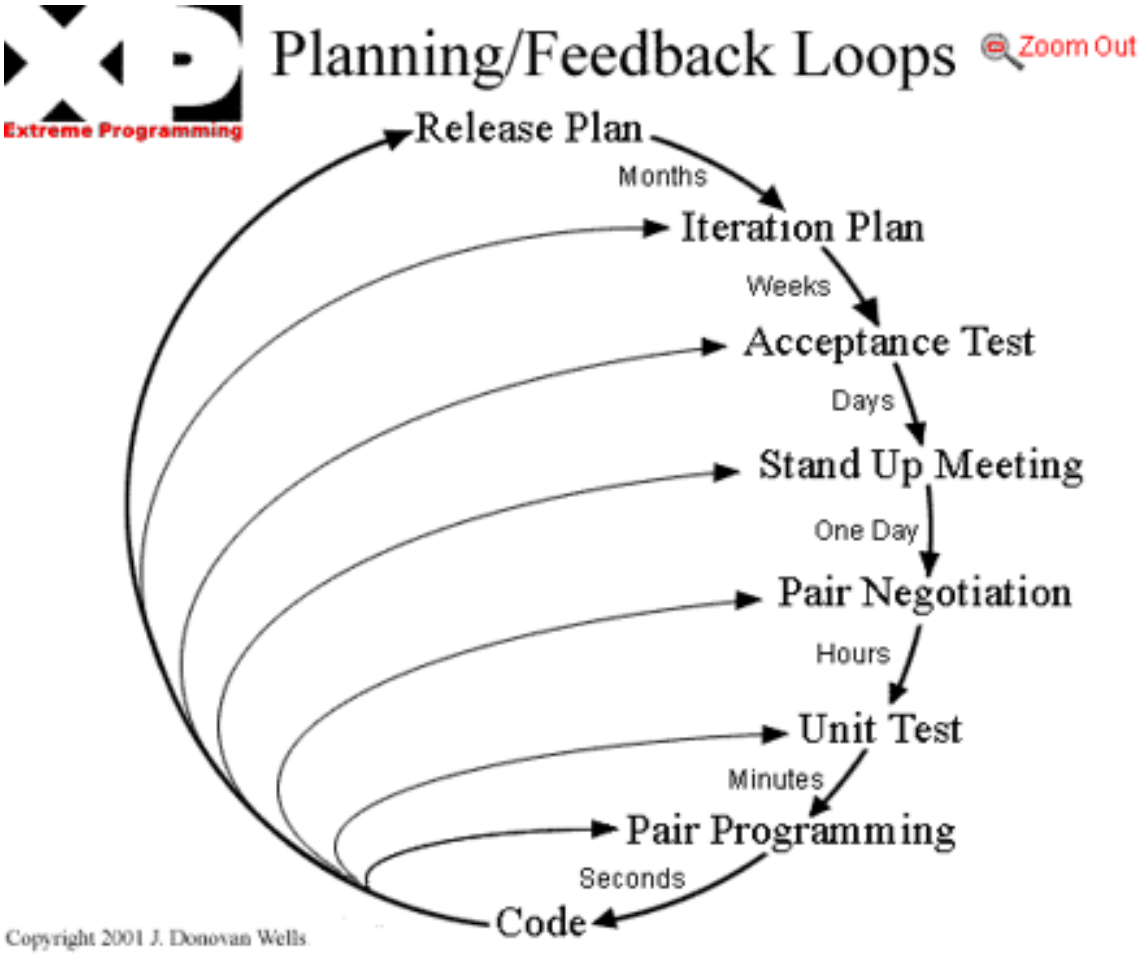
Iterative vs Agile



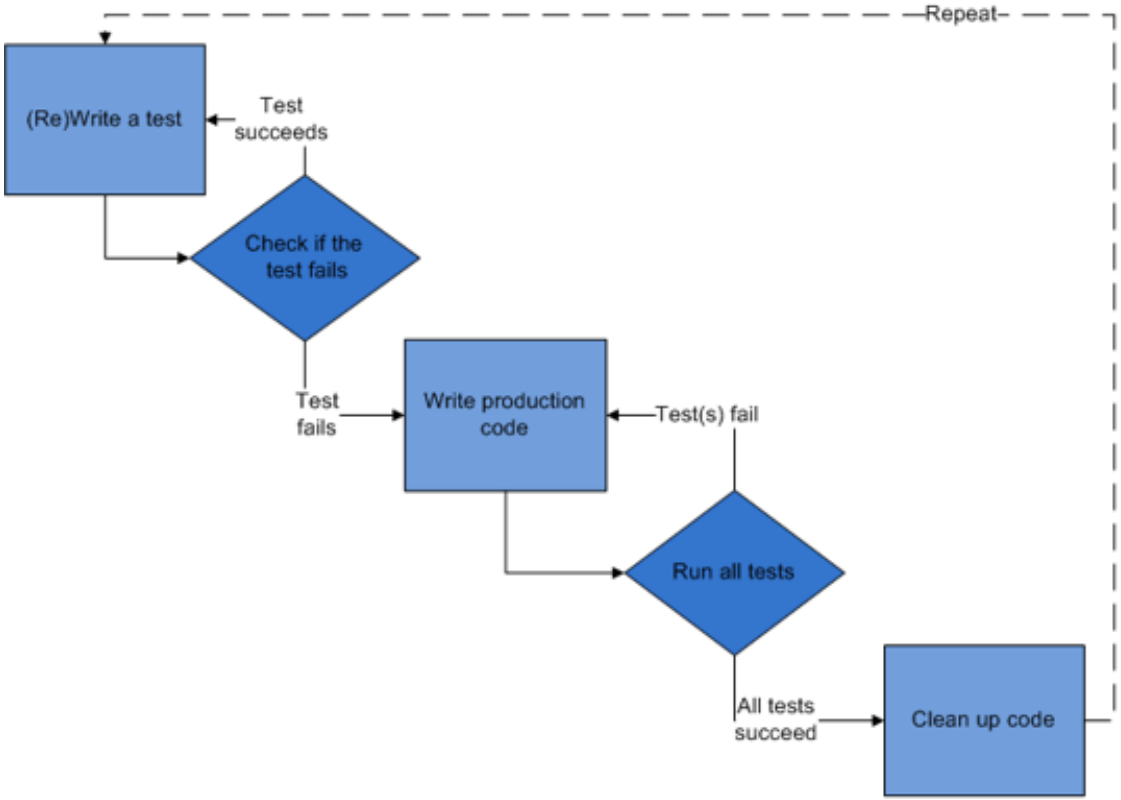
Henrik Kniberg, 2016, Source:

`https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp`

Extreme Programming (XP) Model



Source: `http://www.extremeprogramming.org/introduction.html`

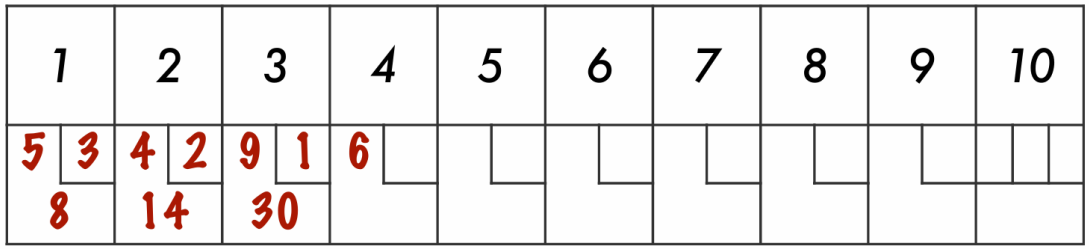


K. Beck (2003), Source: Test-driven development, wikipedia

Test Doubles used in TDD (and Unit Testing)

- Dummy
- Stub & Spy
- Fake
- Mock

Type	Purpose	Logic	Example
Dummy	Satisfy parameter requirements	None	Passing unused dependencies
Spy	Record interactions	Uses real object	Verify method calls and arguments
Fake	Lightweight substitute for real object	Simplified logic	In-memory database
Mock	Simulate behavior and verify interactions	Pre-programmed logic	Ensure a method was called a set number of times

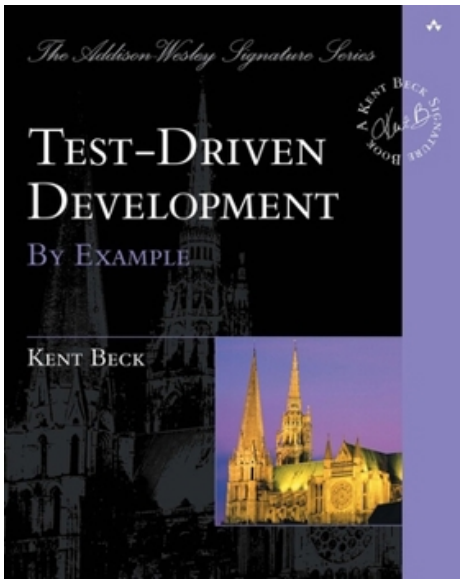


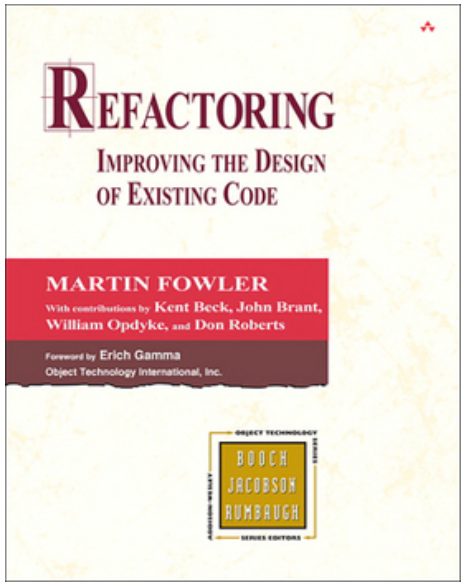
- Ten frames in each game
- Two balls thrown in (most) frames
- Bonus points for spares and strikes

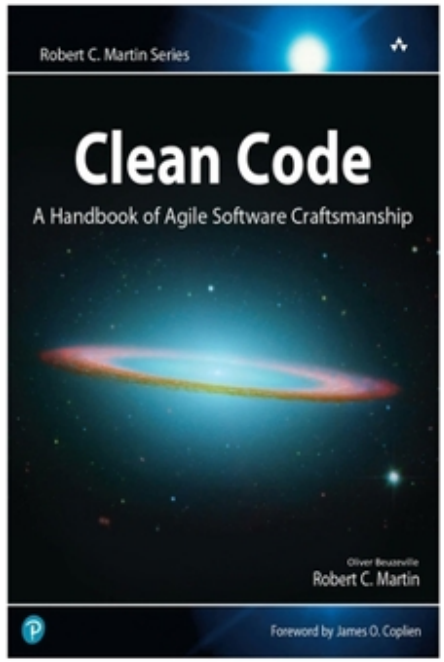
Eclipse Demo

Example based on [MM06]

Recommended Reading





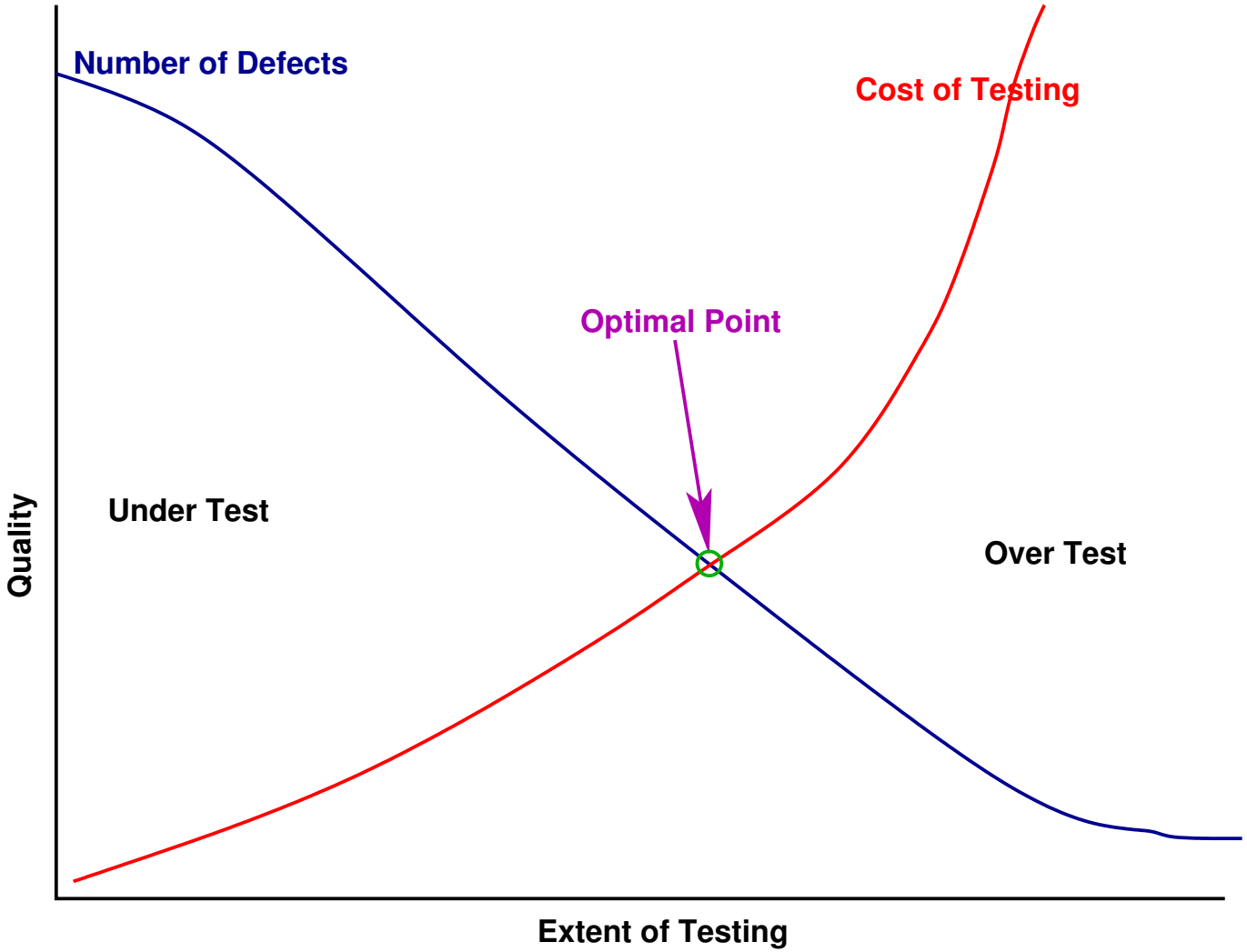


[Bec22, Fow18, Mar09]

How do you know the right way to do testing?

- Kind of software
- Stage of development
- Availability of reference models
- Availability and quality of specifications
- Skills of QA Team
- Availability of resources
- Novelty of product

- Constraints
  - The regulator *insists* we do MC/DC testing
  - *And* they won't let us build a protoype
- Costs
  - Exhaustive testing is almost always impossible
  - Effort alone is no guarantee



- Meet pre-agreed coverage goals
- Meet pre-agreed defect discovery rate
- Marginal cost
- Team consensus
- Management tells you to ship it

- Why am I taking this again?
- Studies show big increase in long-term recall
  - [YCMK24]
  - [LC11]

https://bit.ly/42otDlj

1. What are three types of software systems that need different testing?

Many possible answers, including

- Off-the-shelf products: apps for lots of public users
- Internal: stable environment, non-public users (and probably not so many either)
- Embedded: hardware constraints, high on optimization

2. What is 'expoloratory testing'?

Manual (dynamic) testing that is not based on specification. Creatively explore the system, based on a goal.

3. What is 'specification-based testing'?

Testing the system agains existing specification documents. Can be done through contract-based testing.

4. How is fuzz testing still useful if you don't know the correct output/response?

Even if you don't know the correct output, a system should handle failure with grace. Fuzz testing, when used to break a system, can check the reponse of the system on failure.

5. What efect does waterfall SDLC have on testing?

It makes it hard to test properly, as it bundles all testing activity at the very end of implementation

6. What's a good reason to stop testing?

Many possible answers. My personal recommendation would be: Meeting pre-approved coverage criteria and getting consensus among the team(s)

References I

- Kent Beck, Test driven development: By example, Addison-Wesley Professional, 2022.
- Martin Fowler, Refactoring: improving the design of existing code, Addison-Wesley Professional, 2018.
- Daniel Galin, Software quality assurance: from theory to implementation, Pearson Education India, 2004.
- Keith B Lyle and Nicole A Crawford, Retrieving essential material at the end of lectures improves performance on statistics exams, Teaching of Psychology **38** (2011), no. 2, 94–97.

References II


MH Miraz and M Ali, Blockchain enabled smart contract based applications: Deficiencies with the software development life cycle models. arxiv 2020, arXiv preprint arXiv:2001.10589.


Robert C Martin, Clean code: a handbook of agile software craftsmanship, Pearson Education, 2009.


Micah Martin and Robert C Martin, Agile principles, patterns, and practices in c, Pearson Education, 2006.

References III


Gautam Yadav, Paulo F Carvalho, Elizabeth A McLaughlin, and Kenneth R Koedinger, Beyond repetition: The role of varied questioning and feedback in knowledge generalization, Proceedings of the Eleventh ACM Conference on Learning@ Scale, 2024, pp. 451–455.