

# Final exam: practice problems

# Equivalence class testing

float **f** (float **x**, float **y**)

Problem 1: partitions for input domain

x: [-1.5, 0]  $\cup$  [0, 1.5]

y: [2.5, 3.5]

Input values for

- Weak/strong equivalence testing
- Weak/strong *robust* equivalence testing

# Equivalence class testing

float **f** (float **x**, float **y**)

Problem 2: partitions for input domain

x: [-1.5, 0]  $\cup$  [0, 1.5]

y: [0,2.5]  $\cup$  [2.5, 3.5]

Input values for

- Weak/strong equivalence testing
- Weak/strong *robust* equivalence testing

# Structural testing

```
read(x);  
read(y);  
if (x > 0 and y > 0) then  
    write (“both positive”);  
end if;  
if (x > 0) then  
    write (“x positive”);  
end if;
```

Input domain D

Test cases for complete

- statement coverage
- edge coverage
- condition coverage
- path coverage

# Symbolic execution

```
read(a); read(n);  
j := n;  
while a ≠ n do  
  j := j - a;  
  a := a + 1;  
end while;
```

Compute CFG

Path condition for exiting after two iterations?

# Review

# Software qualities

- Software is built to meet a certain functional goal and satisfy certain qualities
  - correctness, reliability, robustness, maintainability, performance, usability, verifiability, etc.
- Software processes must meet certain qualities
  - productivity, timeliness, visibility

# Software Engineering principles

1. Rigor and formality
2. Separation of concerns
3. Modularity
4. Abstraction
5. Anticipation of change
6. Generality
7. Incrementality

# Software production process

- Life cycle of a software product
  - requirements
  - architecture design and specification
  - coding and testing
  - delivery and deployment
  - maintenance and evolution
- Process models
  - Code&Fix -> Waterfall -> Waterfall with feedback  
-> Incremental delivery

# Rapid software development

- Iterative, incremental development process leads to *faster* delivery of *more useful* software
- Agile development
- Extreme programming

# Management

- Why needed: coordinate the activities and resources involved in projects
  - Planning, organizing, staffing, directing, controlling
  - “Plan the work and work the plan”
- Measuring productivity
- Tools for planning and monitoring

# Specification

- Specifications as contracts
- Specification qualities: Precise, Consistent, Complete, Incremental
- Operational specifications
  - Data Flow Diagrams, Finite State Machines, Petri nets
- Descriptive specifications
  - Entity-Relationship Diagrams, Logic/algebraic specifications

# Software architecture and design

- Architecture: high-level structure and organization of the system
- System decomposition into modules
- Module interfaces
- Main relationships among modules
- Information hiding

# Design

- Design notation: TDN/GDN
- Abstract Data Types
- Stepwise refinement

# Functional/Structural Testing

- WHITE BOX (structural) testing
  - tests *what the program does*
  
- BLACK BOX (functional) testing
  - tests *what the program is supposed to do*

# Program analysis

- Informal analysis: code analysis/  
walkthroughs
- Formal analysis: axiomatic semantics

# Symbolic execution

- Middle way between testing and analysis
- Executes the program on symbolic values
- One symbolic execution corresponds to many actual executions

# Tools and environments

- IDEs
- Static analysis
- Testing

Next class

**Project presentations**