Incremental Testing of Object-Oriented Class Structures
M. Harrold, J. McGregor, K. Fitzpatrick
1992
Presented by Dave Tahmoush

Object Oriented Programming
- Benefit from reuse of information-hiding modules
  - Called classes
  - A class has attributes
    - Data members or instance variables
    - Member functions or methods
  - Classes can be used to define new classes, or subclasses
    - Inheritance allows subclasses to use attributes from parent class
      - May also cancel attributes
      - Redefine attributes
      - Create new attributes
    - Would like to create libraries of tested classes to reuse
      - Completely repeating too expensive

Heirarchical Incremental Class Testing
- Reuse testing information from parent class
- Create testing history
  - Test suites for each attribute
- Incrementally update to guide testing of subclass
  - Inherit testing history, and update it
  - Automatically classify attributes
    - Test or not, or only partially test?
- Inheritance is guide to testing

Inheritance in Object-Oriented Systems
- Modifier M changes the attributes of parent class P to create new class R
- Incremental modification technique
- M contains attributes that alter class R
- Types of attributes in R
  - New, defined in M
  - Recursive, defined in P but available in R
  - Redefined, defined in P and changed in M
  - Virtual of all above types, specified but incomplete
- Examples on next slide

Example of inheritance, with P on left, M in the middle, and R on the right.
Note the examples of new, recursive, and redefined attributes

Inheritance can be thought of as incremental
- A is parent to B
- B is parent to C
- Thus only need to determine how to extend testing from parent to child, and can use recursively to test grandchildren, etc.
Heirarchical Incremental Class Testing

- Test base class
  - Test each member function
  - Test interactions among the member functions
  - Save test cases and execution information in a testing history
- Test subclass
  - Use testing history to avoid retesting when not necessary

Base Class Testing

- Test functions using traditional techniques
  - Specification based, or black box
    - (TS, test?)
  - Program based, or white box
    - (TP, test?)
- Test interactions among the member functions
  - Called integration testing
    - Focuses on interfaces between functions or units
    - IO format, format of entry or exit parameter values
  - Intra-class testing, when the functions are in the same class
    - (TIP, test?) or (TIS, test?)
  - Inter-class testing, when the functions are in different classes
    - Example is class Shape on next slides

Class Shape

Private:
- Point reference_point;

Public:
- void put_reference_point(Point); // access to data
- void move_to(Point); // defined to be erase() and draw()
- void erase(); // calls draw() to overwrite area
- void draw() = 0; // pure virtual – no implementation
- virtual float area(); // has an initial implementation
- shape(Point); // constructor
- shape(); // constructor

The testing history for class Shape. Note that there are two shape functions but their integration test cases are omitted for brevity.

move_to() calls erase() and draw()
erase() calls draw()

Subclass Testing

- Transform testing history from parent to child
- Modifications are analyzed to transform the testing history
  - New or Virtual-New functions fully tested
    - Use Y for full test with new test cases
  - Recursive or Virtual-Recursive functions not retested
    - Use N for no retesting
  - Integration tests run if interact with changed code (New or Redefined)
    - Use P for partially retested
  - Redefined or Virtual-Redefined functions fully tested
    - Use Y for retest, may reuse some test cases and build new ones
Algorithm for transforming a testing history from parent to child

---

EquiTriangle is a subclass that modifies Triangle

---

Experimentation

- Determine the savings using this technique
  - Compare the number of attributes to test
- Code to test
  - Base class Interactor has subclass Scene
  - Class Scene has subclass MonoScene
  - Class MonoScene has subclass Dialog
  - Data next slide

---

Conclusions

- Savings in the amount of testing
- Algorithmic approach may reduce time to analyze classes to determine what must be tested