ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Incremental Testing of Object-Oriented Software

Stéphane Barbey

Stephane.Barbey @ di.epfl.ch

Swiss Federal Institute of Technology in
Lausanne
Software Engineering Laboratory

June 1996

URL http://lglwww.epfl.ch/~barbey

Overview

Conditions for incremental testing?
Inheritances?

Some examples
    The creation operation

So what with polymorphism?

Conclusion

Conditions for Incremental Testing?

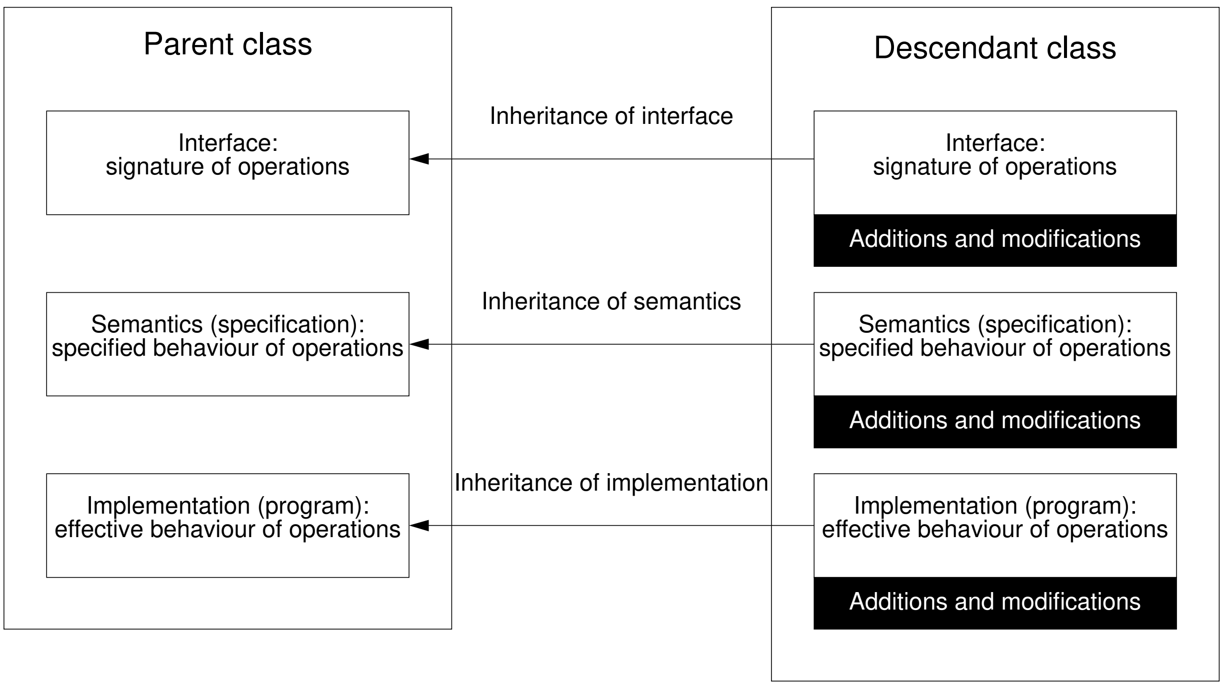
- the derived type is a true subtype of its parent type
- the program for the parent type satisfies its specification

Small Inheritance Example

```
Class Money
Inherit Float;
   Rename float -> money;
   Forget *, /;
End Money;
```

```
class Money
   inherit Float
   export “+”, “-”;
end -- Money
```


InheritanceS



Differences in Inheritance

Those three kinds of inheritance are orthogonal:

- Inheritance of interface
  Contravariance with same semantics and same implementation
- Inheritance of semantics
  An example of such practice is the construction of a class *Queue* based on part of the interface and on the implementation of a class *List*.
- Inheritance of implementation
  For example, a new implementation of an operation could be programmed in a descendant class to enhance its performance, although neither its interface, nor its semantics change.

The Traditional Test Process

function Test(P, SP) **return** (Yes, No, Undecided) **is**
begin
  (T, $\models_O$):= Generate_Test_Context (SP);
    -- perform the test set and the oracle generation
  Results := Execute_Tests (P, T, $\models_O$);
  return Compare (Results, Expected_Results);

exception
  when Undecidable_Error | Untractable_Error =>
    return Undecided;
end Test;

Incremental Testing

- Keep tests
  - for properties that have a new implementation
- Remove tests
  - tests that cannot be applied anymore
  - tests that are not useful
- Add tests
  - for situations that were not previously present

**Goal is to maximize removed tests,
and minimize added tests**

The Account Class

```
Class Account;
   Interface
      Use Money; Type account;
      Methods
         balanceOf _ : money;
         deposit _ : money;
         withdraw _ : money;
      Creation
         create _ : money
   Body
      Places
         balance: money;
      Axioms
         create m : -> balance m;
         balanceOf m: balance m -> balance m;
         deposit m: balance b -> balance b + m;
         (m <= b)= true => withdraw m:
            balance b -> balance b-m;
      Where
         m, b : money
End Account;
```

```
typedef int money;

class Account
{
      money balance;

   public:

      Account (money m);

      void deposit (money m);
      void withdraw (money m);
      money balanceOf ();
};
```

A Test Set for the class Account

$T_{Account}$ = {
<<a.create 100><a.deposit 50>true, true>,
<<a.create 100><a.deposit 50><a.deposit 50>true, true>,
<<a.create 100><a.deposit 50><a.withdraw 70>true, true>,
<<a.create 100><a.deposit 50><a.withdraw 200>true, false>,
<<a.create 100><a.deposit 50><a.balance 150>true, true>,

<<a.create 100><a.withdraw 50>true, true>,
<<a.create 100><a.withdraw 50><a.deposit 50>true, true>,
<<a.create 100><a.withdraw 50><a.withdraw 15>true, true>,
<<a.create 100><a.withdraw 50><a.withdraw 100>true, false>,
<<a.create 100><a.withdraw 50><a.balance 50>true, true>,
<<a.create 100><a.withdraw 150>true, false>,

<<a.create 100><a.balanceOf 100>true, true>,
<<a.create 100><a.balanceOf 100><a.deposit 100>true, true>,
<<a.create 100><a.balanceOf 100><a.withdraw 100>true, true>,
<<a.create 100><a.balanceOf 100><a.withdraw 150>true, false>,
<<a.create 100><a.balanceOf 100><a.balanceOf 100>true, true>}

Quick Account

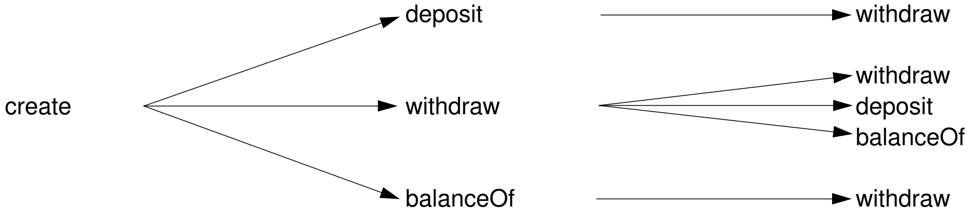
- No change of specification,
- Change of Implementation

```
Class Quick_Account;
    Inherit Account;
        Rename account -> quick_account;
End Quick_Account;
```

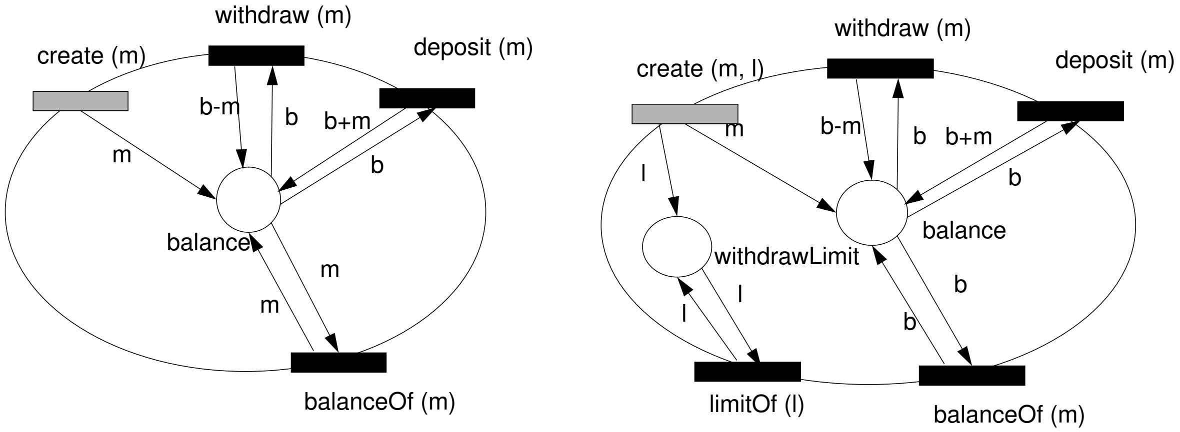
```
class Quick_Account: public Account
{
    public:
        void withdraw (money m);
};
```

Incremental testing requires a certain knowledge of the (specification of) the implementation.

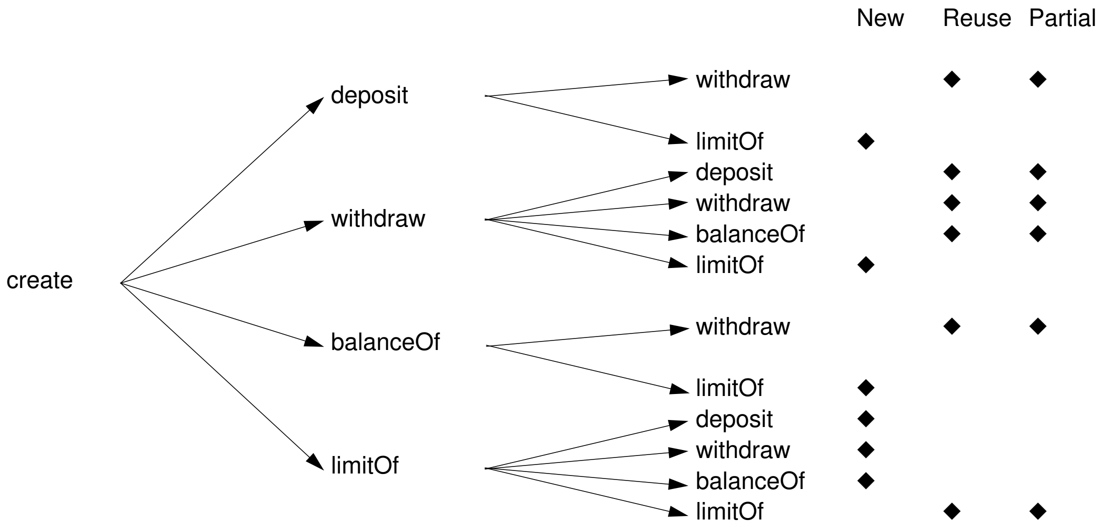
Abstract Execution Path for the Class Quick _Account



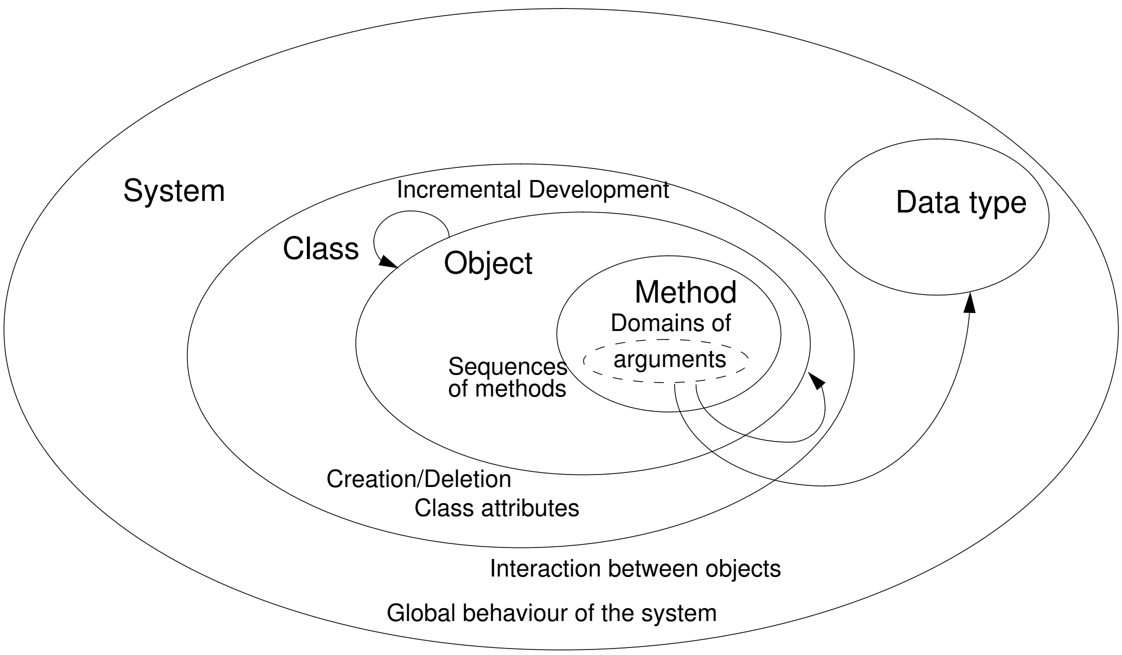
Specification of a Teen_Account Interface



Abstract execution path for the class Teen_Account (first three steps)



Architecture of Object-Oriented Systems



Creation operation are not on the same level

Creation Operations

- Creation operations are not inherited (Gott sei dank!)
- Therefore all paths are new!

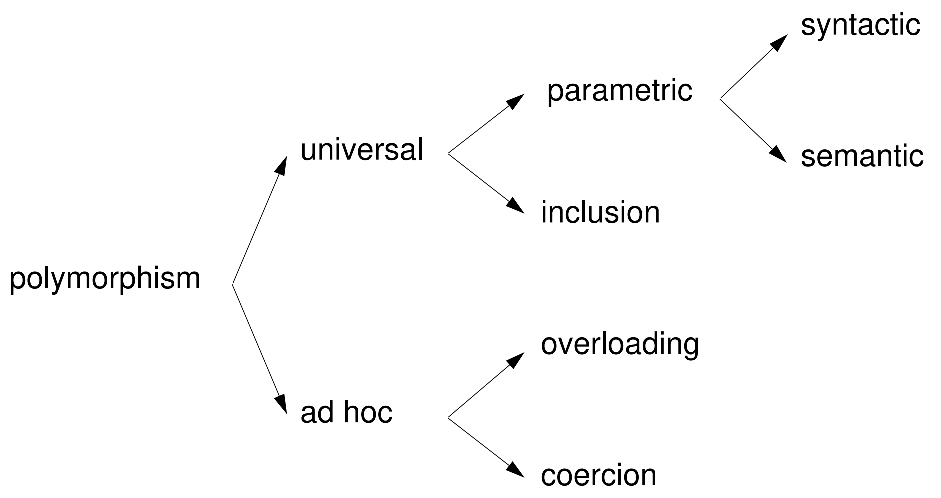
- Need for an abstraction function for creators, even for inheritance?

Example

- For Account
  <<a.create 100><a.withdraw 50><a.deposit 50>true, true>

- For Teen_Account
  <<a.create 100 40><a.withdraw 50><a.deposit 50>true, false>
  <<a.create 100 60><a.withdraw 50><a.deposit 50>true, true>

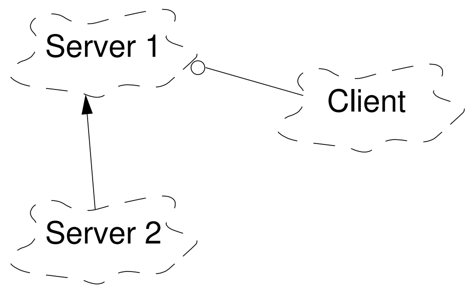
Varieties of polymorphism



Can inclusion and parametric polymorphism be treated the same way?

Polymorphism & Subtyping

What happens in the Client when Server2 objects can be substituted to Server1 objects?




```
Class Bank;
   Interface
      Use Money; Use Account;
      Type bank;
      Methods
         transfer _ _ _ : account, account, money;
      Creation
         ...
   Body
      Axioms
         (m < b) = true =>
         transfer source sink m
            with (source.balance b .. source.withdraw b) .. sink.deposit b:
               ->;
      Where
         source, sink : account;
         m, b: money
End Account;
```


When the Client and the Server are the Same

Redispatching (Calling “Self”)

```
class A
{
   public:
      void P1;
         { this->P2};
            // calls P2 according to the real
            // type of “this”
      virtual void P2;
         {...};
};
```

```
class A
{
   public:
      virtual void P2;
         {...};
};
```

- Same Interface
- Same Implementation
- Same Body
- *Different Behavior!*

Solutions for Polymorphism

Some possible strategies

- exhaustive testing
- least constraining
- random “à la McGregor”
- subtyping (with observers)
- observers (on the Client)

The Incremental Test Process

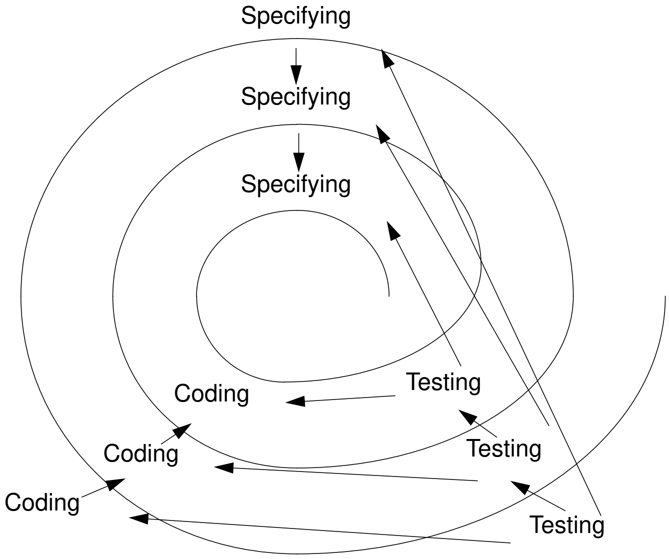
```
function Incremental_Test (Class, Parent_Tests)
  return (Yes, No, Undecided) is
begin
  Incremental_Context := Class.Interface+Class.Semantics+Class.Implementation;

  Reuse := Reusable_Tests (Parent_Tests, Incremental_Context);
  Delta := Difference (Class.Parent, Class);

  (New, ⊨o):= Generate_Test_Context (Delta);
     -- perform the test set and the oracle generation
  Results := Execute_Tests (P, {New, Reuse}, ⊨o);
  return Compare (Results, Expected_Results);

exception
  when Undecidable_Error | Untractable_Error =>
     return Undecided;
end Test;
```

Regression Testing in a Spiral Life-Cycle



What to (re-)test in descendants when the parent change?

Conclusion

- Incremental testing is also a matter of inheritance.
- Three orthogonal kinds of inheritance.
- The creation function causes problems.

- Polymorphism is a client/server concept
- ... but sometimes the client and the server are mixed.

- More needs for full regression testing