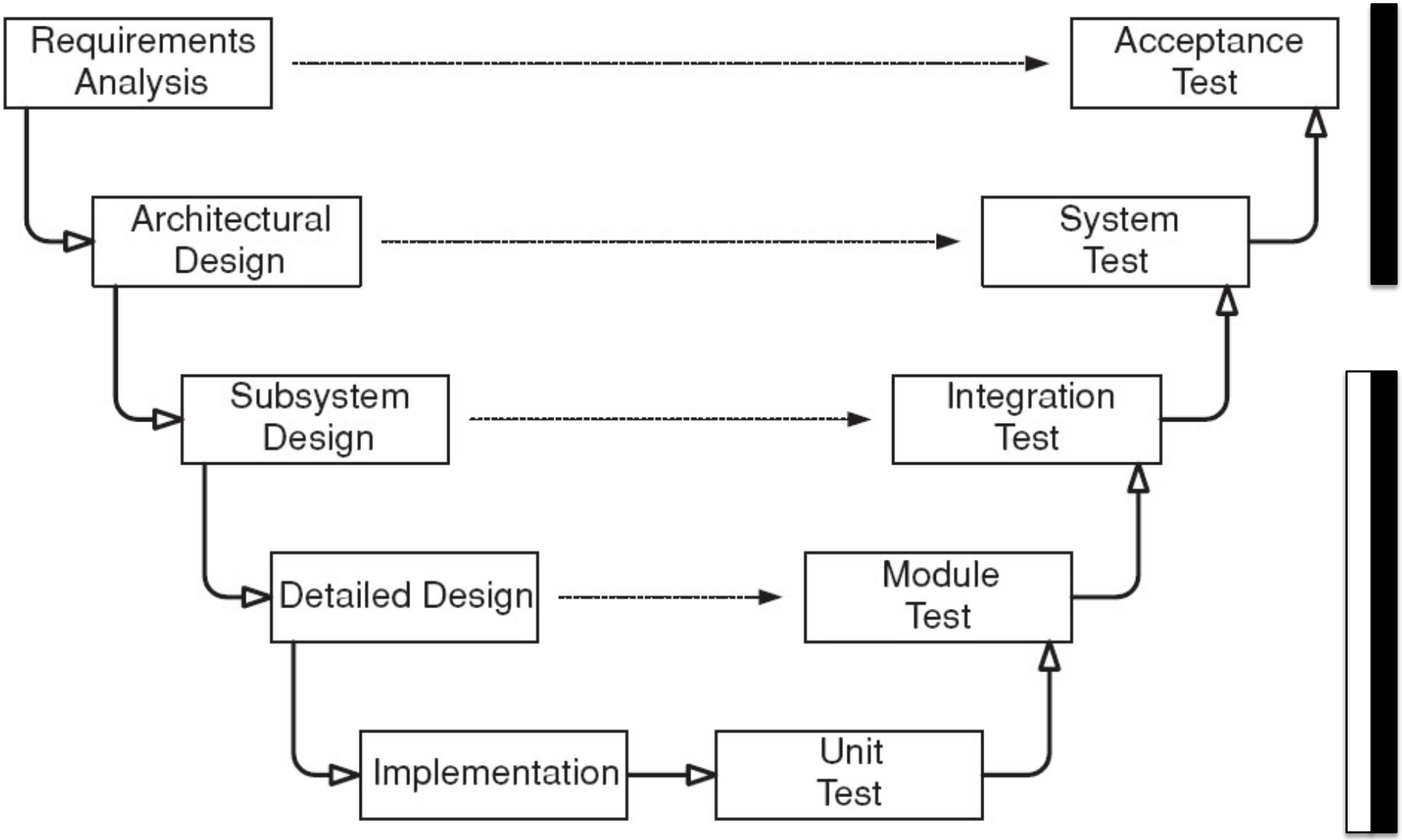
Functional Testing

Input space definition

Andrea Caracciolo

Testing scope



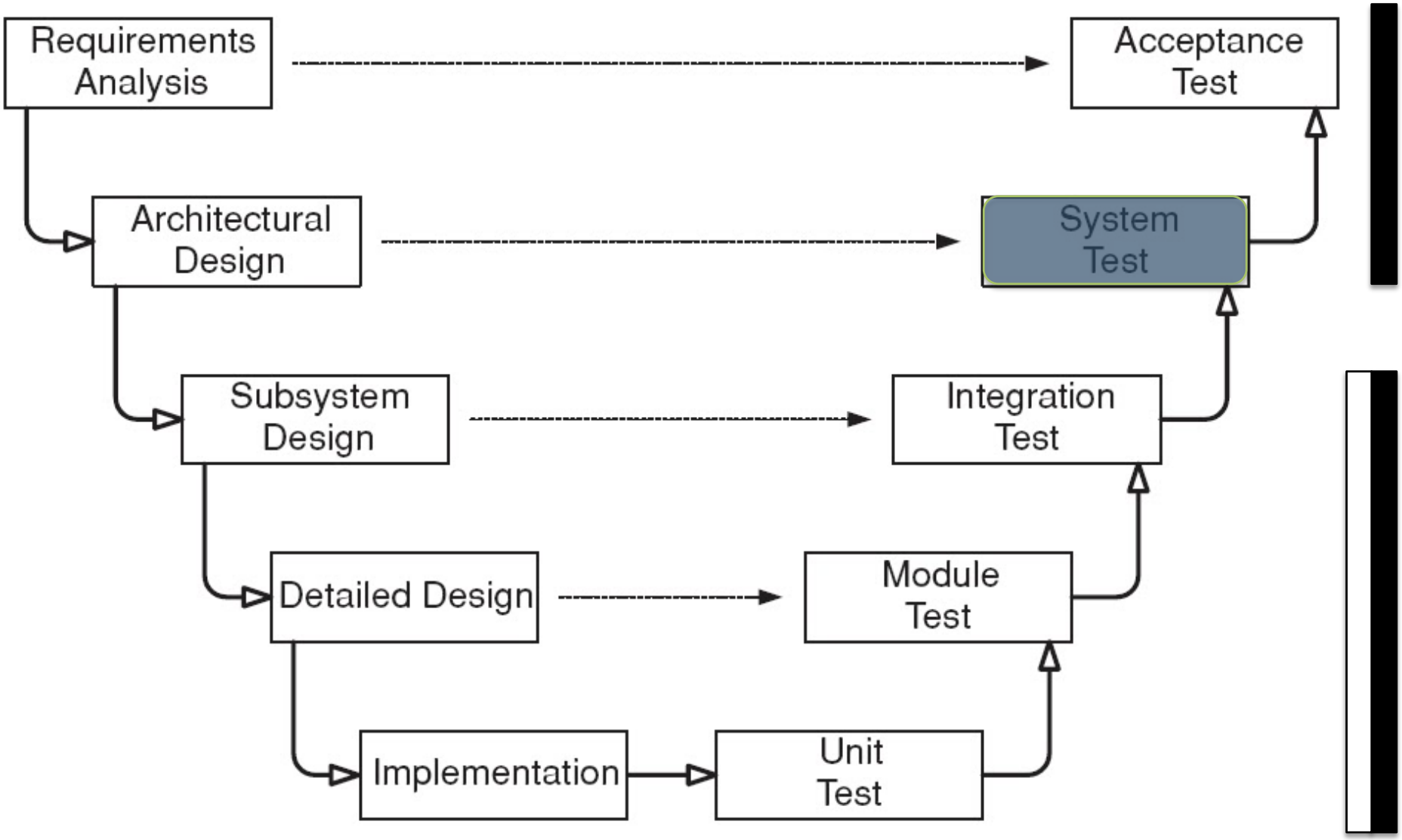
Testing approach

- White box testing
  - Statement coverage
  - Branch testing
  - Data-flow testing
  - …
- Black box testing
  - Equivalence partitioning
  - Boundary values analysis
  - …

Implementation driven
(bottom-up)

Specification driven
(top-down)

Testing scope



Functional testing

1. Identify parameters
2. Identify parameter characteristics
3. Identify representative values
4. Generate test-case specifications
5. Generate test-cases
6. Run test cases

- The maximal price of the bid must be >= the minimal price defined in the auction.
- The maximal price of the bid can be increased anytime, as long as the auction is not over.
- Two bidders can not have the same maximal price.
- If the n>0 bidders are sorted according to their maximal prices MP1 <= MP2 <= ... <= MPn, the current winner of the auction is the bidder n; the current selling price is defined as MP(n-1)+increment. The current winner and the current selling price is updated each time a new bid is placed or increased. If there are no bidder (n=0), there is no current winner nor current selling price.
- Buyers see the item minimal price, increment, current winner, and current selling price.
- When the current winner changes, the old winner receives an email.
- At the time the auction terminates and there was at least one bidder (n>0), the transaction proceeds in accordance with the current winner and the current selling price at that time. The winner receives a confirmation email. If there was no bidder (n=0), the auction simply closes.
- The highest bidder can't use the money they bid as long as the auction is not over. You need to ensure, I'm able to pay the bid.
- If several users interact with the system concurrently, a user might see and act on stale (outdated) data, e.g. stale selling price. The system must detect such situation (optimistic locking?) and inform users accordingly.

Auction feature

Story analysis

- Functionality
  - Creating an auction
  - **<u>Placing a bid</u>**
  - Terminating an auction

Parameters

1. **Identify parameters**
   - Independently testable features
   - Other elements of the environment on which the unit depends on
     - E.g. database, application state, ..

2. **Identify parameter characteristics**
   - Meaningful attributes for each parameter

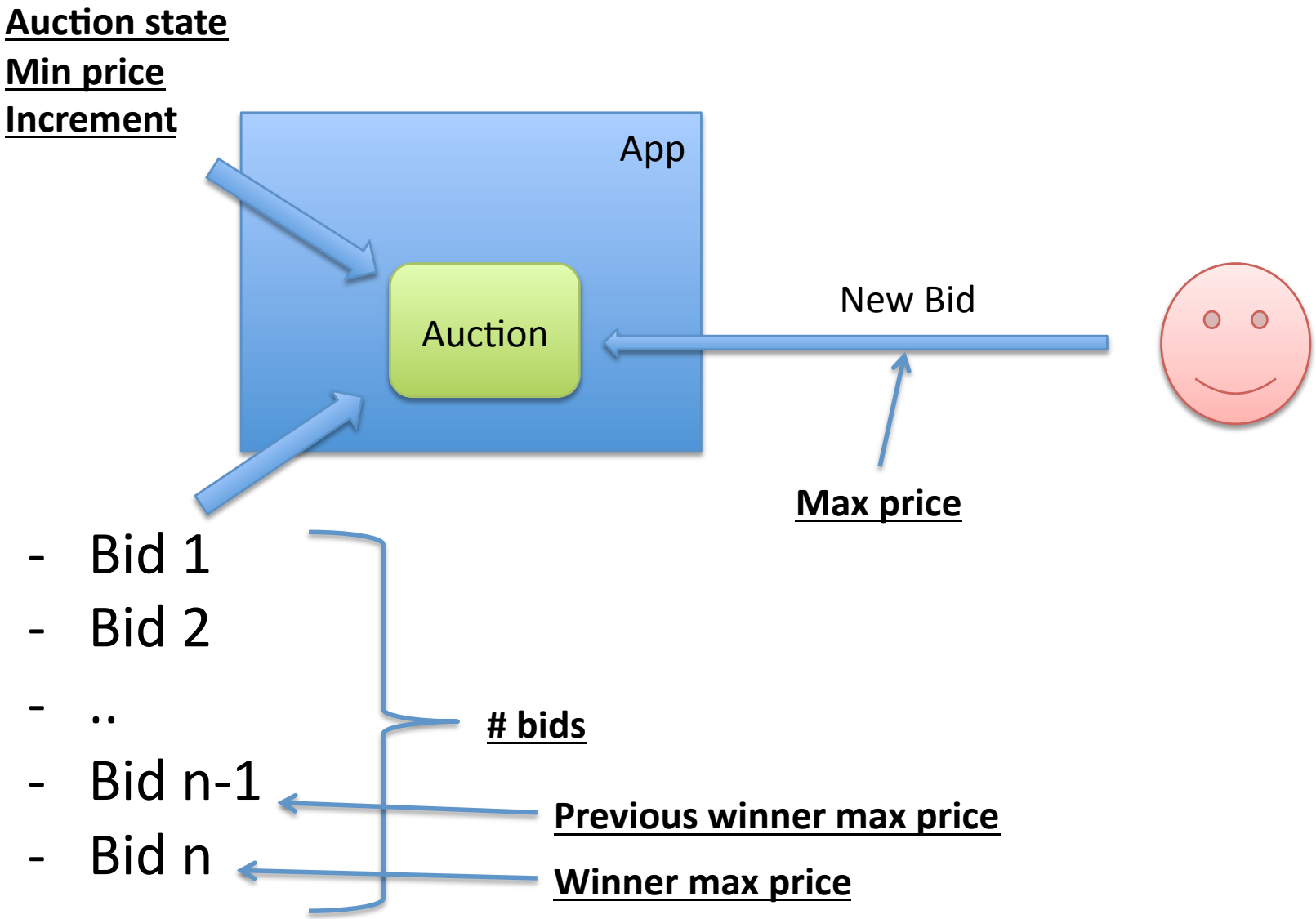
- The maximal price of the bid must be >= the minimal price defined in the auction.
- The maximal price of the bid can be increased anytime, as long as the auction is not over.
- Two bidders can not have the same maximal price.
- If the n>0 bidders are sorted according to their maximal prices MP1 <= MP2 <= ... <= MPn, the current winner of the auction is the bidder n; the current selling price is defined as MP(n-1)+increment. The current winner and the current selling price is updated each time a new bid is placed or increased. If there are no bidder (n=0), there is no current winner nor current selling price.
- Buyers see the item minimal price, increment, current winner, and current selling price.
- When the current winner changes, the old winner receives an email.
- At the time the auction terminates and there was at least one bidder (n>0), the transaction proceeds in accordance with the current winner and the current selling price at that time. The winner receives a confirmation email. If there was no bidder (n=0), the auction simply closes.
- The highest bidder can't use the money they bid as long as the auction is not over. You need to ensure, I'm able to pay the bid.
- If several users interact with the system concurrently, a user might see and act on stale (outdated) data, e.g. stale selling price. The system must detect such situation (optimistic locking?) and inform users accordingly.

- The maximal price of the **bid** must be >= the minimal price defined in the **auction**.
- The maximal price of the bid can be increased anytime, as long as the *auction is not over*.
- Two bidders can not have the same maximal price.
- If the n>0 bidders are sorted according to their maximal prices MP1 <= MP2 <= ... <= MPn, the current winner of the auction is the bidder n; the current selling price is defined as MP(n-1)+increment. The current winner and the current selling price is updated each time a new bid is placed or increased. If there are no bidder (n=0), there is no current winner nor current selling price.
- Buyers see the item minimal price, increment, current winner, and current selling price.
- When the current winner changes, the old winner receives an email.
- At the time the auction terminates and there was at least one bidder (n>0), the transaction proceeds in accordance with the current winner and the current selling price at that time. The winner receives a confirmation email. If there was no bidder (n=0), the auction simply closes.
- The highest bidder can't use the money they bid as long as the auction is not over. You need to ensure, I'm able to pay the bid.
- If several users interact with the system concurrently, a user might see and act on stale (outdated) data, e.g. stale selling price. The system must detect such situation (optimistic locking?) and inform users accordingly.

Parameters

- Bid
  - Max price
- Auction
  - Increment
  - Min price
  - # bids
  - Winner max price
  - Previous winner max price
  - Auction state

Parameters


Auction state
Min price
Increment
App
Auction
New Bid
Max price
- Bid 1
- Bid 2
- ..
- Bid n-1
- Bid n
bids
Previous winner max price
Winner max price

Values

3. Identify representative values

- Identify *equivalent partitions*

Example: Month

- Valid: 1 <= X <= 12
- Invalid: X < 1, X > 12

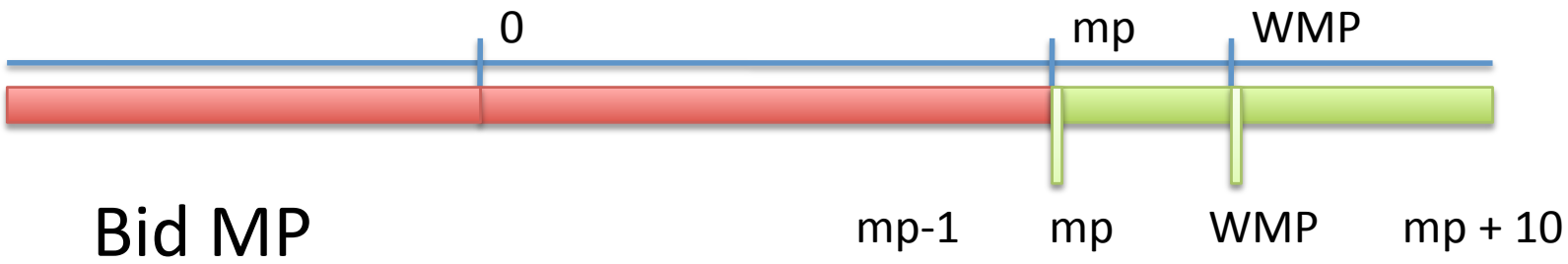
Values

- Bid
  - Max price (MP): < mp, = mp, > mp, = WMP
- Auction
  - Increment (inc): 0, Many
  - Min price (mp): 0, Many
  - # bids (bids): 0, 1, Many
  - Winner max price (WMP): NO, > LWMP
  - Previous winner max price (PWMP): NO, >= mp
  - Auction state (state): running, closed, not existing

Specification

4. Generate test-case specifications

- Check boundary values
  - A <= X <= B
    - Invalid: A – 1, B + 1
    - Boundary: A, B
    - Valid: A < n < B



Specification

- Bid
  - Max price (MP):
    - mp - 1 [error]
    - mp
    - mp + 10
    - WMP [error if bids > 0]

    - * [error if status != running]

Implementation

5. **Generate test-cases**
   - Identify input/output pairs
   - Write the corresponding code

Implementation

- Bid
  - Max price (MP):
    1. mp – 1
    2. WMP [if bids > 0]
    3. * [status != running]
       - Exception
       - Winner unchanged
       - #bids unchanged

    4. mp
    5. mp + n
       - Email to previous winner
       - Credit of new winner is “frozen”
       - #bids + 1

Output

Test::Unit

```
class MyTest < Test::Unit::TestCase
  class << self
    def startup
      puts 'runs only once at start'
    end
    def shutdown
      puts 'runs only once at end'
    end
  end

  def setup
    puts 'runs before each test'
  end
  def teardown
    puts 'runs after each test'
  end

  def my_test
    assert(true)
  end
end
```

Constant data
- Not test-case specific
- time-intensive operations

Data modified by test case
- Not test-case specific

Test suite

Test case

Implementation

- Model expected default state in startup/setup

```
def startup
    @auction = Auction.new
    @auction.setIncrement(1)
    @auction.setMinPrice(10)

    @user1 = User.new(..)
    @user2 = User.new(..)
end
```

Implementation

- Model expected default state in shutdown/ teardown

```
def teardown
    @auction.resetBids()
    @user1.resetBids()
    @user2.resetBids()
end
```

Implementation

- Model test case specific state in test case
- Test case: WMP [if bids > 0]

```
# TestCase 1.3
# Description: When multiple bids .....
def test_duplicatedWMP
        @user1.placeBid(@auction, 13)

        exception = assert_raise(InvalidBidError) {
                @user2.placeBid(@auction, 13)
        }
        assert_equal("Duplicated WMP", exception.message)
        assert_equal(1, @auction.bids)
        assert_equal(@user1, @auction.winner)
end
```

Write documentation

Choose meaningful names

Use exceptions

Check all the post-conditions

Mocking

- mock objects are simulated objects that mimic the behavior of real objects in controlled ways
- Why to use:
  - supplies non-deterministic results (e.g., the current time or the current temperature);
  - has states that are difficult to create or reproduce (e.g., a network error);
  - is slow (e.g., a complete database, which would have to be initialized before the test);
  - does not yet exist or may change behavior;
  - would have to include information and methods exclusively for testing purposes (and not for its actual task).

Mocking

- Rspec
  - http://rspec.info
- Mocha
  - https://github.com/freerange/mocha
- mocksmtpd
  - https://github.com/koseki/mocksmtpd