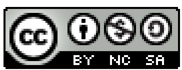
cc BY NC SA

Software Engineering

Lecture 03 – UML & Testing

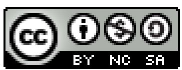
© 2015-20 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



UML & Testing

- UML/Modelling (Recap)
- (Unit) Testing



UML/Modelling (Recap)

- UML: Unified Modelling Language
- Used for visualization of system design
- ISO standard since 2000 (currently v2.5)
- Multiple diagram types (14!)
  - Structure diagrams
  - Behaviour diagrams
- (Sometimes) used to auto-generate code

CC BY NC SA

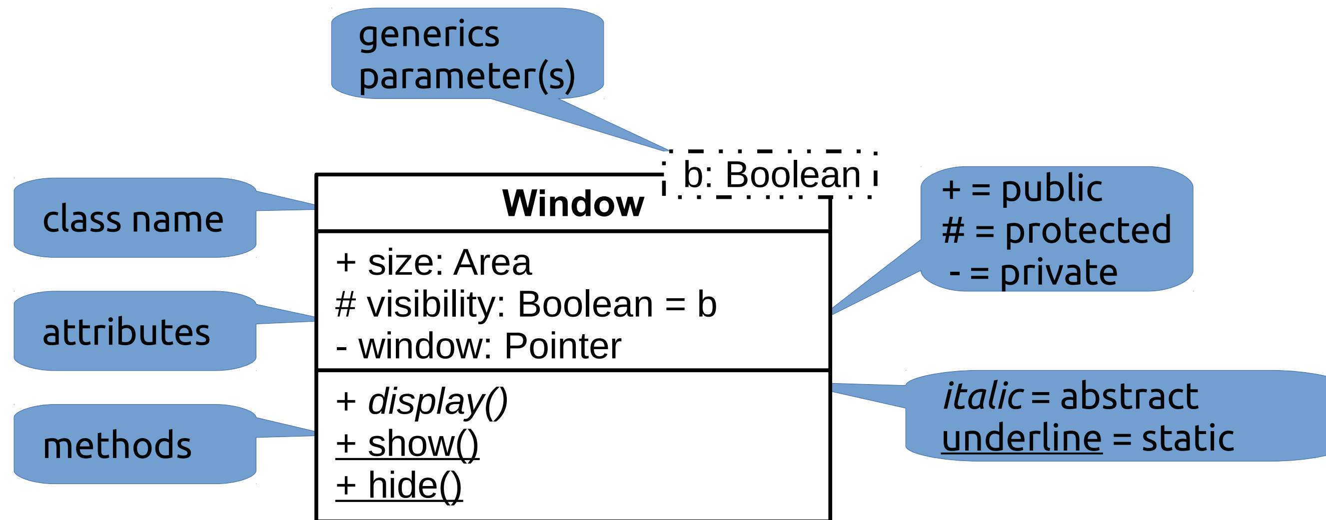
UML Diagram Types

- Tools
  - Violet: http://alexdp.free.fr/violetumleditor/
  - UMLet: http://www.umlet.com/
  - ArgoUML: http://argouml.tigris.org/
- 5 most common diagram types [ES2007]:
  - Structure: Class
  - Behaviour: Sequence, Use case, State, Activity
- Common to all diagram types: comments

CC BY NC SA

Class Diagram (1)

- Show properties of classes
  - Methods, attributes, visibility, scope



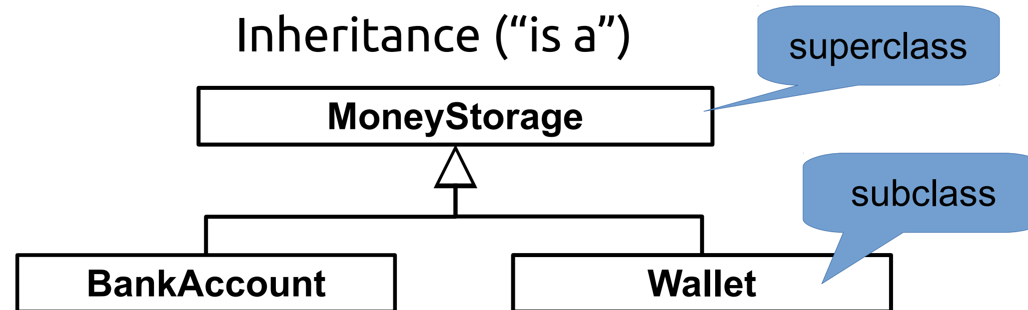
CC BY NC SA

Class Diagram (2)

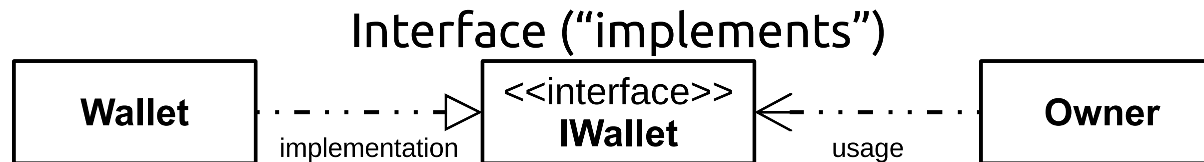
Image source (CC): https://en.wikipedia.org/wiki/Class_diagram

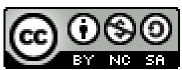
- Show relations between classes
  - Inheritance, implementation, …

Inheritance (“is a”)



Interface (“implements”)



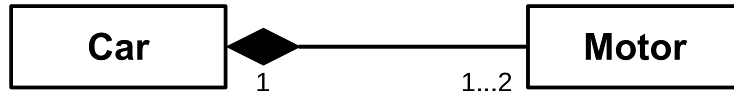


Class Diagram (3)

Image source (CC): https://en.wikipedia.org/wiki/Class_diagram

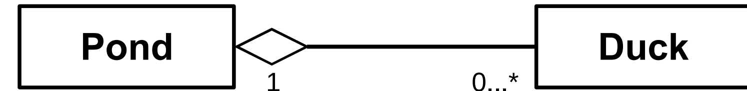
- Shows relations between classes:
  Composition, Associations, Multiplicity, …

Composition ("contains",
is destroyed with container)

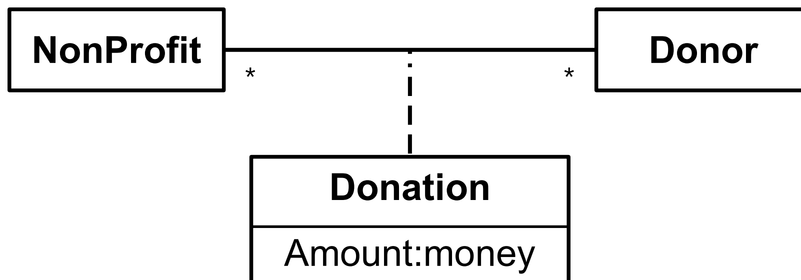


"1 Car contains 1-2 Motors"

Aggregation ("has",
can exist separately)



"1 Pond has arbitrary # of Ducks"



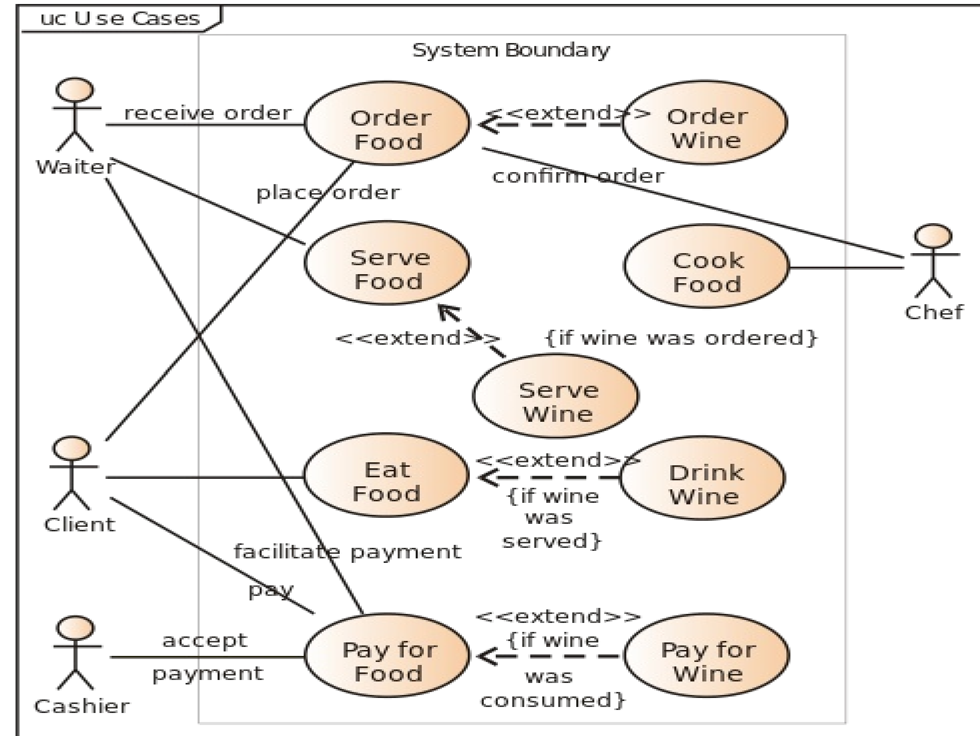
"Every Donor gives to arbitrary # of NonProfits, each NonProfit has arbitrary # of Donors, and each Donation has an individual Amount."

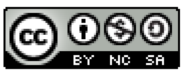
cc BY NC SA

Use Case Diagram

Image source (CC): https://en.wikipedia.org/wiki/Use_Case_Diagram

- Contains *actors* and *actions*
- Useful for communication with customers
- Mimics real world
- Less focus on system internals

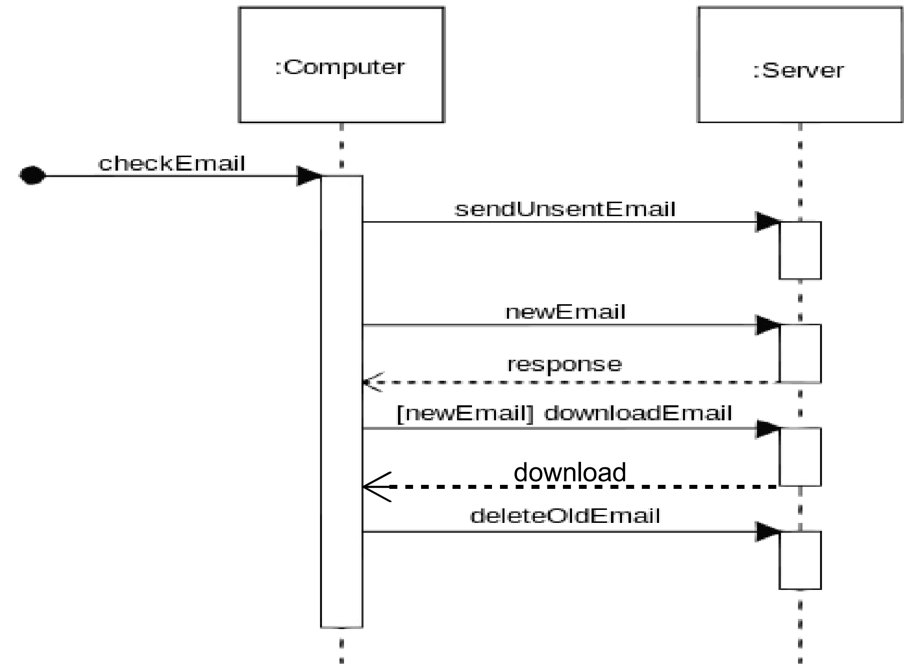




Sequence Diagram

Image source (CC): https://en.wikipedia.org/wiki/Sequence_diagram

- Shows *objects*, *lifelines* and *messages*
- Illustrates ...
  - runtime behavior
  - object lifetimes
  - (a)synchronous calls

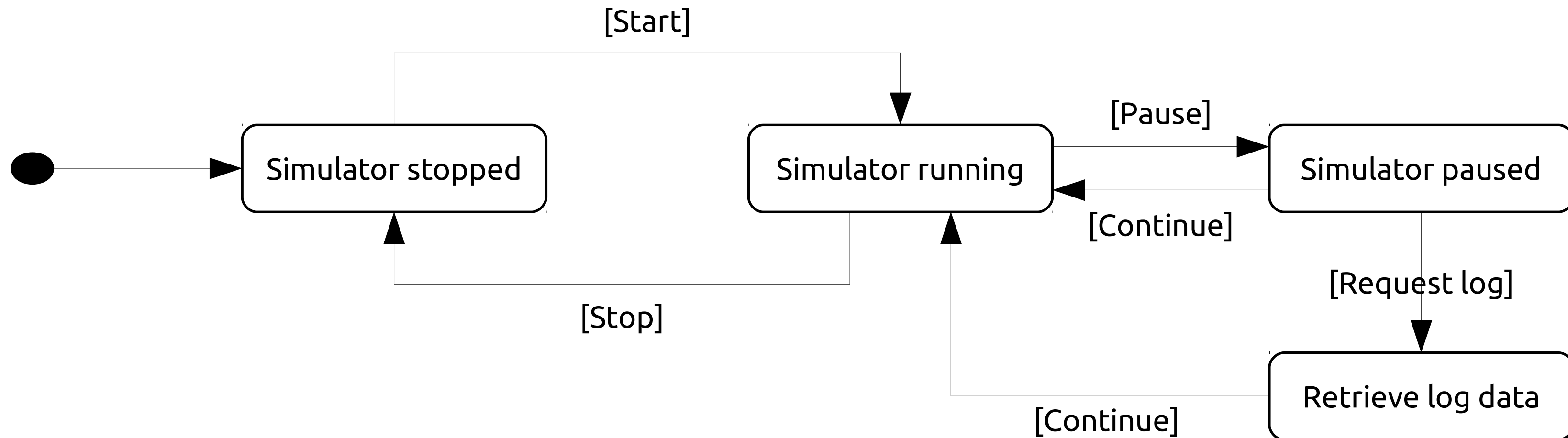


CC BY NC SA

State Diagram

Image source (CC): https://commons.wikimedia.org/wiki/File:UML_State_diagram.svg

- Contains *states* and *transistions*
- Transitions represent external events
- Start transition shows initial state



Activity Diagram

Image source (CC): https://en.wikipedia.org/wiki/Activity_diagram

- Shows actions, decisions and concurrency (black bars)
- Similar to flowchart

Conducting
[else]
[inexperienced participants in the group]
Explain problem
Warm-up practice
Present rules
Call for ideas
[idea(s) available]
[more]
Participants write down their idea
[no idea(s)]
[one]
Suggest lead
Everybody presents their idea in turn
Present idea
Record idea
[no time left]
[time left]
Elaborate on idea
[one]
[more]
[none]
Select most associated idea
Others write down their idea
[time left]
[no time left]
Wrap up

CC BY NC SA

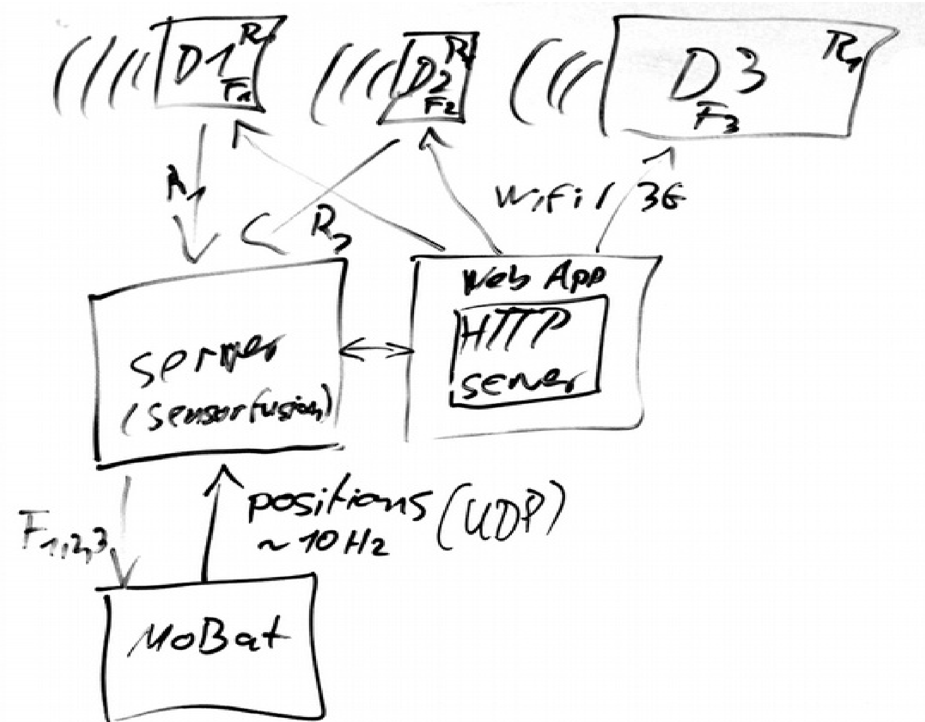
UML best practices

- Use for communication/documentation
- Try to keep diagrams small
- Code generation can be helpful …
- … but usually only for “boilerplate” code (class structure etc.)

CC BY NC SA

UML: alternatives?

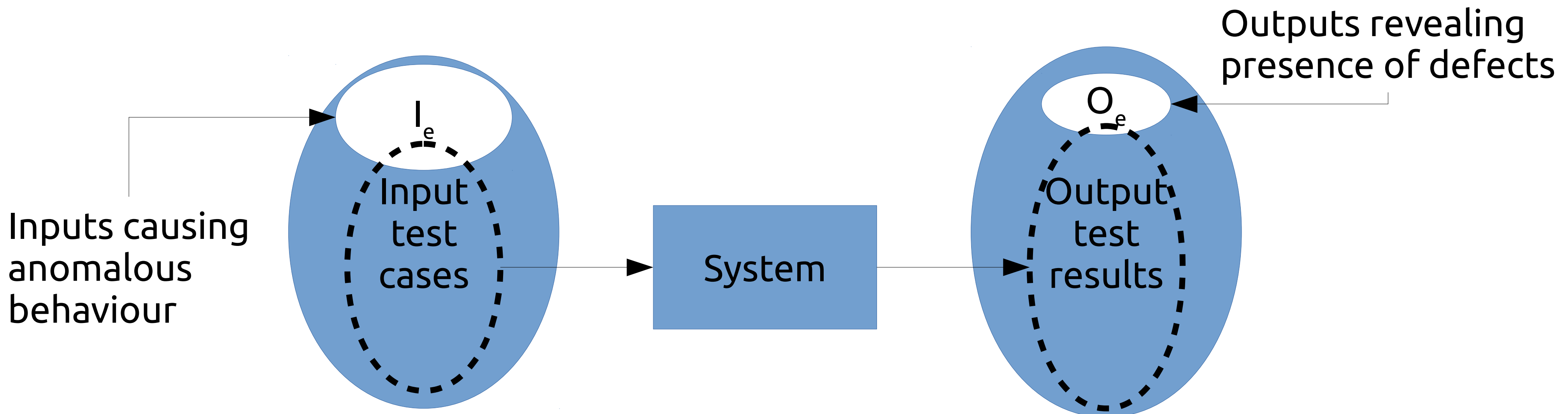
- If UML not strictly required: consider simpler “box-and-line” diagrams
- Most whiteboard sketches fall into this category :-)



cc BY NC SA

Testing

- Abstract: process test cases, check results
- However: tests can only show *presence* of errors, not *absence*.



cc BY NC SA

Testing (2)

- Validation testing
  - Show that software meets requirements
  - Test cases modelled after typical use cases
- Defect testing
  - Obvious goal: find bugs/errors/design flaws!
  - Test cases contain atypical/erroneous data

CC BY NC SA

Testing (3)

- Testing is possible at many levels/stages
- Development testing
  - unit testing
  - component testing
  - system testing
- Release testing
- Performance testing
- User testing

CC BY NC SA

Development Testing

- Performed *iteratively* during development
- Mostly performed by developers themselves (“white-box testing”)
- Independent *test developers* also possible (“black-box testing”)

CC BY NC SA

Unit Testing

- Core idea: test each *unit* of source code individually, e.g. each class
- Goal: test all methods, attributes, states
- Often requires *mock objects/test harnesses* to simulate missing system components
- Testing all *states* may require internal knowledge of the class – problem with black box testing

Component/System Testing

- Test building blocks consisting of multiple units/classes (or sub-blocks), also called *integration tests*
- Focus on interface between sub-units

- Possible types of interface error:
  - Interface misuse, e.g. parameters in wrong order
  - Interface misunderstanding – incorrect assumptions about behaviour of callee, e.g. passing unsorted array to binary search
  - Timing errors – components operate at different speeds → out-of-date information is accessed

"2 Unit Tests, 0 Integration Tests"

Image source (FU): https://www.reddit.com/r/ProgrammerHumor

CC BY NC SA

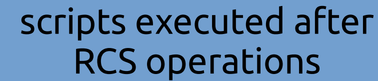
Testing automation

- Tests should (usually) be automated
- e.g. run tests directly after RCS commit
- Test frameworks provide structural support
- 3 phases:
  - Setup – initialize object/environment
  - Call – execute method
  - Assertion – check results
- Often grouped in *test suites*

cc BY NC SA

Testing: best practices

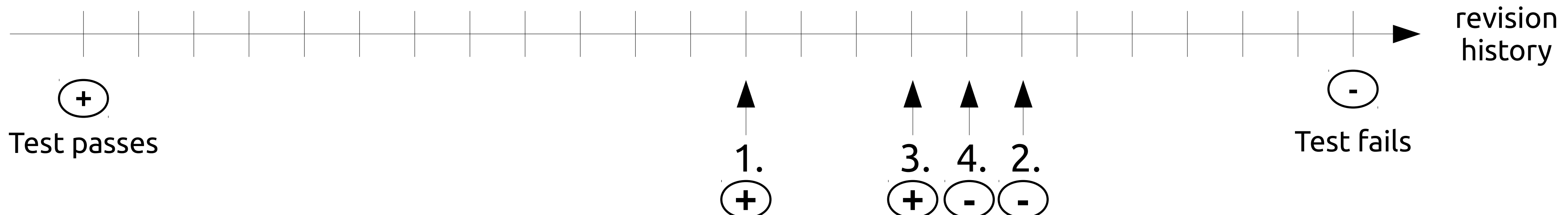
- Use a test framework (JUnit, CPPUnit, …)
- Automate your tests
- Combine with RCS
  - *Commit/push hooks* to trigger tests
  - *Bisection* to find errors in large changesets
- Use “extreme” test cases, e.g. NULL, NaN, -0, INT_MAX, empty set, …
- If at all possible: *write tests first!*



CC BY NC SA

Testing: best practices (2)

- Bisection: binary search in revision history
  - Identify initial “good” and “bad” commit
  - Test the one halfway between good and bad
  - Repeat until only one commit left
- Works best with small commits
- Example: 4th test identifies commit with error



CC BY NC SA

General best practices: Teamwork

- Follow the coding style guide
  - Use a tool like *indent*
- Use team tools
  - RCS
  - Issue tracker
  - Discussion forums
- *Never, ever send code by e-mail.*
- *Never, ever share code via remote folders.*

cc BY NC SA

Questions/suggestions?