Design by Contract in Python: Present and Future

Aaron Bingham
bingham@cenix-bioscience.com

What is DbC?

Big Idea:

Formally document client and provider responsibilities, and have the system automatically check the documentation against the implementation.

Why DbC?

- Documentation quality
  - precision and accuracy
- Implementation quality
  - DbC complements testing
  - Simpler code through clear responsibilities
  - Fewer defensive programming checks reduces code size and error rate
- Supports design and design-implementation transition

Principles

- Contracts are part of the system documentation
- Contracts are written as logical assertions about program behavior
- Contracts are verified automatically (usually at runtime)

Assertions: the 3 contractual forms

- Preconditions
  - Define client responsibilities
  - Checked before method execution
  - ORed with superclass preconditions
  - May only be *weakened* by subclasses
- Postconditions
  - Define provider responsibilities
  - Checked after method execution
  - ANDed with superclass postconditions
  - May be *strengthend* by subclasses
- Class invariants
  - Define class-internal consistency constraints
  - Checked before and after qualified (inter-object) calls
  - Checked after call even when exceptions are raised

Assertions: additional forms

- Loop invariants
- Loop variants: check for termination
- Checks: Equivalent to Python's **assert** statement.
- Useful, but have nothing to do with contracts per se

OLD

- Initial values of instance and arguments are saved.
- Allows checking for correct state transitions in postconditions

Example: a stack using Logilab Aspects

- Abstract data type (stack.adt)
- Abstract base class with contract (stack.py)
- Concrete implementation (stack.py)

Specifications and implementations

- Bertrand Meyer, Eiffel programming language
  - The original
- Terrence Way, PEP 316 / Contracts for Python
  - Uses contracts embedded in docstrings
- Logilab Aspects
  - Includes contract aspect
  - Broadly similar to PEP 316
- Reinhold Plösch, Design by Contract for Python
  - Paper, implementation not available
- Daniel Arbuckle, PyDBC
  - Uses metaclasses
- Dmitry Dvoinikov, IPDBC
  - Uses common base class

Implementation comparison

Feature	Eiffel	Contract	Aspects	Plösch	IPDBC	PyDBC
OLD	yes (3)	yes (1)	yes (2)	yes (2)	no	no
Return values	yes	yes	yes	?	yes	no (4)
Parameters in postcondition	yes	yes	yes	yes	yes	no (4)
Precondition strengthening	yes	yes	yes	?	no	no
Violations raise exceptions	yes	yes	yes	no (7)	yes	yes
Contracts visible in docs	yes	yes	yes	yes	no	no
Private assertions hidden in docs	yes	no	no	no	n/a	n/a
Named assertions	yes	no	no	no	no	no
Private attribute names	n/a	no (4)	no (4)	?	yes	yes
Module and function contracts	n/a (5)	yes	yes	no	no	no
Integrated type checking	n/a (6)	no	no	yes	no	no

(1) Shallow copies of explicitly listed values
(2) Deep copies
(3) Copy depth depends on storage declarations
(4) Support could be added relatively easily
(5) Every function and every variable in Eiffel must be part of some class
(6) Eiffel is statically typed
(7) Violations are logged to a file

Conclusions: state of implementations

- A solution using decorators would be interesting for comparison
- Only Contract and Aspects are workable solutions right now
- Both have similar bugs in inheritance handling; both should be fixable.
- Both need support for transparent private-attribute name-mangling
- Aspects needs to at least indicate the line number of the failing assertion

Conclusions: required support

- Documentation tools: hide assertions involving private and protected attributes
- Need a tool to control contract checking at package, module, class, and method level without editing affected module
- We're not far off!

References

- Arbuckle, Daniel; PyDBC; http://www.nongnu.org/pydbc/
- Dvoinikov, Dimitry; Yet Another Design by Contract Module for Python; http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/436834
- Logilab; Aspects; http://www.logilab.org/projects/aspects
- Meyer, Bertrand; Object-Oriented Software Development, 2nd Edition
- Plösch, Reinhold; Design by Contract for Python; http://www.google.de/url?sa=U&start=1&q=http://www.swe.uni-linz.a
- Way, Terrence; Contracts for Python; http://www.wayforward.net/pycontract
- Way, Terrence; PEP 316 – Programming by Contract for Python; http://www.python.org/dev/peps/pep-0316/