Design By Contract

Mincom Technology Talk
August 2011
Mark Perry

About Me

Mark Perry
Senior Dialog Consultant

Principle areas of interest:

- Architecture
- Development
- Processes

Outline

Assumptions

Introduction
History
Elements
Exceptions
Java
Related Work

Introduction

Design By Contract:

- Approach to design and implementation
- Define formal, precise specifications
- Uses metaphor of business contracts
- Preconditions, postconditions and invariants to classes

Contract Principles

- Binds two or more parties: supplier and client
- Explicit
- Specifies mutual obligations and benefits
- No hidden clauses
- Relies on general rules to contracts (laws, regulations, standard practices, etc.)

Motivation

- Help produce correct software - specification with code
- Clear responsibility of concerns between client and service
- Basis for reasoning or proof of software
- Basis for testing, debugging, self checking and exception handling
- Software documentation
- Higher reuse through more reliable software
- Inheritance
- No 1 Java Request for Enhancement (RFE) (http://bugs.sun.com/bugdatabase/top25_rfes.do)

Example Contract

- Australia Post
- Guaranteed Delivery Contract

Party	Obligations	Benefits
Client	Provide package of less than specified weight and size and pay money.	Package delivered within x hours.
Supplier	Deliver package within x hours.	Package is not too big, heavy or unpaid.

Industrial Vat Example

- See vat.e

Influences

- Based on axiomatic semantics in late 60s and early 70s (Robert Floyd, Tony Hoare and Edsger Dijkstra)
- Influenced by formal specification languages (Z, VDM, Object Z, etc.)
- Eiffel first mainstream programming language with contracts (1986, Bertrand Meyer)
- Other languages
  - Nana (C++), Cofoja, JML, Contractor4J (Java), Spec#, Code Contracts (.NET), D, Racket, Ada, Groovy, JavaScript, Lisp, Perl, Python, Ruby and others

Hoare Logic

- Hoare Triple
  - {Pre} Command {Post}
- Simple imperative language
  - Guarded Command Language (GCL)
- Axioms and inference rules for development

GCL Specification

Integer Square Root (rounded down)

```
|[ con N: int { N >= 0};
    var x: int;
    square_root
    {x^2 <= N and (x + 1)^2 > N}
]|
```

GCL Program Refinement

Integer Square Root (rounded down)

```
|[ con N: int { N >= 0};
    var x: int;
    x := 0;
    {invariant: 0 <= x and x^2 <= N, bound: N – x^2 > 0}
    do (x + 1) * (x + 1) <= N → x := x + 1; od
    {x^2 <= N and (x + 1)^2 > N}
]|
```

Abstract Data Types (ADT)

- Mathematical model of data types and their semantics
- Defined by operations and effects of operations
- Stack ADT operations: new, empty, push, pop, top

Stack ADT

Type

- Stack

Functions

- new: Stack[T]
- empty: Stack[T] → Boolean
- push: Stack[T] X T → Stack[T]
- pop: Stack[T] → Stack[T]
- top: Stack[T] → T

Stack ADT (2)

Preconditions

- pop: not empty
- top: not empty

Axioms

- empty(new)
- not empty (push(s, x))
- top(push(s, x)) = x
- pop(push(s, x)) = s

Contract Elements

Precondition

Postcondition

Class Invariant

Assertion

Loop Invariants and Variants

Class Correctness

- When is a class correct?
  - Instantiation: {PRE} BODY {POST and INV}
  - Methods: {PRE and INV} BODY {POST and INV}

Loop Invariants and Variants

Loops common source of defects

- Off by one
- Edge cases
- Infinite loops

Loop elements

- Guard
- Postcondition
- Invariant property that generalises Postcondition
- Bound on iterations (variant)

Inheritance

- Class Invariant of child class is conjunction of the class's invariant and all ancestors' invariants
- Method Overriding
  - Precondition kept or weakened; disjunction of parent's precondition
  - Postcondition kept or strengthened; conjunction of parent's postcondition
- Behavioral subtyping - similar to contravariant parameters and covariant return types

Runtime Monitoring

- Eiffel allows runtime configuration of contracts
  - Particular cluster (package)
  - Contract level (none, require, ensure, invariant, loop, check, all)

  How much monitoring?
  - Debugging – all
  - Production - require

Arianne 5 Disaster

- Rocket delivering payloads into low Earth orbit
- Exception 37 seconds into flight (in Ada) not processed; rocket self destructed
- $500 million - uninsured
- Exception caused by incorrect conversion: 64 bit real converted to 16 bit int
- Analysis had “proven” exception could not occur; the horizontal bias could always be represented as a 16 bit int
- Error not caused by: design, implementation, language, testing, quality assurance
- Reuse error
  - Analysis correct for Arianne 4
  - Assumption in design document

Eiffel Exceptions

- Failure
  - An operation cannot fulfill it's contract
- Exception
  - Undesirable event during execution of method
- Exception Handling
  - Concede failure and trigger exception in client
  - Retry

Exception Mechanisms

- Rescue
  - Handle exception
- Retry
  - Performed within rescue clause
  - Perform method again
- See exception_example.txt

Exception Correctness

- Instantiation
  - {PRE} BODY {INV and POST}
- Normal method
  - {INV and PRE} BODY {INV and POST}
- Exception
  - {True} RESCUE {INV}
- Transaction Comparison

Contracts For Java

- Cofoja, a Google 20% project
- Release Feb 2011
- Class invariants, preconditions and postconditions, exceptions implemented as errors
- Mechanisms: annotation processing, bytecode instrumentation, non intrusive, separate contract compilation, type safe contracts, standard compiler

Cofoja Stack

- See java source
- Contracts, inheritance, runtime monitoring

Cofoja Process

- See compilation and deployment process diagram, p5

Contract Lifecycle

- See execution of annotation processor, p9

Motivation Review

- Help produce correct software - specification with code
- Clear responsibility of concerns between client and service
- Basis for reasoning or proof of software
- Basis for testing, debugging, self checking and exception handling
- Software documentation
- Higher reuse through more reliable software
- Inheritance

Related Fields

- Program verification
- Formal specification and verification
- Specification based testing
- Hoare logic and program refinement
- Type Theory
- Concurrency

References

Meyer: Object Oriented Software Construction, 1997.

Meyer: Applying Design By Contract.

Harper: Practical Foundations of Programming Languages, 2010, http://www.cs.cmu.edu/~rwh/plbook/book.pdf.

Walden and Nerson, Seamless Object-Oriented Software Architecture, 1994, http://www.bon-method.com.

Pierce, Types and Programming Languages, 2002.

Pierce, Advanced Topics in Types and Programming Languages, 2005.

Poernomo, Adapting the Proofs-as-Programs to Imperative SML,
http://www.cs.cornell.edu/Nuprl/PRLSeminar/PRLSeminar03_04/Poernomo/iman_nuprl_seminar.pdf

Sorensen and Urzyczun, Lectures on the Howard-Curry Isomorphism

Altenkirch, Danielsson, Loh and Oury: Dependent Types Without the Sugar

Hanks: Structured Propositions as Types.

Guevers: Introduction to Type Theory

Pfenning: Lecture Notes on Proofs as Programs

Wadler: Proofs are Programs: 19th Century Logic and 21st Century Computing

Awodey: Propositions as [Types]

Altenkirch, McBride and McKinna: Why Dependent Types Matter

References (2)

Hinze, Jeuring and Loh: Typed Contracts for Functional Programming

Barnett, Muller, Fahndrich, Schulte, Leino and Venter: Specification and Verification: The Spec# Experience

Aberhold: Second Order Programs with Preconditions

Guha, Matthews and Findler: Relationally-Parametric Polymorphic Contracts

Rieken: Design By Contract for Java – Revised

Leavens, Baker, Ruby: Preliminary Design of JML: A Behavioural Interface Specification Language for Java

Nanevski, Ahmed, Morrisett and Birkedal: Abstract Predicates and Mutable ADTs in Hoare Type Theory

Findler and Felleisen: Contracts for Higher-Order Functions

Antoy and Hanus: Contracts and Specifications for Functional Logic Programming

Xu: Static Contract Checking for Haskell