Assertions, Exceptions, and Module Stability

John Beresniewicz

Oracle Corporation

Agenda

- Design by Contract
- Assertions
- Exceptions
- Modular code
- PL/SQL construction techniques

Design by Contract

Design by Contract is a powerful metaphor that... makes it possible to design software systems of much higher reliability than ever before; the key is understanding that reliability problems (more commonly known as bugs) largely occur at module boundaries, and most often result from inconsistencies in both sides' expectations.

Bertrand Meyer, Object Success

Contract elements

- PRECONDITIONS
  - *What will be true when module is entered?*
- POSTCONDITIONS
  - *What will be true when module completes?*
- INVARIANTS
  - *What will not be changed by module execution?*

PL/SQL and Design by Contract

- Errors occur at module interfaces
- Design by Contract = formalizing interfaces
  - *IN values must obey preconditions*
  - *OUT and RETURN must satisfy postconditions*
- Exceptions as an invariant violation (system state change)

Assertions and contracts

Good contracts are those which exactly specify the rights and obligations of each party...In software design, where correctness and robustness are so important, we need to spell out the terms of the contracts as a prerequisite to enforcing them. Assertions provide the means to state precisely what is expected from and guaranteed to each side in these arrangements.

Bertrand Meyer,
Object-Oriented Software Construction

Basic module structure

- ASSERT
  - *Preconditions*
- COMPUTE
  - *Small modules with clear contracts*
- RETURN
  - *Function-oriented*

PL/SQL assertions

- Test boolean and signal if FALSE
- Implement in PL/SQL as a procedure

```
PROCEDURE assert (cond_IN IN BOOLEAN);

assert(parm1 BETWEEN 0 AND 100);
assert(plsqltbl.COUNT > 0);
assert(vbl2 IS NOT NULL);
assert(fcnX > constantY);
```

Simplest assert procedure

```
PROCEDURE assert (cond_IN BOOLEAN)
IS
BEGIN
   IF NOT NVL(cond_IN,FALSE)
      THEN
         RAISE ASSERTFAIL;
      END IF;
END assert;
```

- NULL tests FALSE and raises exception

Assertion failures are bugs

- Precondition assertion failure signals error in client module
- Postcondition assertion failure signals error in server module

Turning off assertions

- Comment out but leave in code
- Suppress for performance issue only

```
FUNCTION calledoften
   (p1 varchar2, p2 integer) RETURN BOOLEAN
IS
BEGIN
 -- assert(LENGTH(p1) BETWEEN 10 AND 100);
 -- assert(BITAND(p2,3) = 3);
    /* code for module... */
END calledoften;
```

Exceptions

...whenever available, a method for engineering out failures is preferable to methods for recovering from failures.

Bertrand Meyer,
Object-Oriented Software Construction

Catching an exception on purpose

```
FUNCTION IsNumber (txt_IN IN varchar)
 RETURN BOOLEAN
IS
   test  NUMBER;
BEGIN
   BEGIN
      test := TO_NUMBER(txt_IN);
   EXCEPTION
      WHEN VALUE_ERROR THEN null;
   END;
 RETURN (test IS NOT NULL);
END IsNumber;
```

- Exception provides the essential information

Best practice: smart scoping

- WHEN an Oracle exception can be anticipated in a section of code,
- AND that exception can be safely handled,
- THEN enclose the code in a sub-block and handle the exception (and only that exception)

Best Practice: declare safely

- Initialize declarations safely
- DO NOT initialize variables at declaration with function calls

```
PROCEDURE willnotfail IS
    localvar INTEGER;
BEGIN
    localvar := initfunction;
EXCEPTION
    WHEN OTHERS THEN null;
END willnotfail;
```

Worst practice: catch and ignore

```
FUNCTION badfcn(p1_IN integer)
  RETURN BOOLEAN IS
BEGIN
   /* some code */
EXCEPTION
   WHEN OTHERS THEN RETURN null;
END badfcn;
```

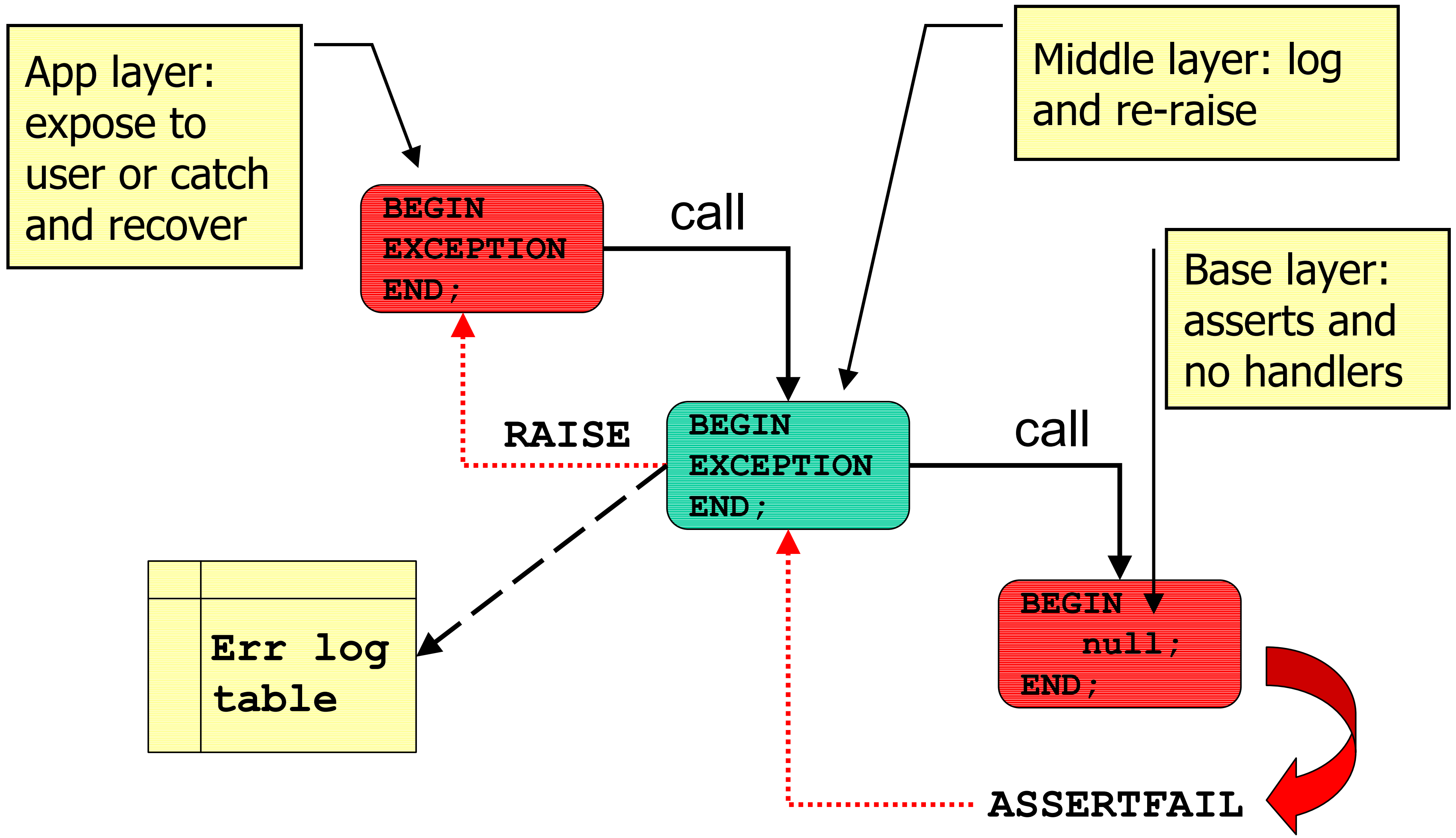
- Masks errors and thus lies to callers
- Returns BOOLEAN with NULL value

Catch, cleanup and re-RAISE

```
EXCEPTION
   WHEN OTHERS
   THEN
      log_error(SQLCODE);
      /* local clean up
            (e.g.close cursors) */
      RAISE;
```

- Log errors and clean up
- Re-raise exceptions to caller
- “Dead programs tell no lies”
  – *The Pragmatic Programmer*

Layered exception handling


App layer: expose to user or catch and recover
Middle layer: log and re-raise
Base layer: asserts and no handlers
BEGIN
EXCEPTION
END;
call
RAISE
BEGIN
EXCEPTION
END;
call
Err log table
BEGIN
null;
END;
ASSERTFAIL

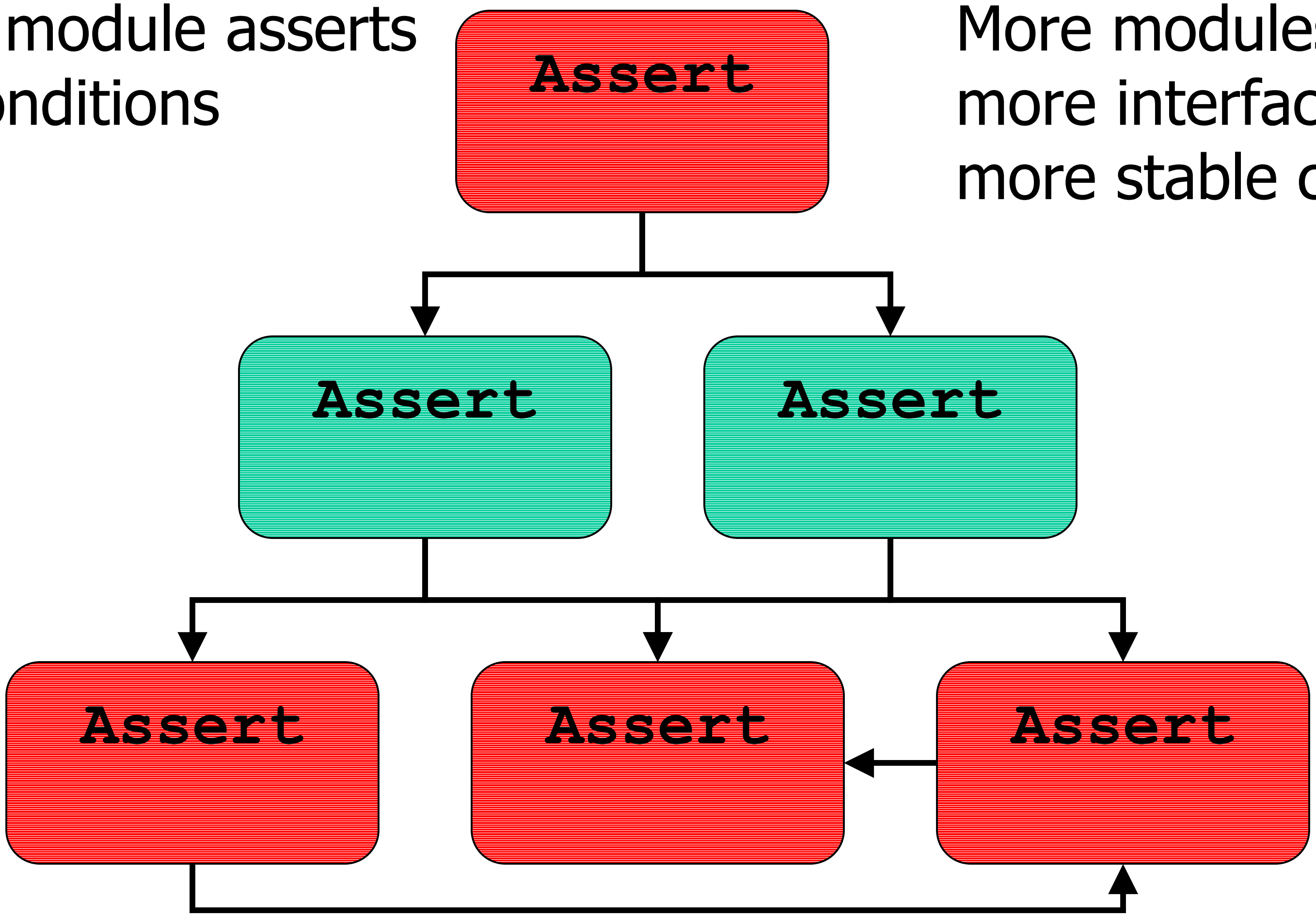
MODULAR CODE

Assembling systems from stable components

Increased contract enforcement

Each module asserts preconditions

More modules = more interfaces = more stable code



"Assertion-tree" of contract-hardened code

Contract-driven modules

- Postcondition guarantee is strong requirement
- Stability derives from rigid contracts
- Systems of precondition assertion trees

Modularization principles

- Single-purpose functions and procedures
- Minimize and organize coupling
- Maintain package coherence
- Simplify interfaces

PL/SQL Construction

- Assert all preconditions
- Modular and function-oriented programming
- Standard local packaged assert (SLPA)

SLPA: specification

```
CREATE PACKAGE pkgname AS

  ASSERTFAIL       EXCEPTION;
  ASSERTFAIL_C     CONSTANT INTEGER := -20999;
  PRAGMA EXCEPTION_INIT(ASSERTFAIL, -20999);
  PKGNAME_C        CONSTANT VARCHAR2(20):=‘pkgname';

  --PROCEDURE assert (bool_IN IN BOOLEAN
  --                  ,msg_IN IN VARCHAR2 := null);
```

- Standardizes ASSERTFAIL exception
- Avoid coupling via duplication

SLPA: implementation

```
PROCEDURE assert (bool_IN IN BOOLEAN
                 ,msg_IN IN VARCHAR2 := null)
IS
BEGIN
   IF NOT NVL(bool_IN,FALSE) -- fail on null input
   THEN
      RAISE_APPLICATION_ERROR
         ( ASSERTFAIL_C, 'ASSERTFAIL:'||
            PKGNAME_C||':'||SUBSTR(msg_IN,1,200)
         ) ;
   END IF;
END assert;
```

- Error message used to signal code location of bug

Transparent error detection

```
FUNCTION tinytest (p1 INTEGER) RETURN NUMBER
IS
BEGIN
   assert(p1 IS NOT NULL,'tinytest:p1 null');
   assert(p1 > 0 AND p1 < 100,
         'tinytest:p1 out of range');
   RETURN (p1/10);
END tinytest;
```

Message identifies function and precondition

```
SQL> execute :t :=utl01.tinytest(null);
BEGIN :t :=utl01.tinytest(null); END;

*
ERROR at line 1:
ORA-20999: ASSERTFAIL:UTL01:tinytest:p1 null
```

Client receives ORA-20999 with detailed debug information

Function-oriented programming

- BOOLEAN Functions

- Basic function structure:
  - *Assert*
  - *Compute*
  - *Return*

- Assert functions in higher-level modules

- Assertion-trees

Example: building a stable module

- Requirement:
  - *Boolean function to tell if date falls on weekend*
- Issue:
  - *Location dependent weekend (US and IL)*
- Solution:

```
FUNCTION isWeekend(loc_IN IN varchar2
                  ,date_IN IN date)
RETURN BOOLEAN;
```

Contract elements

Preconditions

- Date_IN is not null
- Loc_IN is not null
- Loc_IN either 'US' or 'IL'

Hmm...seems pretty simple

Postconditions

- Return TRUE if date_IN is weekend for loc_IN
- Return FALSE otherwise

Module implementation

```
FUNCTION isWeekend(loc_IN IN varchar2
                  ,date_IN IN date)
 RETURN BOOLEAN IS
   tmp_dy integer := TO_CHAR(date_IN,‘D’);
BEGIN
   assert(loc_IN IN (‘US’,’IL’));
   assert(date_IN IS NOT NULL);
   CASE loc_IN
      WHEN ‘US’ THEN RETURN (tmp_dy IN (7,1));
      WHEN ‘IL’ THEN RETURN (tmp_dy IN (6,7));
   END CASE;
END isWeekend;
```

- 9i CASE statement does the work

Problem with TO_CHAR?

- TO_CHAR initializes tmp_dy at declaration
- Do we REALLY know how the 'D' format mask of TO_CHAR works under all NLS settings?

The date format element D returns the number of the day of the week (1-7). The day of the week that is numbered 1 is specified implicitly by the initialization parameter NLS_TERRITORY.

Oracle8i SQL Reference

Asserting problem not present

```
-- September 2,2001 is Sunday
assert(1 = TO_CHAR(TO_DATE('09:02:2001','MM:DD:YYYY')
                  ,'D') );
```

- New precondition: Sunday is day 1
- Harder: make module succeed for all NLS settings
- Flexibility vs. stability tradeoffs

Conclusions (opinions)

- Design by Contract can help reduce PL/SQL defect rates
- The SLPA technique is a good starting point
- More work needs to be done on SQL and contracts