Design by contract for software vigilance and Diagnosability

Yves Le Traon
Professor, University of Luxembourg

Research domains

Component testing
Integration testing
System testing
Model-driven engineering
Security modelling & testing
Fault localization
Component testing
Code

Software testing: cost and trust

« in God we trust, for the rest we test » (A. Petrenko)

Testing

Reliability

Detecting inconsistencies between implementation and specification

Design for testability
(1) Design for trust
Testing issues

Trust the software => trust the tests

- Trust is higher if tests are good
- How « testing » tests ?
- A test is « good » if it has a high fault revealing power
- If tests are not able to detect faults we voluntarily injected… be cautious

A test qualification technique: Mutation analysis
Design for trust

Specification
- executable contracts

Implementation
- Design by Contract
  - Trust based on consistency

V & V: Test Cases, verification

- Reuse trustable components

Two examples of my research

Contracts and reliability
- contracts as embedded oracle functions

System testing
- System testing
  - Contracts for test generation

Overview

1. Contracts as embedded oracles
   - Vigilance
   - Diagnosability
2. Contracts for test generation
3. Conclusion about Design by Contract
Contracts as embedded oracles

The objectives

- Quantitative estimate of what contracts really improve in the software
- We propose two estimates
  - Vigilance
  - Diagnosability
- Obtain general trends
  "Things must be as simple as possible, but no simpler". A. Einstein

Design-by-contract™

- A design philosophy (B. Meyer)
- Component-based OO approach
- A component
  - is not responsible from its inputs consistency (caller responsibility)
    - may refuse to work if caller breaks the contract
  - is responsible from its result
- Specification is derived into executable contracts
Design-by-contract™: analysis

Contracts can detect faults: help to fault localization
Contracts may not detect all faults: contracts quality

BankAccount
(balance ≥ overdraft)

balance: Sum
overdraft: Sum

deposit (amount: Sum)
(pre: amount > 0)
(post: balance = balance@pre + amount)

withdraw (amount: Sum)
(pre: amount > 0)
(post: balance = balance@pre - amount)

Overview

1. Contracts as embedded oracles
   • Vigilance
     • Diagnosability
2. Contracts for test generation
3. Conclusion about Design by Contract

Vigilance

Informally “the quality or state of being wakeful and alert”

component

contracts

Isolated vigilance contracts quality

Intuition: Combination is better than addition

Global vigilance
Vigilance

- **Vigilance (V)**: The vigilance expresses the probability that the system contracts dynamically detect faulty states that would have otherwise provoked a failure.

- Weakness is the contrary = 1 - V

Axiomatization: intuitive properties

Examples:

**GVA2 - System concatenation.** The global vigilance of a system obtained by concatenation of two systems S1 and S2 cannot be lower than the lowest vigilance of S1 and S2.

**GVP3 - Contract addition.** For any system, its global vigilance cannot decrease by addition of a contract.

⇒ Useful for formal validation

Vigilance: Test dependency

A component plugged into a system has a vigilance enhanced by its clients contracts ⇒ **test dependency** (TD)

Definition. Det(Ci, Cj): probability that Ci’s contracts detect an error in Cj
Global vigilance

- Let $Prob_{error}(i, S)$ be the probability the failure in $S$ comes from the component $C_i$

$$V(S) = 1 - Weak(S) = 1 - \sum \text{Prob}_{error}(i, S) \cdot LocWeak(C_i, S)$$

$$LocWeak(C_i, S) = Weak,$$

prob. component $i$ does not detect the faulty state

Formal and Empirical validation


- GVP3 - Contract addition. For any system, its global vigilance cannot decrease by addition of a contract.
- Axiomatization
- Mathematical modelling
- Theoretical validation of the model
- GVP3 - Contract addition. For any system, its global vigilance cannot decrease by addition of a contract.
- Results

<table>
<thead>
<tr>
<th>Min.</th>
<th>Max.</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>83%</td>
<td>69%</td>
</tr>
</tbody>
</table>

% mutants of provider killed by client's contracts

Det(Ci,Cj)

Vigilance: Empirical results and interpretation

A Telecommunication Switching System (SMDS): 17 classes, 72 relationships

The Pylon library: 50 classes and 134 relationships

The InterViews library: 146 classes and 420 relationships

GVP3 - Contract addition. For any system, its global vigilance cannot decrease by addition of a contract.
### Vigilance: Conclusion

- About the results:
  - no contracts $\iff$ system not vigilant
  - vigilance improves rapidly with contracts quality
  - very high vigilance is very expensive: almost 40% more contracts to improve from 80% to 100% vigilance

### Overview

1. Contracts as embedded oracles
   - Vigilance
   - Diagnosability
2. Contracts for test generation
3. Conclusion about Design by Contract

### Diagnosability

- Diagnosability expresses the effort for the localization of a fault.
Diagnosability: the help of contracts

Classical software

Diagnosis scope

Diagnosability: the help of contracts

Designed by contract software

Diagnosis scope

Diagnosability

\[ \delta = \frac{\text{max}}{2} (P_{\text{faulty} \times \delta}) \]

DiagnosisScope(i,j) = (j-i+1) * |IS|

\[ \text{Det}_j = P_j \frac{1}{\text{#contracts}} (1 - p_j) \]

\( IS_1 \)
\( IS_2 \)
\( IS_3 \)
\( IS_4 \)
\( IS_5 \)

\( IS_{\text{contracts}} \)
**Conclusion Vigilance & Diagnosability**

- Measures estimate the contribution of contract quality and density
- The quality of contracts is more important than their quantity
- Related work
  - Automated fault localization (ICSE 2006)


**Design for trust**

- Specification: executable contracts
- Implementation
- Trust based on consistency
- V & V: Test Cases, verification
- Reuse trustable components
Overview

1. Contracts as embedded oracles
   - Vigilance
   - Diagnosability
2. Contracts for test generation
3. Conclusion about Design by Contract

Contracts for test generation
Problem analysis: Model-Based System Testing (for product lines)

- System requirements ...
  - ... evolve very often
    - Nokia: 69% of the requirements modified, 22% modified twice
      → need to build quickly new tests from the new requirements
  - ... are in natural language
    → need of a formalization to apply automatic test generation techniques

The problem: a gap to bridge

Requirements

1.1 "Register a book"
the "book" becomes "registered" after the "librarian" did "register" the "book".
the "book" is "available" after the "librarian" did "register" the "book".

Test objectives

- [connect(p1), plan(p1,m1)]
- [connect(p1), plan(p1,m1), open(p1,m1), close(p1,m1)]

Test selection criteria

- Simulation
- Requirement model
- Test objectives

Test cases
A use case contract language

- First order logic expressions
  - Boolean properties (predicates) = name+typed parameters
    - Ex: planned(m:meeting)
      manager(u:participant,m:meeting)
  - Enumerated properties
  - Classical boolean operators (and, or, implies, not)
  - Quantifiers (forall, exists)

- Benefits:
  - formalization of the use cases
  - dependencies between the use cases can be deduced

A use case contract language: Deducing dependencies

#use case OPEN
UC open(u:participant; m:mtg)
pre created(m) and [moderator(u,m)] and not closed(m) and not opened(m) and connected(u)
post opened(m)

#use case CLOSE
UC close(u:participant; m:mtg)
pre opened(m) and [moderator(u,m)]
post "...

OPEN(u1,m1);CLOSE(u1,m1) is a correct sequence

The Use Case Transition System (UCTS)

OPEN(p1,m1);ENTER(p1,m1);CLOSE(p1,m1) is a correct sequence
Test selection criteria

<table>
<thead>
<tr>
<th>Nominal behaviors</th>
<th>Robustness behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate sequences leading to all licit application of the use case</td>
<td>Generate sequences leading to an invalid application of the use case</td>
</tr>
<tr>
<td>Correct sequence</td>
<td>Correct sequence</td>
</tr>
<tr>
<td>correct action</td>
<td>Non specified action</td>
</tr>
</tbody>
</table>

“all configurations making its precondition true”

“all configurations making its precondition false”

Hypothesis

- **H1**: Test cases produced from requirements are « efficient » to test the overall system.
  - Adequacy criterion from the industry: code coverage
  - Comparing test criteria

- **H2**: Most real-industrial requirements can be treated with such an approach
  - % operational requirements which can be covered by the approach

H1: Academic experiments

- 3 case studies
  - FTP client
  - ATM
  - Virtual meeting
- Code repartition
- Code Coverage

**Code coverage**

<table>
<thead>
<tr>
<th>Code covered with APT + robustness criterion</th>
</tr>
</thead>
</table>

- Nominal code
  - 65%
- Robustness code w.r.t. spec
  - 8%
- Robustness code w.r.t. env
  - 18%
- Dead code
  - 9%
H2: Experiments with TAS and France Telecom

- TAS: Two components of weapon navigation system (Mirage 2000-9 and Rafale).
- France telecom: Three services on the livebox 2 modem
- Translation of the requirements from English to RDL

0% 20% 70%
cannot be translated (arithmetic, real-time) could be translated (limit of the prototype tool) translated

New issues

- Initiate many researches:
  - MDE for Requirements
  - Product lines testing and verification
- Commercial tools are now available based on similar principles
- Empirical validation of Model-based testing
  - ALCATEL: testing new distributed telecom services
  - French Defense Department: testing cryptographic components

Overview

1. Contracts as embedded oracles
   - Vigilance
   - Diagnosability
2. Contracts for test generation
3. Conclusion about Design by Contract
Conclusion

• Design by Contract
  • an instrument to build trust in a system
  • Declarative approach
  • Lightweight
• Can be used for
  • Fault localization
  • Test generation
  • Security
  • Vigilance \(\rightarrow\) adaptive resilient systems

Creation of the main conference on testing, verification and validation

• Gang of Four : L. Briand, J. Oflutt, B. Baudry, Y. Le Traon
• 1st edition : 303 abstracts, 224 full papers
• 250 participants
• 8 associated workshops
• Selection rate: 20-25%

Questions?

« intelligently react to abnormal situations and ensure the quality of the information » (P1 conclusion)
Threats to validity: sensitivity analysis

never contradict the axioms
Threats to validity: contracts repartition

Table 1: contracts distribution

<table>
<thead>
<tr>
<th>Contract</th>
<th>Frequency</th>
<th>Error</th>
<th>Error / Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Testing</td>
<td>55</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Virtual Meeting</td>
<td>15</td>
<td>0.75</td>
<td>0.5</td>
</tr>
<tr>
<td>Junit Auto-Test</td>
<td>50</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>Loading JDK</td>
<td>20</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>New</td>
<td>35</td>
<td>0.75</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Proof for GV3

$\text{GVP3 - Contract addition. For any system, its global vigilance cannot decrease by addition of a contract.}$

- Consider that we add a contract to one component, for instance, $C_1$
  \[
  \text{Det}(C_1', C_1') = \text{Det}(C_1, C_1) \geq (1) 
  \]
  \[
  \forall k \in [2, q]: \text{LocWeak}(C_k, S') \leq \text{LocWeak}(C_k, S) \geq (2) 
  \]
  Consider that $C_1$ has $q$ servers, we denote $[C_2, C_3, ..., C_q]$
  From (2): $\forall k \in [2, q]: \text{LocWeak}(C_k, S') \leq \text{LocWeak}(C_k, S)$

$$P(S') = 1 - \sum_{C_i} \text{Prob. _Err}(C_i, S') \times \text{LocWeak}(C_i, S') + \sum_{C_i, C_j} \text{Prob. _Err}(C_i, C_j, S') \times \text{LocWeak}(C_i, C_j, S')$$

$$P(S) = 1 - \sum_{C_i} \text{Prob. _Err}(C_i, S) \times \text{LocWeak}(C_i, S) + \sum_{C_i, C_j} \text{Prob. _Err}(C_i, C_j, S) \times \text{LocWeak}(C_i, C_j, S)$$

$$\Rightarrow P(S) \geq P(S')$$
Diagnosability: Measures

- Assumptions:
  - the contracts repartition in a flow is uniform
    - Each IS has same size ISsize (=#stat div #contracts),
  - the closer a contract is to the faulty statement i the more probable it can detect the fault
  - the contracts have an equal probability p to detect a fault coming from the statements they are directly consecutive to
  - each statement has the same probability to be faulty equal to 1/Nstat

Diagnosability

\[
\delta = \frac{1}{2} \sum_{i=1}^{#contracts} \text{DiagnosisScope}(i,j) \times \text{Det}_i \times \text{PFailure} \times \text{DiagnosisScope}(i,n_{contracts})
\]

\[
\text{DiagnosisScope}(i,j) = (j-i+1) \times |IS_i|
\]

\[
\text{Det}_i = \text{the probability that contract } j \text{ detects a faulty statement in IS}_i
\]

Absorption coefficient \( \alpha \): \( p_j = \alpha \cdot p \)

\[
\delta = ISize \cdot \sum_{i=1}^{#contracts} \text{DiagnosisScope}(i,j) \times \text{Det}_i \times \frac{N_{stat} - 1}{N_{stat} - 1} \times \text{Det}(#contracts - i + 1)
\]

Diagnosability: simplified

\[
\delta = \frac{1}{2} \sum_{i=1}^{#contracts} \text{DiagnosisScope}(i,j) \times \text{Det}_i \times \text{PFailure} \times \text{DiagnosisScope}(i,n_{contracts})
\]

\[
\text{DiagnosisScope}(i,j) = (j-i+1) \times |IS_i|
\]

\[
\text{Det}_i = \text{the probability that contract } j \text{ detects a faulty statement in IS}_i
\]

\[
\text{Det}_i = p_j^{|IS_i|} \frac{1}{|IS_i|} (1-p_j)
\]
Cursus and diploma

- 1994: Master in computer science (DEA) from INPG, Grenoble.
- 1994: Engineering Degree - ENSIMAG.
- 1998-2004: Assistant professor, Univ. of Rennes 1 - IRISA lab.
- 2004: Authorization for the management of research (HDR).
- 2004-2006: France Télécom R&D
- Sept. 2008: Professor, Head of the SERVAL team.
- July 2009: Professor, Univ. Luxembourg.

Some results and ongoing work

- Unit component testing
  - Self-testable components (IEEE Software 01)
  - Evolutionary algorithms (IEEE Software 05, STVR 05)
- Integration testing
  - Refactoring of UML models (best paper UML 01)
- System Testing
  - Evolutionary algorithms (IEEE Software 07)
- Testability Analysis
  - Model-driven engineering (Information Software and Technology 05, IEEE TSE 08)
- Security
  - Modeling (ICST 08, Best paper Models 08)
  - Testing (ISSE 07, ISSE 08, ICST 09)
- Communication and networks
  - P2P system testing (Best paper ISSE 08)
- MDE and Barriers to Systematic Model Transformation Testing (SoSym Journ 07, Communications of the ACM 2010)

Research domains

- Component testing
- Integration testing
- System testing
- Model-driven engineering
- Security modeling & testing
- Fault localization
- (meta)models
- Code
Some results and ongoing work

- Unit component testing
  - Self-testable components (IEEE Software 01)
  - Evolutionary algorithms (IEEE Software 05, SIVR 05)
- Integration testing (best paper ISSRE 09, IEEE Trans. on Reliability, ICDOC 01)
- System Testing (IEEE TSE 06, IEEE TSE 07)
- Testability Analysis
  - Refactoring of UML models (Best paper UML 01)
  - Measurements (Information Software and Technology 06, IEEE TSE 06)
- Security
  - Modeling (ICST 06, Best paper Models 08)
  - Testing (ISSRE 07, ISSRE 08, ICST 08, ICST 09)
- Communication and networks
  - P2P system testing (Best paper ISSRE 08)
- MDE and Barriers to Systematic Model Transformation Testing (SciSym jour 07, Communications of the ACM 2010)
  - Aspect Oriented Programming and testing
  - Requirements and Model-driven engineering
  - ad-hoc network testing
  - security contracts

Industrial partnerships and valorization

- Contracts
  - European fundings
    - 2000-2004: Café, Families :Product lines, OO modeling
      - Nokia, Ericsson, Philips...
    - 2005-2006: Modelware :Model-driven Engineering
  - French fundings
    - Cote, Politess, Dali
  - Direct contracts
    - 1995-1998: PEA Aérospatiale/Airbus
    - 2002-2004 :Caroll (INRIA, CEA, THALES)
  - Main french partners
    - THALES TRT and TAS,
    - France Telecom R&D,
    - EADS Test & Services,
    - French Defense Department (DGA)

- Tools developments and valorization
  -UCTSystem
  - Kermeta (metamodeling language for executability)
  - Many prototype tools (AOP testing, Security Testing)

- Two years at France Télécom R&D
  - Real world projects
    - IS Migration, new telecom services modelling and testing (MDE)

- Courses for companies
  - The « Test essentials » program for ALCATEL
  - Thomson, Mitsubishi, EDS...
International visibility

- 9 past PhDs, 3 running PhDs
- >90 int. papers (15 journals)
  - Communications of the ACM (CACM)
  - (3) IEEE Trans. on Software Engineering,
  - IEEE Trans. on Reliability,
  - (2) Software, Testing, Verification & Reliability journal (STVR)
  - (2) IEEE Software,
  - IEEE Design & Test,
  - SoSym,
  - Information & Software Technology.
- PC member
  - IEEE ICST, IEEE ISSRE, IEEE Metrics, ICSOFT, ICFI...
- Steering committees
  - Testing: IEEE ICST, Mutation, IWoTA, SecTest
  - Reliability : IEEE ISSRE

Cursus and diploma

1994. Master in computer science (DEA) from INPG, Grenoble.
1994. Engineering Degree - ENSIMAG.
1997-1998. PostDoc at LCIS-INPG lab (Valence)
2004-2006. France Télécom R&D
Nov. 2006. Associate professor - Télécom Bretagne
Sept. 2008. Professor
July. 2009. Professor
Univ. Luxembourg

Industrial partnerships and valorization

- Contracts
  - European fundings
    - 2000-2004: Café, Families, Product lines, OO modeling
    - NOKIA, Ericsson, Philips
    - 2005-2006: Modelware, Model-driven Engineering
  - French fundings
    - Cete, Politeia, Dali
  - Direct contracts
    - 1995-1998: PEA Aéropsych/Arbus
    - 2002-2004: Carol (INRIA, CEA, THALES)
    - 2008+: French Defense Department (Security Testing)
- Main french partners
  - THALES TRT and TAS,
  - France Telecom R&D,
  - EADS Test & Services,
  - French Defense Department (DGA)
Industrial partnerships and valorization

- Tools developments and valorization
  - UCTSystem
  - Kermeta (metamodeling language for executability)
  - Many prototype tools (AOP testing, Security Testing)

- Two years at France Télécom R&D
  - Real world projects
    - IS Migration, new telecom services modelling and testing (MDE)

- Courses for companies
  - The « Test essentials » program for ALCATEL
  - Thomson, Mitsubishi, EDS...

International visibility

- 9 past PhDs, 3 running PhDs
- 100 int. papers (18 journals)
  - Communications of the ACM (CACM)
  - IEEE Trans. on Software Engineering,
  - IEEE Trans. on Reliability,
  - Empirical Software Engineering
  - Software, Testing, Verification & Reliability journal (STVR)
  - IEEE Software,
  - IEEE Design & Test,
  - SoSym,
  - Information & Software Technology.

- PC member
  - IEEE ICST, IEEE ISSRE, IEEE Metrics, ICSOFT, ICFI...

- Steering committees
  - Testing: IEEE ICST, Mutation, IWoTA, SecTest
  - Reliability : IEEE ISSRE

Hard point 2: test objective generation

- Test objective
  - path of the UCTS
  - sequence of instantiated use cases

- Generating test objectives
  - Extracting short paths in the UCTS
  - Extracting a « reasonable » number of paths
  - Test criteria
    - 4 structural criteria
    - 1 semantic criterion
    - 1 robustness criterion

Test objectives

UCTS

Test generation

Test criteria

UCTS

Test objectives

(UC1p1,p2, UC1p2, UC4p1)
(UC1p3, UC1p4, UC2p1)