Test Automation: An Empirical Perspective.

*Part I -- Introduction*

Long Tutorial at the *GTTSE Summer School on Generative and Transformational Techniques in Software Engineering*, Braga, Portugal, 2011

Arie van Deursen, Delft University of Technology
My Journey

ASF+SDF, DSLs, language prototyping, reverse engineering, reverse engineering, sw architecture, testing, human factors, ...
This Tutorial

Topics
• Software testing
• Test automation
• Web applications
• Plug-in architectures

Angles
• In search for connections
• Pushing the limit
• Empirically grounded
• What’s our impact?
A DSL for Banking

- Specify essence of interest rate products
- Generate Cobol code
- Early 90s – now
- Survived many mergers

---

**Little Languages: Little Maintenance?**

**ARIE VAN DEURSEN**

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands; arie@cwi.nl

**PAUL KLINT**

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands; paul@cwi.nl and University of Amsterdam, Kristalsl 403, 1098 SJ Amsterdam, The Netherlands

**ABSTRACT**

So-called little, or domain-specific languages (DSLs), have the potential to make software maintenance simpler: domain-experts can directly use the DSL to make required routine modifications. On the negative side, however, more substantial changes may become more difficult: such changes may involve altering the domain-specific language. This will require compiler technology knowledge, which not every commercial enterprise has easily available. Based on experience taken from industrial practice, we discuss the role of DSLs in software maintenance, the dangers introduced by using them, and techniques for controlling the risks involved.


Keywords and Phrases: Domain-specific language, software maintenance, interest rate products, language prototyping, software generation, component coordination.


**1 Introduction**

Little languages, tailored towards the specific needs of a particular domain, can significantly ease building software systems for that domain (Bentley, 1986). To cite Herndon and Berranz (1988),

> If a conceptual framework is rich enough and program tasks within the framework are common enough, a language supporting the primitive concepts of the framework is called for. (…) Many tasks can be easily described by agreeing upon an appropriate vocabulary and conceptual framework. These frameworks may allow a description of a few lines long to replace many thousand lines of code in other languages.

We will use the following terminology (see also Figure 1):

**Domain-Specific Language (DSL)** A small, usually declarative, language expressive over the distinguishing characteristics of a set of programs in a particular problem domain (Walton, 1996).
@avandeursen
Arie van Deursen

Just got called for expertise on Risla DSL -
in 1992. One year consultancy job
available.

20 Apr via web  Favorite  Reply  Delete
Risla’s Generated Cobol

- Risla Product Definition
- Risla Compiler
- Legacy Data Structures (VSAM)
- Input Screens (CICS)
- Registration Methods (Cobol)
- Management Info. Routines (Cobol)
Exploring Legacy Systems Using Types

Arie van Deursen  
Leon Moonen

ABSTRACT

We show how hypertext-based program understanding tools can achieve new levels of abstraction by using inferred type information for cases where the subject software system is written in a weakly typed language. We propose TYPEEXPLORER, a tool for browsing COBOL legacy systems based on these types. The paper addresses (1) how types, an invented abstraction, can be presented meaningfully to software re-engineers; (2) the implementation techniques used to construct TYPEEXPLORER; and (3) the use of TYPEEXPLORER for understanding legacy systems, at the level of individual statements as well as at the level of the software architecture — which is illustrated by using TYPEEXPLORER to browse an industrial COBOL system of 100,000 lines of code.


Keywords and Phrases: Software maintenance, program understanding, program analysis, type inference, documentation generation, variable usage, hypertext.


Note: Work carried out under projects SEN 1.1, Software Renovation and SEN 1.5, Domain-Specific Languages.

1. Introduction

Software immigrants, employees that are added to an existing software system in order to produce new versions, face many obstacles: their own code as time between development and maintenance goes by. As a consequence, maintenance tasks become difficult, expensive, and error prone.

To reduce these costs, software engineers are increasingly interested in the use of program understanding and reverse engineering tools. However, these tools often suffer from limited understanding of subject systems — derived from their original design and implementation. Within the COBOL domain, such an understanding can be integrated into an understanding of the individual software system. The software engineer can follow a “trace” through the “opportunity” hypothesized in the system documentation. This is typically those documents that accompany the system and are intended to be used at integration of the software system, in order to ensure reliability.

The fundamental problem of understanding (and integration) is to arrive at a consistent model of the system and the interaction between its components. Our recognition by looking at the system in isolation.

For type inference, the types for program structures are essential: types for program structures, function signatures, parameter types, data variables, etc. Many of the existing software systems, however, are written in older languages with very weak type systems.

This is why we focus on weakly typed languages, in which we need to develop new understanding tools. One such tool is TYPEEXPLORER, which allows software maintainers to explore the type structure of COBOL programs.
Our bachelor project got rated 4 out of 5 stars by the S.I.G. for our code quality. Also got some great feedback to work on. Thanks @sig_eu!
Crawljax: Automated Testing of Ajax Applications
Bas Cornelissen, Andy Zaidman, Arie van Deursen,
A Controlled Experiment for Program Comprehension through Trace Visualization.
*IEEE Transactions on Software Engineering*, 2011
Spreadsheet Analysis

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1) Cell classification

2) Identifying data blocks

3) Name resolution

4) Dateflow construction

5) Name replacement

6) Grouping

Data block-End Result
Sheet-Overall
Can we predict impact?

1. Some *very cool* results are never applied 😞

2. Need to speak the user’s language

3. Simpler might be more effective

4. What is our perspective on evaluation?
What is “Exciting” in an Engineering Field?

1. Invention of wholly new ideas and directions

2. Work of promise that illuminates #1

3. Early application of #2 showing clear prospect of benefit

4. Substantial exploitation of #3 yielding measurable societal benefits

5. Maturing of #4 with widespread adoption by practitioners
[ Some Discussion Outcomes ]

• Run time versus design time
• Repository mining
• Reinventing analysis
• Scaling up debugging
• SE techniques in new domains (health)
• Societal scale systems (REST)
What Can We Learn From The Social Sciences?

Paradigms shaping the practice of research:

• Post-positivism
• Social constructivism
• Participatory / advocacy
• Pragmatism
Post-positivism

- *Conjectures and Refutations: The Growth of Scientific Knowledge*
- Testing of hypotheses
- A priori use of theory
Pragmatism

• Clarify meanings of intellectual concepts by tracing out their “conceivable practical consequences”.
  (Charles Peirce, 1905)

• Do not insist upon antecedent phenomena, but upon consequent phenomena;
  Not upon the precedents but upon the possibilities of action
  (John Dewey, 1931)
Pragmatic Considerations

• Not every belief that is “true” is to be acted upon

• Not committed to single research method

• Research occurs in social (and technological) context

• Research builds up “group knowledge”
The Qualitative Research Palette

- Measuring applicability?
- The outcome as a narrative
- Multi-facetted validity

- Case studies
- Ethnography
- Participant observation
- Grounded theory
- Phenomenology
- Narrative studies
- Participative inquiry
- Interviewing
- Document analysis
- ...

Carolyn B. Seaman.
Qualitative methods in empirical studies of software engineering. *IEEE TSE*, 1999
"Doing research" simply means the systematic use of some set of theoretical and empirical tools to try to increase our understanding of some set of phenomena or events. In the social and behavioral sciences, the phenomena of interest involve states and actions of human systems — of individuals, groups, organizations, and larger social entities — and the by-products of those actions.

The meaning of research evidence, in any area of science, is inherently tied to the means or methods by which that evidence was obtained. Hence, to understand empirical evidence, its meaning, and its limitations, requires that you understand the concepts and techniques on which that evidence is based.

This chapter is about some of the tools with which researchers in the social and behavioral sciences go about "doing" research. It raises some issues about strategy, tactics and operations. Especially, it points out some of the inherent limits, as well as the potential strengths, of various features of the research process by which behavioral and social scientists do research.

SOME BASIC FEATURES OF THE RESEARCH PROCESS

Doing research, in the behavioral and social sciences, always involves bringing together three sets of things:

(a) some content that is of interest,
(b) some ideas that give meaning to that content, and
(c) some techniques or procedures by means of which those ideas and contents can be studied.

For example, the contents of a study might involve the behavior of a jury, conversations in a family about buying a new car, the voting behavior of members of a community, littering in a park, courtship patterns in a small town, and so forth. The ideas might include the concept of attitudes, the notion that education affects political preferences, the concept of conformity, the hypothesis that groups whose members like one another perform tasks better than groups whose members do not like each other, and so forth. The techniques might include a questionnaire to assess individual attitudes, toward a car or a candidate or group mates; a set of procedures for observing family discussions about cars and money; a means to gather election returns; a plan to evaluate the quality of group task products; and so forth.

I will refer to these three sets of things more formally, as three distinct, though interrelated, domains:

(a) The Substantive domain, from which we draw contents that seem worthy of our study and attention;
(b) The Conceptual domain, from which we draw ideas that seem likely to give meaning to our results; and
(c) The Methodological domain, from which we draw techniques that seem useful in conducting that research.
McGrath on Maximizing Criteria

1. **Generalizability** of evidence over population
2. **Precision** of measurement and control over extraneous factors
3. **Realism** of the context in which evidence is gathered.

“Although you always want to maximize all three criteria simultaneously, you cannot do so”
McGrath on Credible Knowledge

1. Results depend on methods.
2. All methods have limitations.
3. Hence, any set of results is limited.

“Credible empirical knowledge requires consistency or convergence of evidence across studies based on different methods.”
### Design Science

<table>
<thead>
<tr>
<th>Type</th>
<th>Conditions</th>
<th>Control of Practice</th>
<th>Example</th>
<th>Users</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illustration</td>
<td>no</td>
<td>yes</td>
<td>small</td>
<td>designer</td>
<td>illustration</td>
</tr>
<tr>
<td>Opinion</td>
<td>imagined</td>
<td>yes</td>
<td>any</td>
<td>stakeholder</td>
<td>support</td>
</tr>
<tr>
<td>Lab demo</td>
<td>no</td>
<td>yes</td>
<td>realistic</td>
<td>designer</td>
<td>knowledge</td>
</tr>
<tr>
<td>Lab experiment</td>
<td>no</td>
<td>yes</td>
<td>artificial</td>
<td>subjects</td>
<td>knowledge</td>
</tr>
<tr>
<td>Benchmark</td>
<td>no</td>
<td>yes</td>
<td>standard</td>
<td>designer</td>
<td>knowledge</td>
</tr>
<tr>
<td>Field trial</td>
<td>yes</td>
<td>yes</td>
<td>realistic</td>
<td>designer</td>
<td>knowledge</td>
</tr>
<tr>
<td>Field experiment</td>
<td>yes</td>
<td>yes</td>
<td>realistic</td>
<td>stakeholder</td>
<td>knowledge</td>
</tr>
<tr>
<td>Action case</td>
<td>yes</td>
<td>no</td>
<td>real</td>
<td>designer</td>
<td>knowledge and change</td>
</tr>
<tr>
<td>Pilot project</td>
<td>yes</td>
<td>no</td>
<td>realistic</td>
<td>stakeholder</td>
<td>knowledge and change</td>
</tr>
<tr>
<td>Case study</td>
<td>yes</td>
<td>no</td>
<td>real</td>
<td>stakeholder</td>
<td>knowledge and change</td>
</tr>
</tbody>
</table>

Roel Wieringa. Design Science Methodology. Presentation for Deutsche Telekom, 2010 (also ICSE, RE tutorial)
This Tutorial

Topics
• Software testing
• Test automation
• Web applications
• Plug-in architectures

Angles
• In search for connections
• To the limit
• Empirically grounded
• What’s our impact?
Why I Love Software Testing (1)

Dependability in practice
Why I Love Software Testing (2)

Research Challenges
Why I Love Software Testing (3)

System Understanding
Test Automation

“software that automates any aspect of testing of an application system”

• Generate inputs & expected results
• Run test suites without manual intervention
• Turn scenarios into executable test cases

JUnit, Rspec/Cucumber, Selenium, TTCN, Hexawise, Hadoop, ...
Testing Chrome in Quotes

• Chromium development places a high premium on tests, tests, tests, tests and more tests
  — The Chromium Buildbots run these tests 24x7

• It is imperative that test suites be updated, maintained, executed, and evolved

• Developers contributing code are expected to always run all tests

• Keep the source tree always building so that regressions are minimized and developer productivity is maximized

• When contributing code, always
  — consider whether your change has enough testing:
  — a new feature or module should almost certainly be accompanied by tests.

See http://dev.chromium.org/developers/testing
GUI Testing

Google WindowTester Pro

Jubula Automated functional testing

New UI Test

Source folder: pacmanFunctionalTest/src
Package: jpacman
Name:
Modifiers: public
Superclass: com.windowtester.runtime.swing.UITestCaseSwing
Create folders automatically if they do not exist
Add build path dependencies automatically

Finish Cancel
Feature: Log in and out
   As an administrator
   I want to restrict access to certain portions of my site
   In order to prevent users from changing the content

Scenario: Logging in
   Given I am not logged in as an administrator
   When I go to the administrative page
   And I fill in the fields
      | Username | admin |
      | Password | secret |
   And I press "Log in"
   Then I should be on the administrative page
   And I should see "Log out"

Scenario: Logging out ...
Exploratory Testing

• Human tester,
  – using brain, fingers, & wit
  – to create scenarios that
  – will cause software either to fail or to fulfill its mission.

• Take advantage of human cleverness
  – *No scripts:* *Exploratory Testing*
  – Record test findings as you go
## “Touring” Tests

<table>
<thead>
<tr>
<th>Touring</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Guidebook tour</td>
<td>• Use the manual</td>
</tr>
<tr>
<td>• Money tour</td>
<td>• The money-generating features</td>
</tr>
<tr>
<td>• Landmark tour</td>
<td>• Key features</td>
</tr>
<tr>
<td>• After hours tour</td>
<td>• Batch functionality</td>
</tr>
<tr>
<td>• Museum tour</td>
<td>• Legacy features</td>
</tr>
<tr>
<td>• Rained-out tour</td>
<td>• Start and then cancel operations</td>
</tr>
<tr>
<td>• Couch potato tour</td>
<td>• Do as little as possible (all defaults)</td>
</tr>
<tr>
<td>• Antisocial tour</td>
<td>• Known bad inputs</td>
</tr>
</tbody>
</table>

http://blogs.msdn.com/b/james_whittaker/
Testing is about Varying Things:
Input, State, Paths, Data, Environment

Input:
- Atomic versus abstract
- Input combinations
- Order of inputs
- Legal versus illegal
- Input filters & checks
- Normal versus special
- Default / user supplied

State
- History of stimuli

Paths
- Routes through system

Data
- Evolution over months

Environment
- Simulate the real world
“Test automation is often built to solve too big a problem. This broad scope makes automation brittle and flaky because it's trying to do too much.

There are certain things that automation is good at and certain things humans are good at.

What I want is automation that makes my job as a human easier.

Automation is good at analyzing data and noticing patterns. It is not good at determining relevance and making judgment calls. Fortunately humans excel at judgment.

So when I am doing exploratory testing, I want automation following me and helping.”
Testing is your Business

• Testing: 50% of costs of software projects

• Your work must simplify testing!
  – For developers
  – For system integrators
  – For stakeholders

• Does it? How?
1. Which levels of test automation are possible?

2. Which levels of test automation are desirable?
Key Points

• Engineering calls for pragmatism
• Assess ‘opportunities for action’
• Qualitative methods to the rescue
• Multiple evaluations required

• You’re working on testing too
GTTSE Outlook

• Wednesday: Reflections on *test automation for the web*

• Thursday: Reflections on *compositional testing*