An Industrial Case Study in Compiler Testing

11th International Conference on Software Language Engineering

Dr. Vadim Zaytsev aka @grammarware

raincode

LABS

compiler experts
Introduction

- I am @grammarware
- MSc (ru, nl), PhD (nl/de)
- Ex-researcher (UKL, CWI)
- Ex-lecturer (UvA)
- Now @ Raincode [Labs]
- Chief Science Officer
- Writes compilers for a living
- Last project: TIALAA (4GL)
  - There Is A Life After AppBuilder
  - https://www.raincode.com/technical-landscape/tialaa/
4GLs are [Badly Designed] DSLs
AppBuilder concepts are dialectic

- A “rule” is program
  - not declarative
- A “set” is a lookup table
  - not a set at all
- A “view” is a model
  - in MVC terms
- “SetEncoding” is a getter
  - while “SetVisible” is a setter
AppBuilder rule syntax is ambiguous

```
CASEOF X
  CASE A B
    FOO
  CASE C D
    BAR
ENDCASE
```
AppBuilder typing is not name-unique

MAP A IN A TO A

MAP A OF B TO X // might mean A OF C OF B

MAP A OF B OF C (N) TO X // the index may refer to A, B or C
AppBuilder semantics is special

❖ TRUNC(X, -3)
❖ DATE('00-00-0000')
❖ DATE('00-00-0000') + 1
❖ DATE('00-00-0000') - 1
❖ USE RULE X // closes the window if client-client call
❖ MAP A TO B
Need for testing methodology

DESPERATE need for compiler testing methodology
D-tests: **direct** access to runtime

```csharp
[TestMethod]
[TestCategory("runtime"), TestCategory("time")]
[TestCategory("CSL:D")]
public void D_TimeAutoMilliDots()
{
    CslTime time;
    for (int h = 0; h < 24; h++)
        for (int m = 0; m < 60; m++)
            for (int s = 0; s < 60; s++)
                for (int f = 0; f < 1000; f++)
                    {
                        time = new CslTime("{h:D2}.#{m:D2}.#{s:D2}.#{f:D3}"");
                        Assert.AreEqual(h, time.Hours);
                        Assert.AreEqual(m, time.Minutes);
                        Assert.AreEqual(s, time.Seconds);
                        Assert.AreEqual(f, time.Milsecs);
                    }
}
```
R-tests: yes/no recognition

```csharp
[TestMethod]
[TestCategory("recognise"), TestCategory("rule"), TestCategory("clear")]
[TestCategory("CSL:R")]

public void R_ClearChar()
{
    Rule root = Parser.Parse(
        "dcl\n" +
        "    L_RESULT char(50);\n" +
        "enddcl\n" +
        "map 'hello' to L_RESULT\n" +
        "clear L_RESULT\n" +
        "print L_RESULT\n" +
        ""
    );
    Assert.AreEqual(root.Lines.Count, 0);
    Assert.IsTrue(FatalSmellFinder.NoProblems(root));
}
```
P-tests: parsing

THAT ESCALATED QUICKLY
N-tests: normalisation
T-tests: typing

```csharp
[TestMethod]
[TestCategory("typecheck"), TestCategory("rule"), TestCategory("clear")]
[TestCategory("CSL:T")]

0 references | Vadim Zaytsev, 41 days ago | 1 author, 5 changes

public void T_ClearDec15()
{
    SymbolTable table = new SymbolTable();
    AbstractTypes.one().PurgeLocalViews();
    table.Populate(FrontEnd.LoadRuleFromText(
        "dcl\n" +
        "    L_RESULT dec(15);\n" +
        "  enddcl\n" +
        "map 42 to L_RESULT\n" +
        "clear L_RESULT\n" +
        "print L_RESULT\n" +
        ""\n    ), null);
    Assert.AreEqual(1, table.Count);
    Assert.IsTrue(table.IsDeclaredStrict("L_RESULT"));
    Assert.IsTrue(table.GetTypeOf("L_RESULT") is CastleTypeDecimal);
    Assert.AreEqual(15, (table.GetTypeOf("LRESULT") as CastleTypeDecimal).Length);
    Assert.AreEqual(0, (table.GetTypeOf("L_RESULT") as CastleTypeDecimal).Scale);
}
```
S-tests: successful execution

```csharp
[TestMethod]
[TestCategory("CSL:S")]
public void S_CastDateTimeToCharY() => RunAndCompare(new S_CastDateTimeToCharY(), "$9$$9$$");
```
Tests were used:

❖ Mostly during middle stages of the project
  ➢ too little info early on
  ➢ easy to test bugfixes for regression later on

❖ To measure progress internally
  ➢ project planning
  ➢ work distribution

❖ To report to the customer
  ➢ challenging to communicate a CC process
1. Native syntax
2. Data types
3. Symbol expansion
4. Procedures
5. Code folding
6. System components
1. Debugging
2. Breakpoint
3. Currently executing
4. Watch on a field
5. Watch on a view
6. Locals show known views
TIALAA supports both client & server
Pilot study results

- ~4 MLOC rules,
- ~13 MLOC bind files,
- ~4 MLOC sets,
- ~3 MLOC panels
- ~42k data structures, ~17k programs
- 100% compilation & verification
- Integration testing ongoing
- Go into production in a few months
Conclusion

❖ Follow @grammarware
❖ 4GLs are bad DSLs
❖ TIALAA is there to replace AppBuilder
   ➢  https://www.raincode.com/technical-landscape/tialaa/
❖ Testing a compiler is a lot of work
❖ No out of the box solution
❖ No out of the box comprehensive methodology
❖ Existing papers are scarce and focused
❖ Request for SLEBoK!
   ➢  http://slebok.github.io
❖ Thanks! Questions?
Testing in TIALAA

- G-tests: can the compiler handle the customer’s codebase?
- R-tests: can the parser recognise this input?
- F-tests: can the parser rightfully reject this input?
- P-tests: can the parser construct a good tree from this input?
- N-tests: can the normaliser rewrite this tree well?
- E-tests: can this input error be fixed automatically?
- T-tests: can this program be typed correctly?
- A-tests: can this program be rejected by static semantic analysis?
- C-tests: can this program be successfully compiled to produce a DLL?
- V-tests: can this program be compiled to a verified DLL?
- U-tests: can this problem be rightfully rejected during compilation?
- S-tests: can this program successfully execute to produce output?
- X-tests: can this program throw the right exception?
- D-tests: does this runtime library function work?
Dijkstra vs Goodenough

NOTES ON STRUCTURED PROGRAMMING

by

Prof.dr. Edsger W. Dijkstra

Program testing can be used to show the presence of bugs, but never to show their absence!

TOWARD A THEORY OF TEST DATA SELECTION*

John B. Goodenough
Susan L. Gerhart**
SofTech, Inc., Waltham, Mass.

We prove a fundamental theorem showing that properly structured tests are capable of demonstrating the absence of errors in a program.