

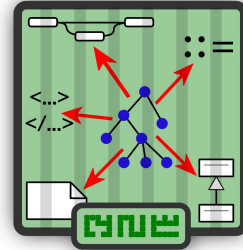
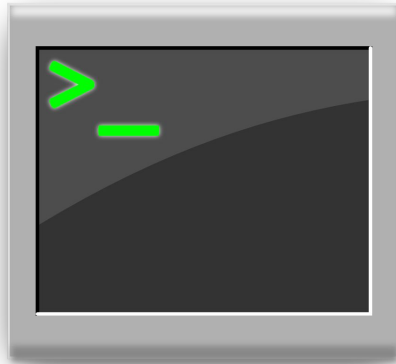
# Parsing @ IDE

V. Zaytsev @ Parsing @ SLE @ SPLASH

raincode **LABS**

———— compiler experts ————

# Grammars in a broad sense



# Grammars in a narrow sense

Which one?

- recognise programs in a language
- parse and interpret
- parse and translate
- parse and compile
- semiparse and analyse
- document
- domain model
- verify & validate

# Grammars for IDE support

- idea from attending PLDI
- What is specific to grammars used in IDEs?
- What IDE features need grammar support?
- How to provide it better?
- OK to be “in a broad sense”

# Main principles

- fast
- partial
- not limited by parsing in a narrow sense

# Syntax highlighting

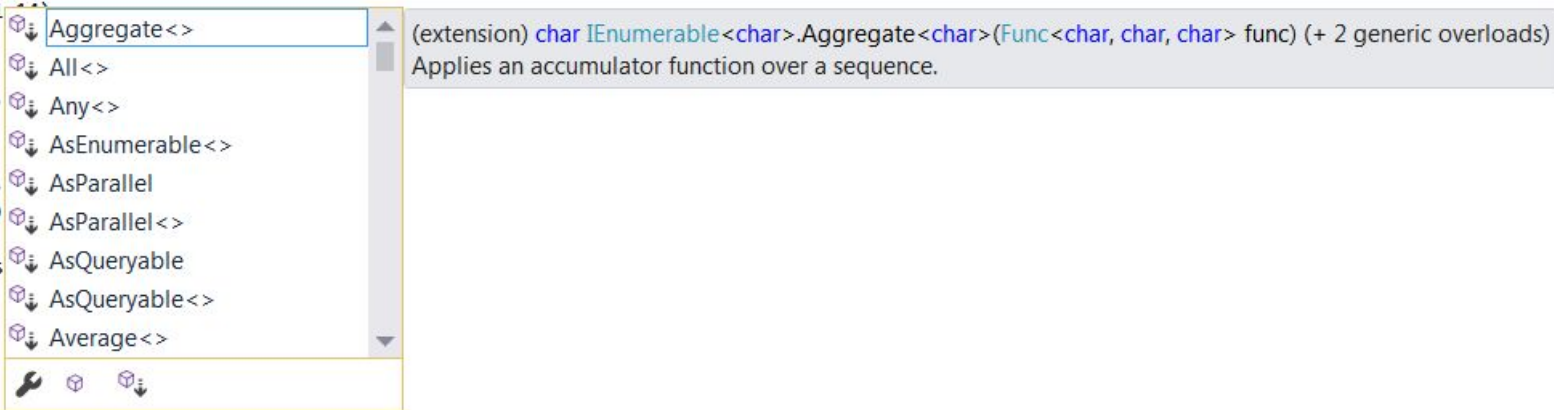
- colour-code tokens
- commonly implemented with regexes
- easy if the tokeniser is precise
- good luck with C++, PL/I, etc
- embryos of common interfaces
- novel solutions possible

```
switch (state)
{
  case 1: // start of the file
    file = new CastleLanguage.Bind.BindFile();
    line = line.PadRight(51);
    if (!line.Begins("$$START "))
    {
      Error(1, line:lnr, pos:"1..8");
      state = 2;
      goto case 2;
    }
    var B1 = line.Substring(8,23);
    var str1_2 = line.Substring(31,14);
    if (str1_2!=" VER54 HPS541 ")
    {
      Error(1, str1_2, line:lnr, pos:"32..45");
      return null;
    }
    var E1 = line.Substring(45,6);
    if (!E1.In("AB30 ", "HPS531"))
    {
      Error(1, E1, line:lnr, pos:"46..51");
      return null;
    }
}
```

# Code completion

- complete keywords
- suggest matching tokens
- guide indentation
- name suggestions
- drop down members

```
var B1 = line.Substring(8,23).;
var str1_2 = line.Substring(31
if (str1_2!=" VER54 HPS541 ")
{
    Error(1, str1_2, line:lnr,
return null;
}
var E1 = line.Substring(45,6);
if (!E1.In("AB30 ", "HPS531")
{
    Error(1, E1, line:lnr, pos
return null;
}
state = 2;
break;
e 2: // ready for a block
```



(extension) `char IEnumerable<char>.Aggregate<char>(Func<char, char, char> func)` (+ 2 generic overloads)  
Applies an accumulator function over a sequence.

# Word selection

- select a word, highlight “the same thing”
- “cheap” visualisation
- liked by devs
- not researched at all

```
res2.ImpName = F2.TrimEnd();
var str2_10 = line.Substring(128,23);
if (str2_10!="")
{
    Error(3, str2_10, line:lnr, pos:"129..151");
    return null;
}
var G2 = line[151];
if (G2=='I')
    res2.Flag1 = true;
else if (G2==' ')
    res2.Flag1 = false;
else
{
    Error(3, G2.ToString(), line:lnr, pos:"152");
    return null;
}
var str2_12 = line[152];
if (str2_12!=' ')
{
    Error(3, str2_12.ToString(), line:lnr, pos:"153");
    return null;
}
var H2 = line.Substring(153,6);
if (!Regex.IsMatch(H2) /* [0-9A-Z]+ */)
{
    Error(3, H2, line:lnr, pos:"154..159");
    return null;
}
```



# Code folding

- blocks in composite statements
  - NOT a solution!
- hierarchical entities
- handful of top constructs?

```
namespace cs1rc
{
  2 references | Vadim Zaytsev, 16 days ago | 2 authors, 23 changes
  internal class Program : CommandLineProgram
  {
    private List<string> LoadedViewNames = new List<string>();
    private Dictionary<string, List<string>> ViewFields = new Dictionary<string, List<string>>();

    public CastleCompilerOptions Options;

    public int ErrorCount = 0;

    1 reference | Vadim Zaytsev, 51 days ago | 1 author, 2 changes
    public Program(string[] args) ...

    0 references | Vadim Zaytsev, 17 days ago | 1 author, 11 changes
    private static int Main(string[] args) ...

    1 reference | Vadim Zaytsev, 16 days ago | 1 author, 15 changes
    private int run() ...

    private List<string> RuleFiles = new List<string>();

    0 references | Vadim Zaytsev, 16 days ago | 2 authors, 14 changes
    protected override void SpecificSupport() ...
  }
}
```

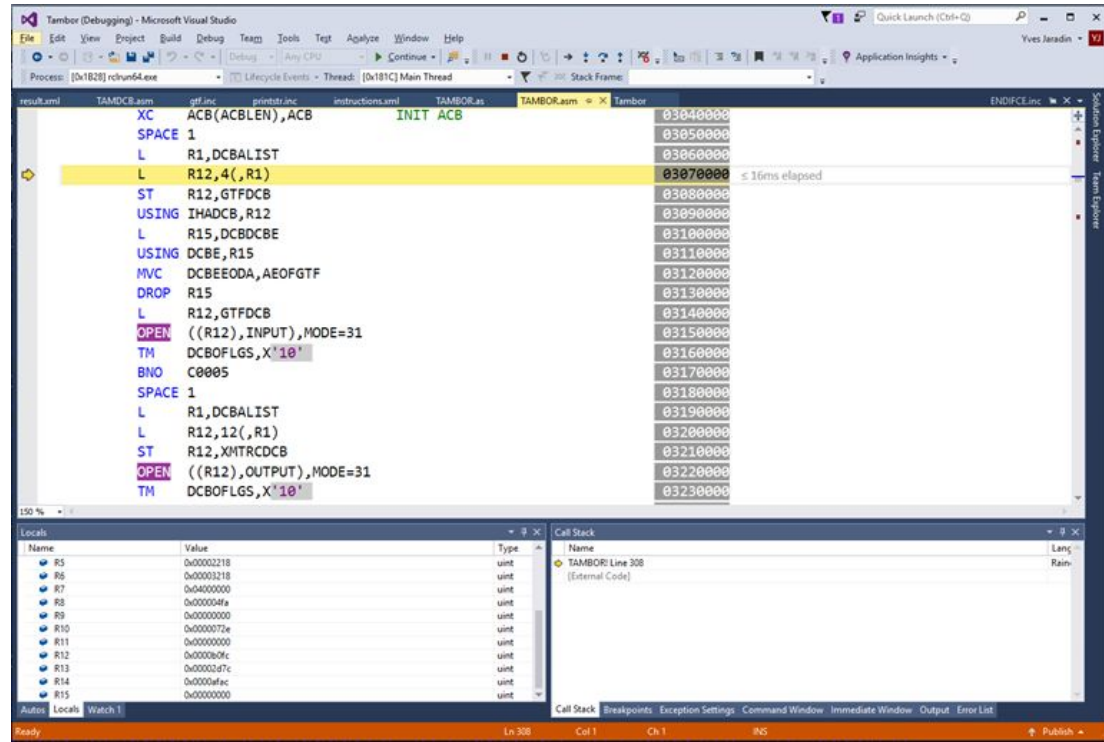
# Visual editing

- let graphs be graphs
- let tables be tables
- let window panels be window panels
- don't let your dreams be dreams!
- projectional?



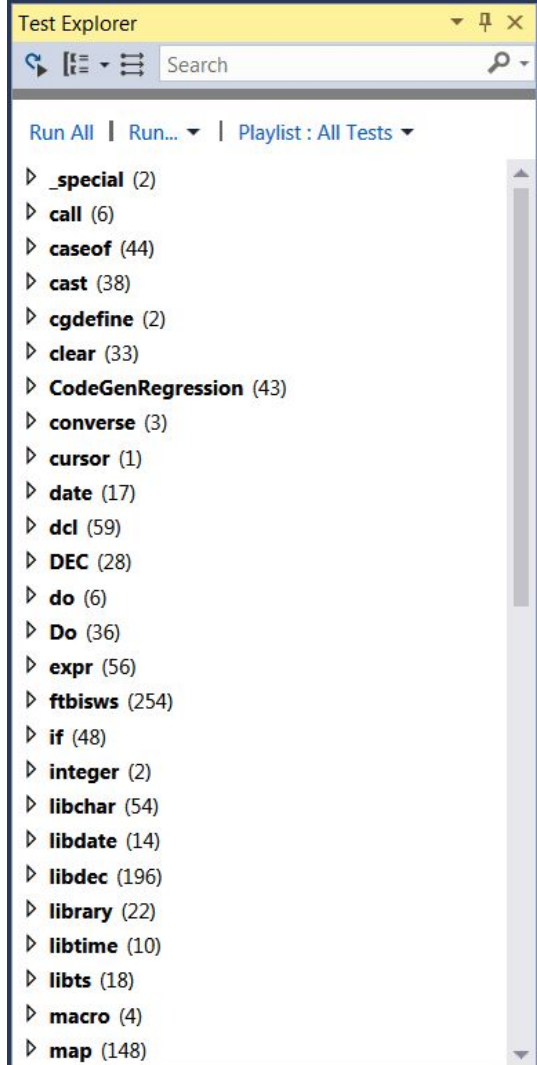
# Debugging

- only relevant for executable programs
- step over
- step into
- breakpoint
- watch
- cross language boundaries



# Testing

- discover tests
- running tests
  - live
- analysing tests
  - green and red
  - coverage
- incrementality
- why not advanced features?
  - model-based
  - test generation
  - fuzzing
  - mutants



# Coding conventions

- formatting
- pretty-printing
- naming
- calling
- templates
- deprecating language constructs
- satisfying global constraints
- smell detection



## Does Python Smell Like Java? Tool Support for Design Defect Discovery in Python

Nicole Vavrova<sup>1</sup> and Vadim Zaytsev<sup>2</sup>

The Art, Science, and Engineering of Programming, 2017, Vol. 1, Issue 2, Article 11

Submission date: 2016-12-01

Publication date: 2017-04-01

DOI: <https://doi.org/10.22552/programming-journal.org/2017/1/11>

Full text: PDF 

### Abstract

The context of this work is specification, detection and ultimately removal of detectable harmful patterns in source code that are associated with defects in design and implementation of software. In particular, we investigate five code smells and four antipatterns previously defined in papers and books. Our inquiry is about detecting those in source code written in Python programming language, which is substantially different from all prior research, most of which concerns Java or C-like languages. Our approach was that of software engineers: we have processed existing research literature on the topic, extracted both the abstract definitions of nine design defects and their concrete implementation specifications, implemented them all in a tool we have programmed and let it loose on a huge test set obtained from open source code from thousands of GitHub projects. When it comes to knowledge, we have found that more than twice as many methods in Python can be considered too long (statistically extremely longer than their neighbours within the same project) than in Java, but long parameter lists are seven times less likely to be found in Python code than in Java code. We have also found that Functional Decomposition, the way it was defined for Java, is not found in the Python code at all, and Spaghetti Code and God Classes are extremely rare there as well. The grounding and the confidence in these results comes from the fact that we have performed our experiments on 32'058'823 lines of Python code, which is by far the largest test set for a freely available Python parser. We have also designed the experiment in such a way that it aligned with prior research on design defect detection in Java in order to ease the comparison if we create our own actions as a replication. Thus, the importance of the work is both in the unique open Python grammar of highest quality, tested on millions of lines of code, and in the design defect detection tool which works on something else than Java.

1. [vavova@gmail.com](mailto:vavova@gmail.com), Universiteit van Amsterdam, Netherlands
2. [vadim@grammarware.net](mailto:vadim@grammarware.net), Raincode Labs, Belgium

### The Journal

About  
Purpose and Operation  
Boards  
Awards  
Publisher  
Volumes

### ISSUES


Volume 2, Issue 1  
Volume 1, Issue 2  
Volume 1, Issue 1

### For Authors

Call for Papers  
Timeline  
Submissions  
Copyright


Article feed (atom)  
Article feed (RSS)

## Language design and implementation for the domain of coding conventions

Full Text:  PDF

Authors: [Boryana Goncharenko](#) [University of Amsterdam, Netherlands](#)  
[Vadim Zaytsev](#) [Raincode, Belgium / University of Amsterdam, Netherlands](#)

Published in:

- Proceeding  [SLE 2016](#) Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering  
Pages 90-104


Amsterdam, Netherlands — October 31 - November 01, 2016

[ACM](#) New York, NY, USA ©2016

[table of contents](#) ISBN: 978-1-4503-4447-0

doi>[10.1145/2997364.2997386](https://doi.org/10.1145/2997364.2997386)



 2016 Article

### Bibliometrics

- Citation Count: 0
- Downloads (cumulative): 89
- Downloads (12 Months): 89
- Downloads (6 Weeks): 5

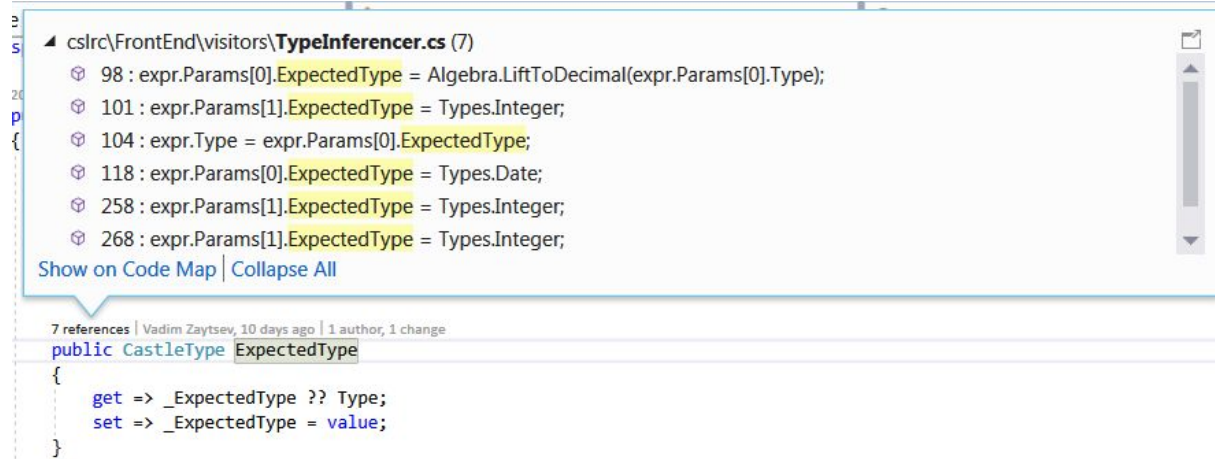
# Refactoring

- recommender systems
- ReSharper, CodeMaid, etc
- grammars are originally rewriting systems
- seldom used for rewriting
- can be insanely complex
- hard to do right
- hard to v&v

```
7 references | Vadim Zaytsev, 10 days ago | 1 author, 1 change
public CastleType ExpectedType
...
set => _ExpectedType = value;
}
}
3 references | Vadim Zaytsev, 46 days ago
public CastleNode()
{
    Type = AbstractTypes.c
}
}
2 references | Vadim Zaytsev, 46 days ago
public CastleNode(CastleNode
{
    if (proto == null)
        return;
}
}
...
public CastleType ExpectedType
public CastleType GetExpectedType()
{
}
public void SetExpectedType(CastleType value)
{
    get => _ExpectedType ?? Type;
    set => _ExpectedType = value;
}
...
Preview changes
```

# Navigating the codebase

- go to definition
- find references
- analyse dependencies
- analyse co-changes



The screenshot shows an IDE window with a code snippet from `cs\src\FrontEnd\visitors\TypeInferencer.cs`. The snippet contains several lines of code where `ExpectedType` is accessed. A callout box highlights these references and provides navigation options. Below the snippet, the definition of `ExpectedType` is shown as a property of `CastleType`.

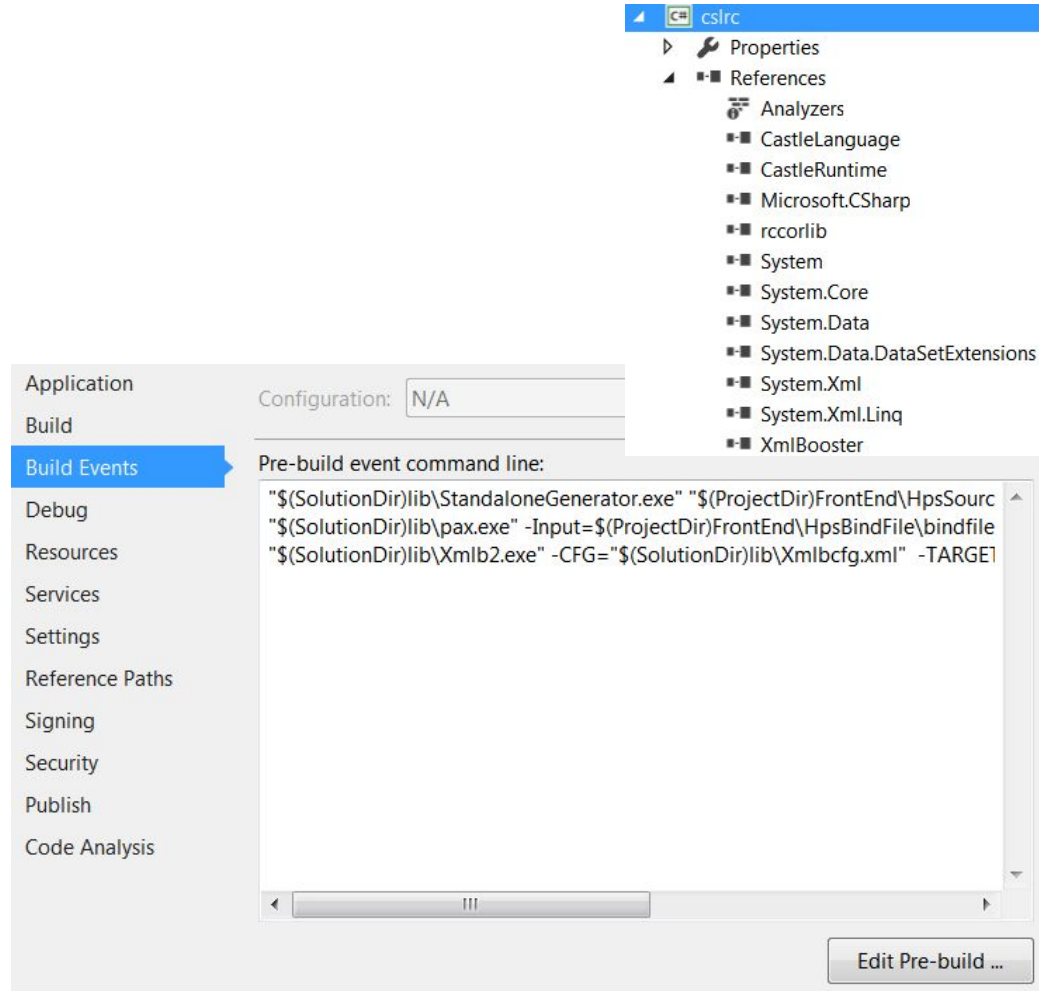
```
cs\src\FrontEnd\visitors\TypeInferencer.cs (7)  
98 : expr.Params[0].ExpectedType = Algebra.LiftToDecimal(expr.Params[0].Type);  
101 : expr.Params[1].ExpectedType = Types.Integer;  
104 : expr.Type = expr.Params[0].ExpectedType;  
118 : expr.Params[0].ExpectedType = Types.Date;  
258 : expr.Params[1].ExpectedType = Types.Integer;  
268 : expr.Params[1].ExpectedType = Types.Integer;  
Show on Code Map | Collapse All
```

7 references | Vadim Zaytsev, 10 days ago | 1 author, 1 change

```
public CastleType ExpectedType  
{  
    get => _ExpectedType ?? Type;  
    set => _ExpectedType = value;  
}
```

# Configuring a build

- compiling
- deploying
- delivering
- versioning
- building in the right order





# Helping

- tooltips
- hover infoboxes
- API guidance
- explaining errors
  - recommending fixes

```
file.Members.Add(block);
```

[🔍] (local variable) [BindMember](#) block

```
x.Substring();
```

▲ 2 of 2 ▼ [string](#) [string](#).Substring(**int** startIndex, **int** length)

Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.

**startIndex:** *The zero-based starting character position of a substring in this instance.*

# Conclusion

- IDEs are built ad hoc
- IDEs are built with a framework bias
- there is [or can be] a class of IDE-specific grammars
- mostly greenfield research
- way beyond [single] grammars [in a narrow sense]
- vastly different user stories
  - JS: live to the extreme
  - C++: many changes, always incomplete info
  - C#: style and paradigm switching
  - PL/I: cache to the extreme (yesterday's trees are good enough)
- Please do it (willing to collab)