

High fidelity source-to-source transformations with parse tree diff

Jurgen Vinju
Centrum Wiskunde & Informatica
Swat.engineering BV
TU Eindhoven

source-to-source transformations; programmatically

- either: language preserving

- refactoring

- quick fix

- formatting

here we care a lot about fidelity

- or: *not* language preserving

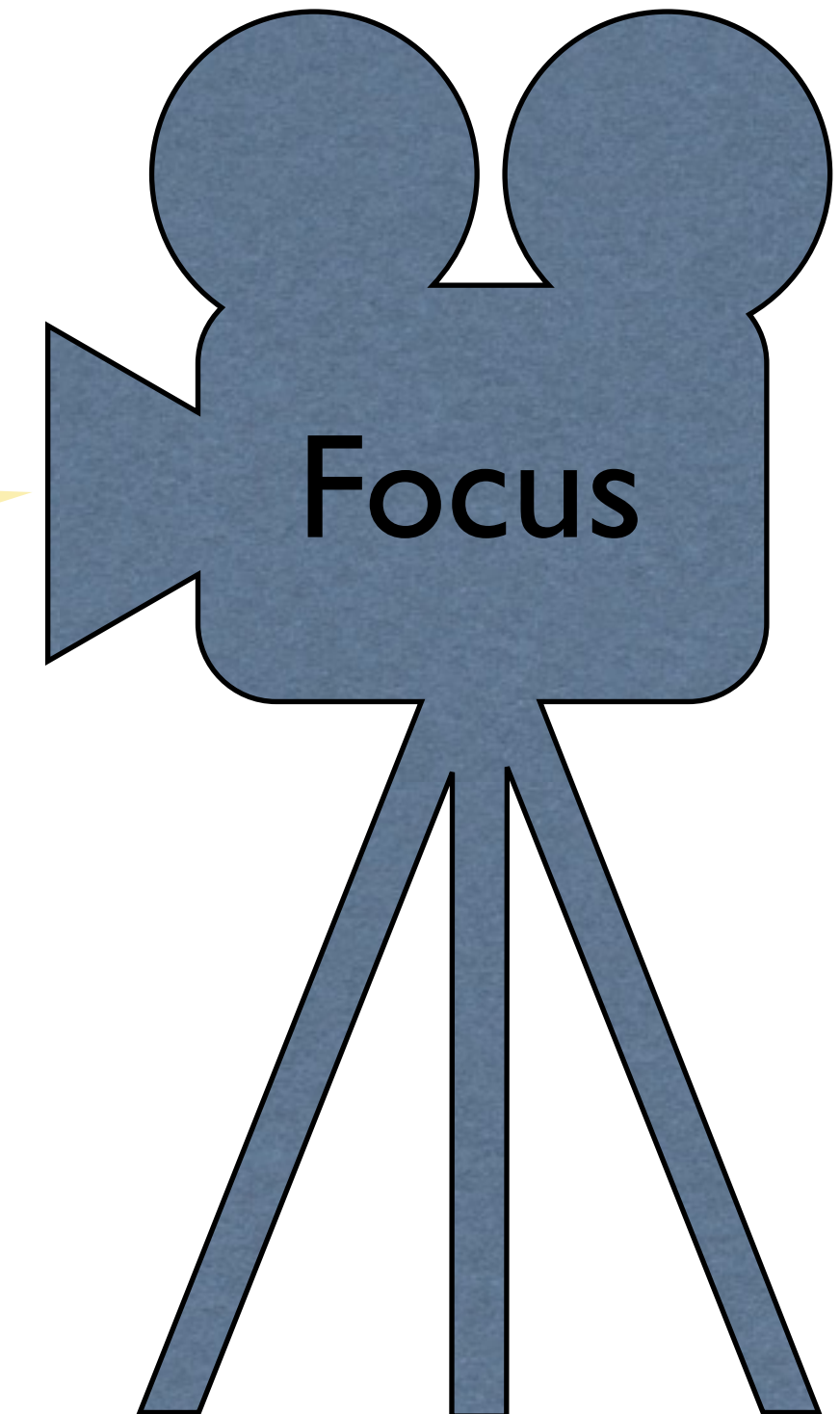
- code generators

- transpilers

- decompilers

- model extractors (reverse engineering tools)

here we care less about fidelity



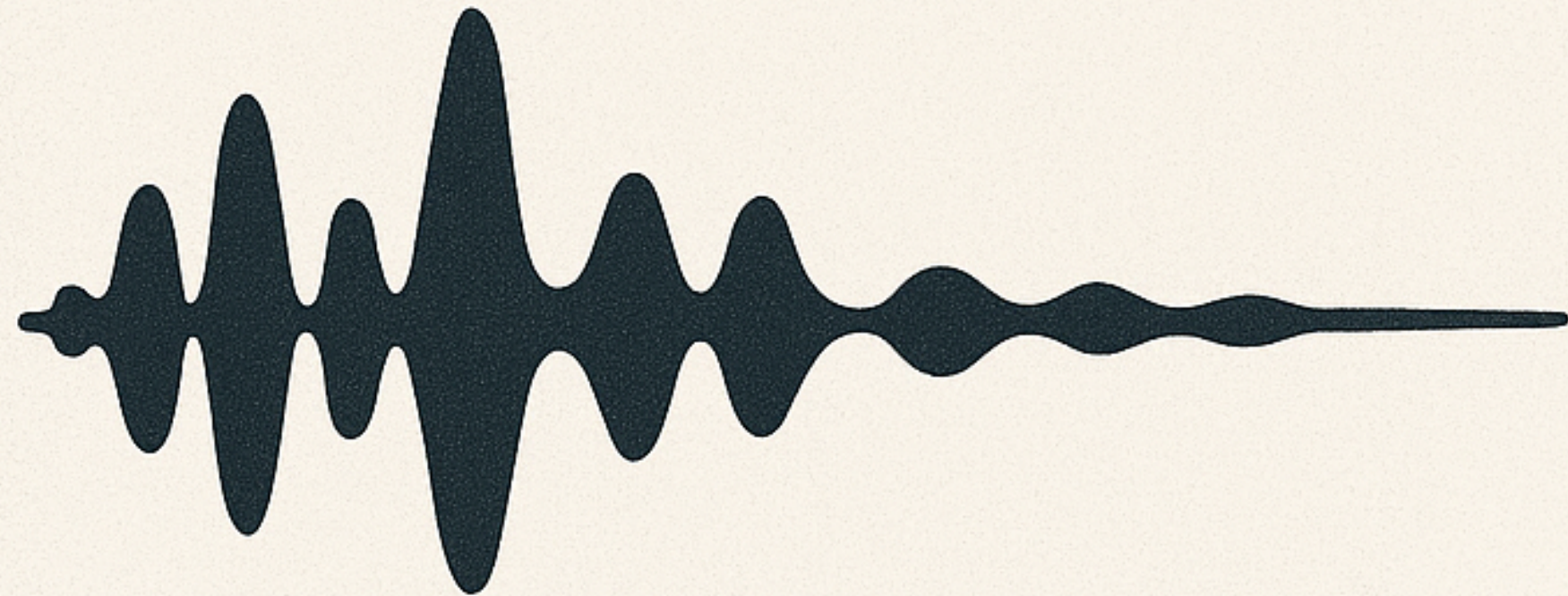
Requirements for source-to-source

- the baseline is “grep and sed”: regular expressions, “search and replace”
 - string manipulation, pretty Hi-Fi by nature...
- better: more “syntactically and semantically exact”
 - beyond regex: parsing and semantic modeling (trees, tables and graphs)
- but we still need: “**high fidelity**”: no new *noise* and not *lossy*
 - *source code comments* and *indentation* !!!!!
 - normalization side-effects: If to if, $i=i+1$ to $i++$, and loss of (brackets)

Quality of source-to-source transformation

lo-fi

LOSSY
and **NOISY**

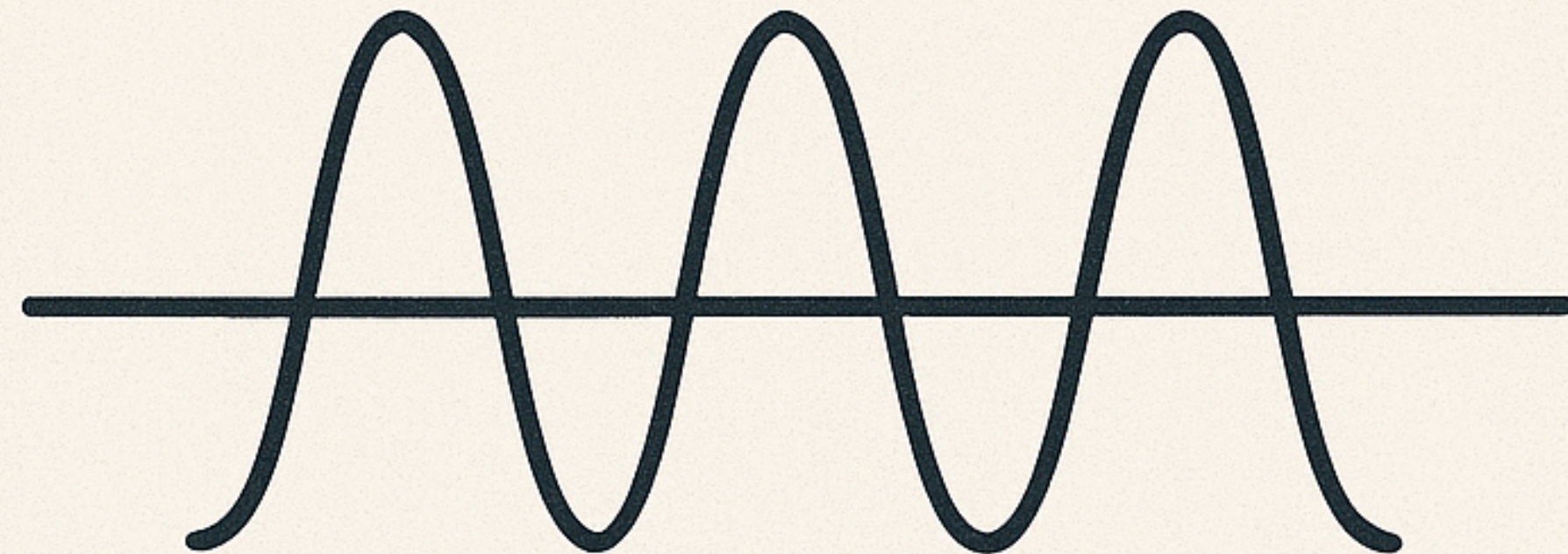


```
if ( condition ){  
  return b;}
```

```
_____  
_____  
_____
```

hi-fi

LOSSLESS
and **NOISELESS**



```
if ( condition ){  
  return b;  
}  
// Check the condit
```

```
_____  
_____
```

Quality of source-to-source transformations

- Quality is: high-fidelity, no *visual* loss and no *visual* noise
- Another quality is: *exactness* (correctness)
 - syntactically *correct*, by using syntax definitions exactly
 - semantically *correct*, by using semantic models exactly
- **How: conditional rewrite rules written in concrete syntax**
 - readable and understandable
 - open to query *any semantic model* via any (pre)conditions
 - correctness is contextual: e.g. refactoring vs. quickfix vs. formatting

example: flip the branches

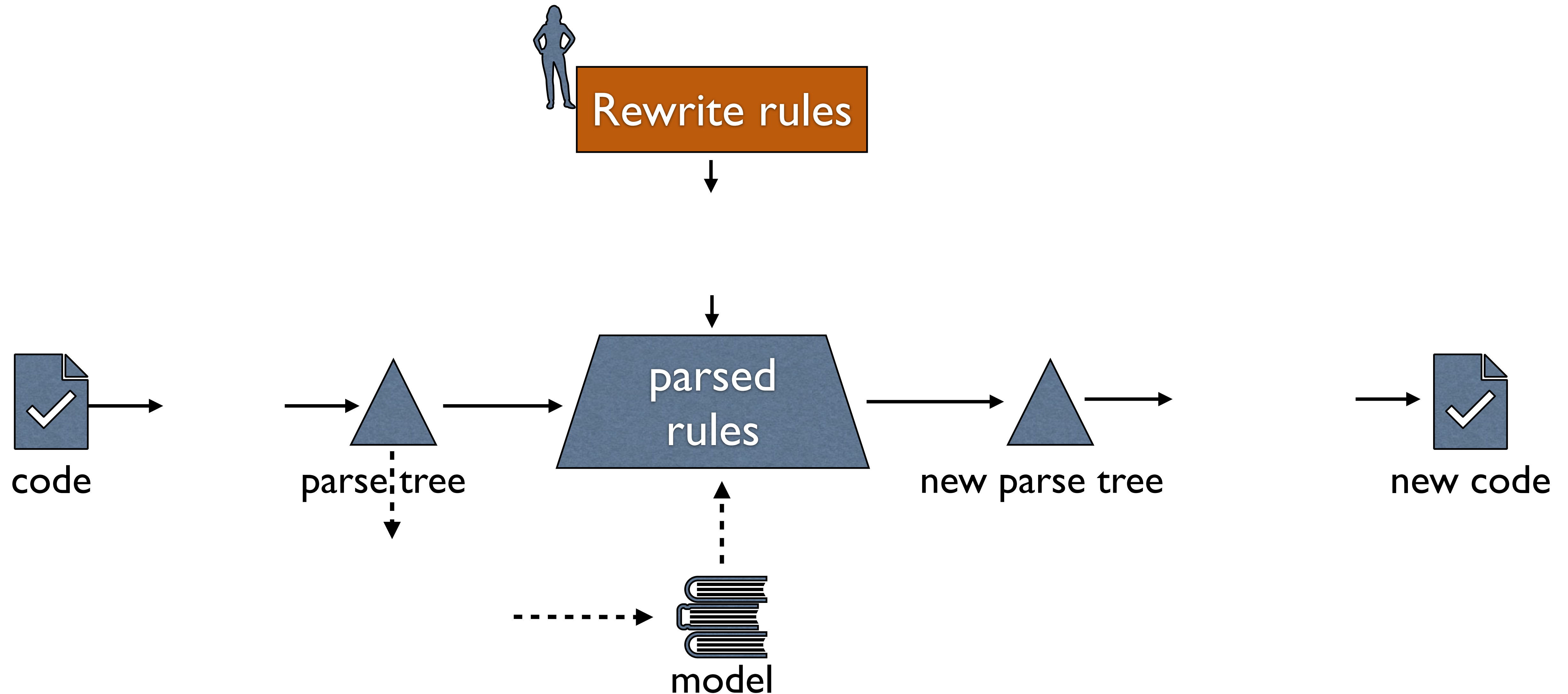
(Statement) `if (!<cond>) { <stats1> } else { <stats2> }`
=>
(Statement) `if (<cond>) { <stats2> } else { <stats1> }`

- pattern matches syntax trees *exactly*
- substitution creates new syntax tree *exactly*
- whitespace and comments are **ignored** during matching
- the **pattern** is great but the **substitution** does not meet our requirements: it's *visually lossy* and **noisy**.

demo this

demo a noisy rewrite

The mechanics of source-to-source



Real metaprogrammers generate “edit scripts”

- as old as the original **diff** (1974) and **patch** programs (1984) in Unix
- fundamental to textual version control: **rcs**, **cvs**, **svn**, **git**, ...
- patching **visual** concrete syntax is also fundamental to visual modeling environments
- The “edit” command: “inputText” $\models$ edit(offset, length, “newSubstring”)
- All collected edit commands executed (in reverse order of offset) produce a new file.
- Limitation: **overlapping** edits have to be staged in separate diff/patch cycles.

Real metaprogrammers generate(d) “edit scripts”

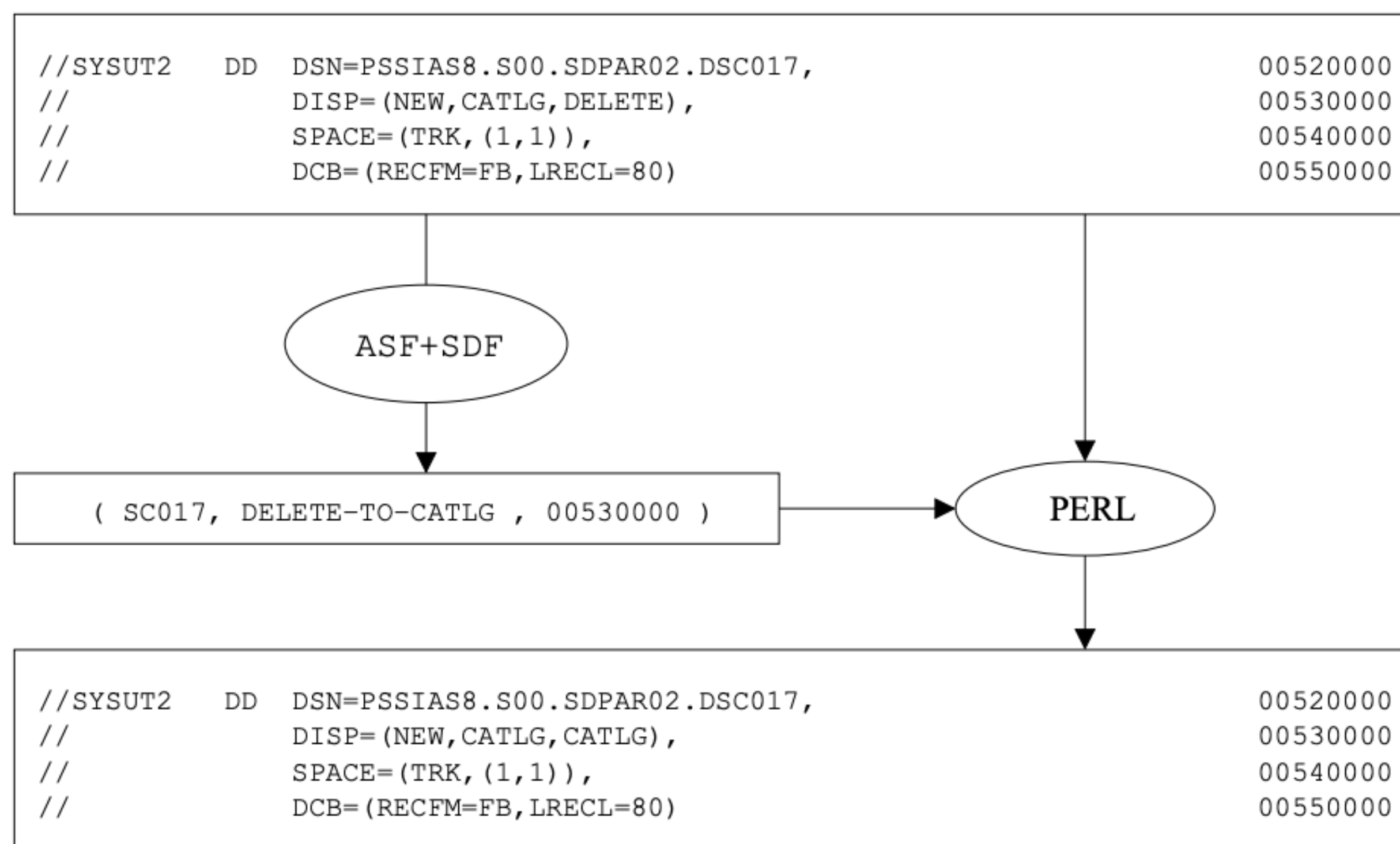


Figure 1: Sample input and output of a source code transformation in JCL. The DELETE keyword is replaced by CATLG, but in a specific context.

RULE 2000: Steven Klusener, Mark van den Brand, YT.

Received: 26 July 2021 | Revised: 22 January 2022 | Accepted: 22 February 2022
DOI: 10.1002/spe.3082

EXPERIENCE REPORT

WILEY

Large-scale semi-automated migration of legacy C/C++ test code

Mathijs T. W. Schuts^{1,2} | Rodin T. A. Aarssen^{3,4} | Paul M. Tielemans¹ | Jurgen J. Vinju^{3,4}

¹Philips, Best, The Netherlands

²Software Science, Radboud University, Nijmegen, The Netherlands

³Software Analysis and Transformation, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

⁴Software Engineering and Technology, Eindhoven University of Technology, Eindhoven, The Netherlands

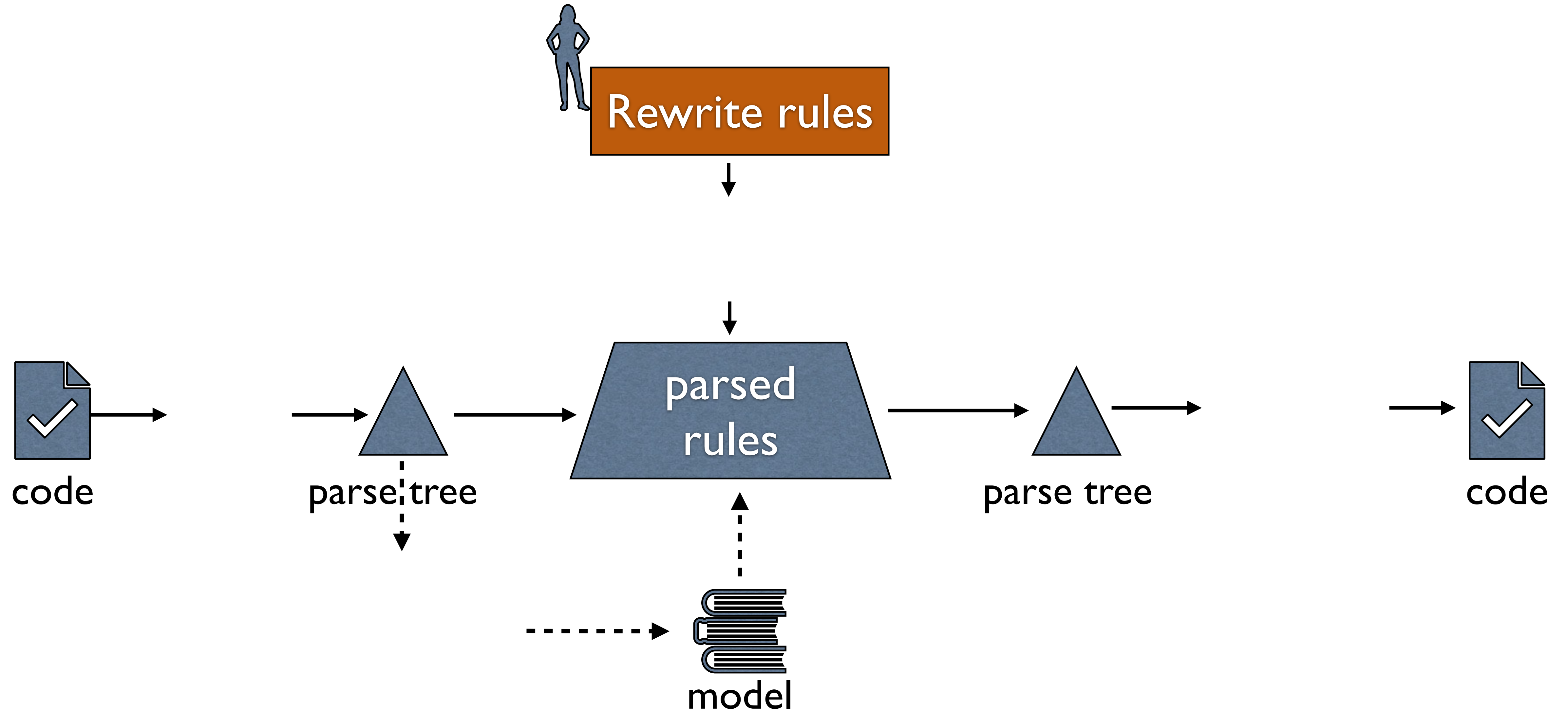
Abstract

This is an industrial experience report on a large semi-automated migration of legacy test code in C and C++. The particular migration was enabled by automating most of the maintenance steps. Without automation this particular large-scale migration would not have been conducted, due to the risks involved in manual maintenance (risk of introducing errors, risk of unexpected rework, and loss of productivity). We describe and evaluate the method of automation we used on this real-world case. The benefits were that by automating analysis

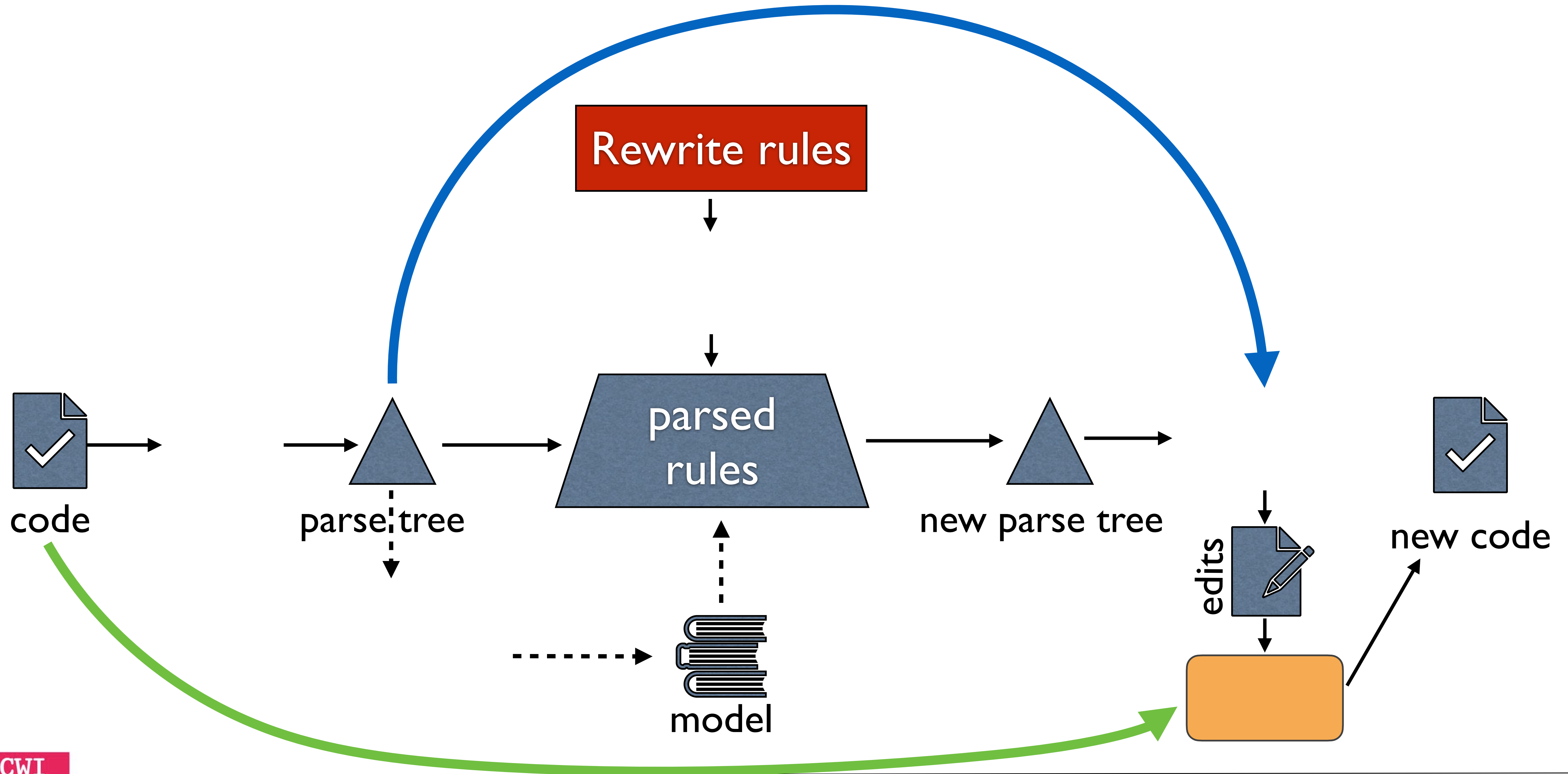
Observation:

All industrial applications generate lists of patches rather than rewriting parse trees. Duh.

The mechanics of source-to-source



Rewrite mechanics with “treeDiff”



demo a hifi rewrite

treeDiff algorithm

- Pseudocode of treeDiff: given an original and new (sub)tree (ot, nt):

if (ot == nt) **done**;

if (ot.rule != nt.rule) **then collect** `edit(ot.offset, ot.length, nt.text)`

if (ot.rule **is list**) **then with** [prefixOt, commonSublist, postfixOt] **and** [prefixNt, commonSublist, postfixNt]
recurse(prefixOt, prefixNt) **and recurse**(postfixOt, postfixNt)

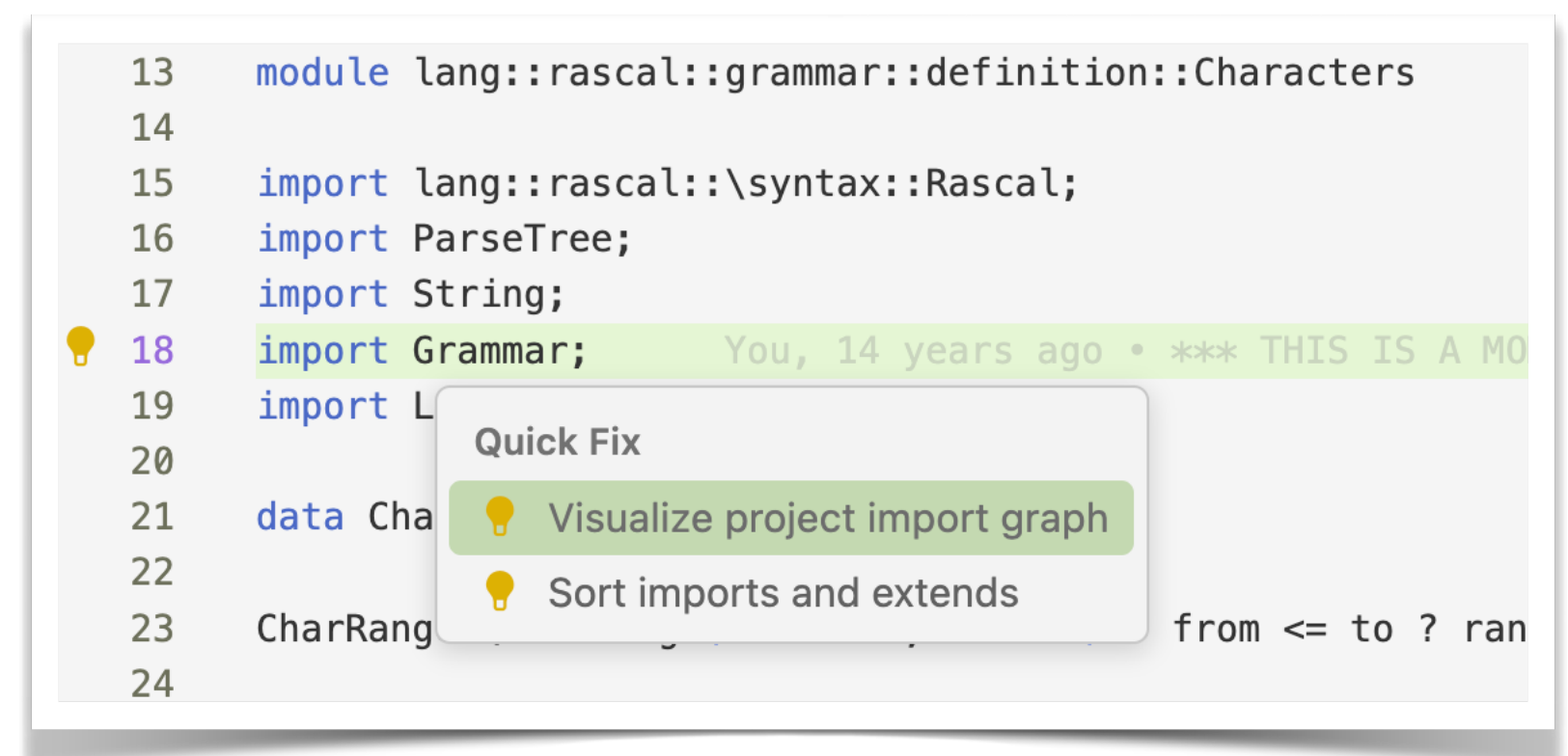
if (ot.rule == nt.rule) **then for all** (*non-layout*) children otc **and** ntc: **pairwise recurse**(otc, ntc)

- add original indentation to the replacement: `edit(ot.offset, ot.length, learnIndentation(ot.text, nt.text))`
- commonSubexpression detection similar to commonSublist (not shown today)
- *language parametric and hifi*: lossless concrete parse trees are the key enabler [Vinju 2005]

Conclusions

do you have
questions?

- A new keystone for the source-to-source pipeline: `parse` | `rewrite` | `treeDiff`
 - `treeDiff`: non-whitespace *code* edits, infers *indentation* from the original
 - `layoutDiff`: non-code *whitespace* edits, infers *comments* from the original
- HiFi has always been a **critical requirement** for source-to-source tools
- Patch API is great for DSL/PL **UX** (LSP, VScode, Eclipse, ...)
 - formatting, quickfix, refactoring, lenses, undo, preview, side-by-side, ...
- Room for improvement... even more comment preservation.
- HiFi `treeDiff` **improves pull request reviews**
- Comment and indentation preservation are no longer a concern for the language engineer



```
13 module lang::rascal::grammar::definition::Characters
14
15 import lang::rascal::\syntax::Rascal;
16 import ParseTree;
17 import String;
18 import Grammar;
19 import L
20
21 data Cha
22
23 CharRang
24
```

Quick Fix

- Visualize project import graph
- Sort imports and extends

You, 14 years ago • *** THIS IS A MO

from <= to ? ran