Refactoring for Dynamic Languages

Rafael Reia

Instituto Superior Técnico
Universidade de Lisboa

26th Jun, 2015

Outline

Outline

Motivation

Motivation

```
(define (fibs n)
  (let ((fibs
          (let loop ((previous 0)
                     (current 1)
                     (index 0))
            (if (= index n)
                (list)
                (cons current
                      (loop current
                            (+ previous current)
                            (+ index 1)))))))
    (for ((fib (in-list fibs)))
      (displayln fib))))
```

Motivation

```
(define (fibs n)
  (let ((fibs
          (let loop ((previous 0)
                     (current 1)
                     (index 0))
            (if (= index n)
                (list)
                (cons current
                      (loop current
                            (+ previous current)
                            (+ index 1)))))))

    (for ((fib (in-list fibs)))
      (displayln fib))))
```

Motivation

```
(define (print-list fibs)
  (for ((fib (in-list fibs)))
    (displayln fib)))
```

```
(define (fibs n)
  (let ((fibs
          (let loop ((previous 0)
                     (current 1)
                     (index 0))
            (if (= index n)
                (list)
                (cons current
                      (loop current
                            (+ previous current)
                            (+ index 1)))))))
    (print-list fibs)))
```

Motivation

```
(define (print-list fibs)
  (for ((fib (in-list fibs)))
    (displayln fib)))

(define (fibs n)
  (let ((fibs

        (let loop ((previous 0)
                   (current 1)
                   (index 0))
          (if (= index n)
              (list)
              (cons current
                    (loop current
                          (+ previous current)
                          (+ index 1)))))))

    (print-list fibs)))
```

Motivation

```
(define (compute-fibonacci n)
  (let loop ((previous 0)
             (current 1)
             (index 0))
    (if (= index n)
        (list)
        (cons current
              (loop current
                    (+ previous current)
                    (+ index 1))))))
```

```
(define (print-list fibs)
  (for ((fib (in-list fibs)))
    (displayln fib)))

(define (fibs n)
  (let ((fibs
```

```
         (compute-fibonacci n)))
```

```
    (print-list fibs)))
```

Copy Paste - What can go wrong

Copy Paste - What can go wrong

Motivation

- Lack of Refactoring tools for Dynamic Languages

- Lack of Refactoring tools adequate to beginners

Objectives

- Correct

- Useful

- Simple to use

Refactoring Correctness

- Formal Approach

- Informal Approach

Refactoring tools - Classification
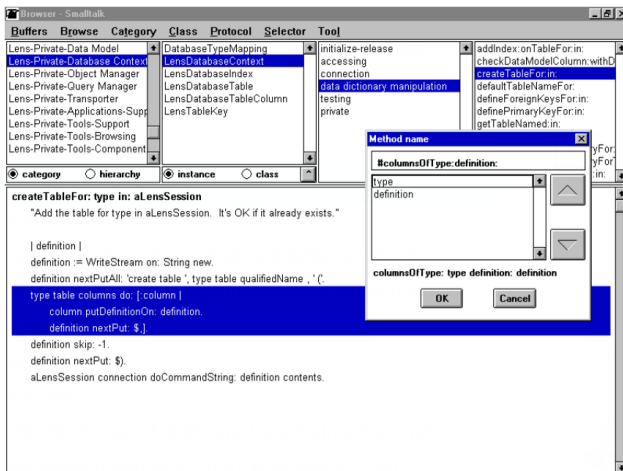
- Manual

- Semi-Automatic

- Fully-Automatic

Outline

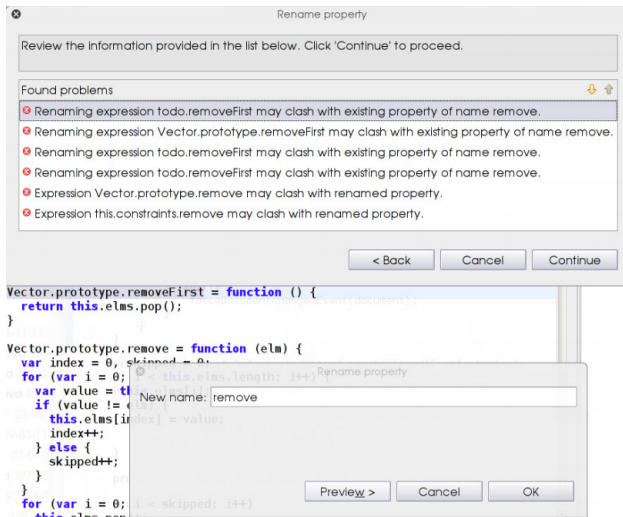# How we Refactor

- Done continuously

- Differs among users

- Refactoring tools are underused

Scheme
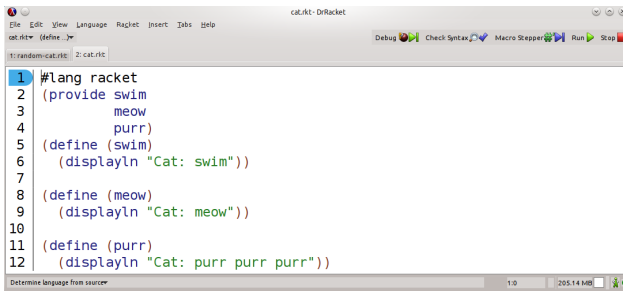
```scheme
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```
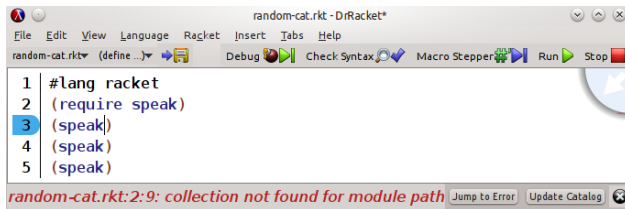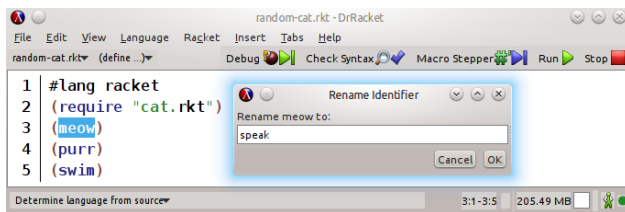
SmallTalk

JavaScript

Racket

DrRacket's Rename

Outline

Architecture
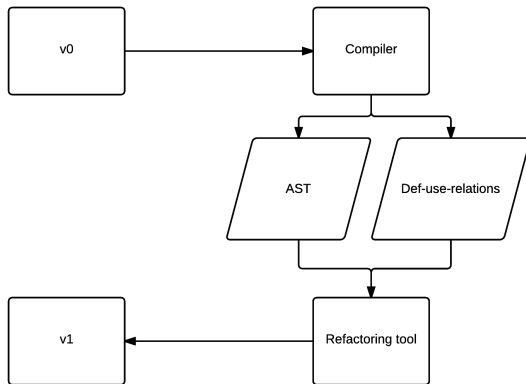
Validation

- Extract function

- Rename

- Add-Prefix

Validation - Extract function

```
1   #lang racket
2   (define (fibs n)
3     (let ((fibs
4           (let loop ((previous 0)
5                      (current 1)
6                      (index 0))
7             (if (= index n)
8                 (list)
9                 (cons current
10                      (loop current
11                            (+ previous current)
12                            (+ index 1)))))))
13      (for ((fib (in-list fibs)))
14        (displayln fib))))
```

```
1   #lang racket
2   (define (fibs n)
3     (let ((fibs
4           (let loop ((previous 0)
5                      (current 1)
6                      (index 0))
7             (if (= index n)
8                 (list)
9                 (cons current
10                      (loop current
11                            (+ previous current)
12                            (+ index 1)))))))
13      (for ((fib (in-list fibs)))
14        (displayln fib))))
```
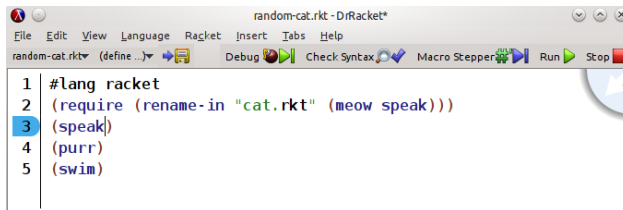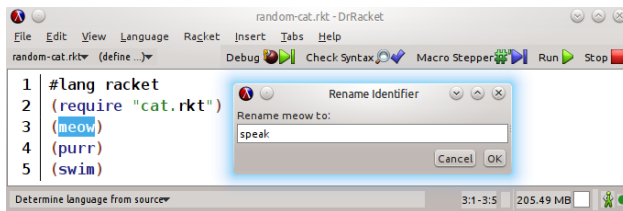
Validation - Extract function

```racket
1   #lang racket
2   (define (print-list fibs)
3     (for ((fib (in-list fibs)))
4       (displayln fib)))
5   (define (fibs n)
6     (let ((fibs
7            (let loop ((previous 0)
8                       (current 1)
9                       (index 0))
10            (if (= index n)
11                (list)
12                (cons current
13                      (loop current
14                            (+ previous current)
15                            (+ index 1)))))))
16      (print-list fibs)))
```

Validation - Rename

Validation - Add prefix

```
1  #lang racket
2  (require plot3d)
3  (define (xyz->pos p)
4    (pos (cx p) (cy p) (cz p)))
5  (define (xyz->dir p)
6    (dir (cx p) (cy p) (cz p)))
7  (define (pos->dir p)
8    (dir (pos-x p) (pos-y p) (pos-z p)))
```

```
1  #lang racket
2  (require (prefix-in pct: plot3d))
3  (define (xyz->pos p)
4    (pct:pos (cx p) (cy p) (cz p)))
5  (define (xyz->dir p)
6    (pct:dir (cx p) (cy p) (cz p)))
7  (define (pos->dir p)
8    (pct:dir (pct:pos-x p) (pct:pos-y p) (pct:pos-z p)))
```

Evaluation

- Refactoring Correctness

- Usability and Simplicity

Thank you!

Questions?