A Plea for Pluggable Programming Language Features

Bernhard G. Humm, Darmstadt University of Applied Sciences
Ralf S. Engelschall, Capgemini sd&m Research, Germany

5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2010)
Athens, Greece
July 23rd 2010
Strict demarcation between programming languages, e.g., typing

**Dynamic Typing**
- Examples: Smalltalk, Scheme, Python, PHP
- Advantages:
  - More concise
  - More expressive
  - More re-usable
  - Less expensive

**Static Typing**
- Examples: C/C++, Java, C#
- Advantages:
  - Earlier error detection
  - Better performance
  - Better documentation
  - Better IDE support

Business Information System
- Code Generation
- Script
- Entire application
- Final Product
- Alpha Release

Bernhard G. Humm, Darmstadt University of Applied Sciences: Pluggable Programming Language Features. ENASE 2010
Pluggable programming language features

- Independent
- Optional
- Extensible
- Configurable

Parameter Typing
- Dynamic
- Static

Visibility (public, private)
- Optional
- Mandatory

Parameter Mode: i, o, i/o
- Optional
- Mandatory

Pre-/Postconditions
- Optional
- Mandatory

Exceptions
- Optional
- Mandatory

Side effects: r, w, r/w
- Optional
- Mandatory

Documentation
- Optional
- Mandatory

Bernhard G. Humm, Darmstadt University of Applied Sciences: Pluggable Programming Language Features. ENASE 2010
Programming language feature configuration per component

| Parameter Typing |  |  |
|------------------|---------------------------|
| Dynamic          | Static                   |

| Visibility       |  |  |
|------------------|---------------------------|
| Optional         | Mandatory                |

| Pre-/Postconditions |  |  |
|---------------------|---------------------------|
| Optional            | Mandatory                |

| Documentation      |  |  |
|---------------------|---------------------------|
| Optional            | Mandatory                |
# Programming language feature configuration per development stage

<table>
<thead>
<tr>
<th>Development stage</th>
<th>Operation Names</th>
<th>Parameter Names</th>
<th>Parameter Modes</th>
<th>Parameter Obligation</th>
<th>Parameter Types</th>
<th>Visibility</th>
<th>Exceptions</th>
<th>Type Restrictions</th>
<th>Pre-/Postconditions</th>
<th>Side Effects</th>
<th>Semantics and Parameter Documentation</th>
<th>Non-functional Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof of Concept</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alpha Release</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beta Release</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Product</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: interface specification
create customer (Prototype)

(define-function create-customer (name address date-of-birth))
Example: Static Parameter Typing (Alpha Release)

```
(define-function create-customer
  ((name :type Structured-Name)
   (address :type Structured-Address)
   (date-of-birth :type Date))
  (:result-type Customer))
```

Parameter Types

Result type
Example: Pre- / Postconditions (Beta Release)

```
(define-function create-customer
  ((name :type Structured-Name)
   (address :type Structured-Address)
   (date-of-birth :type Date))
  (:result-type Customer
   :pre (is-valid? address)
   :pre (lies-in-past? date-of-birth)
   :pre "No duplicate of previously created cust."
   :post (get-id result)))
```

Pre-conditions

Post-condition
Example: full documentation (Final product)

(define-function create-customer
  ((name :type Structured-Name :documentation "Customer name consists of ...")
   (address :type Structured-Address :documentation "Postal address ...")
   (date-of-birth :type Date :documentation "Customer birth date"))
  (:result-type Customer :result-documentation "New Customer object"
   :pre (is-valid? address)
   :pre (lies-in-past? date-of-birth)
   :pre "No duplicate of previously created customer"
   :post (get-id result) :documentation "Creates a new Customer object")

Parameter documentation
Result documentation
Operation documentation
Conclusions

• Programming languages today:
  - Strict demarcation:
    rapid application development vs. industry scale development
  - fixed set of features per programming language

• Advantages of pluggable programming language features:
  - Reduced coding overhead where possible
  - Increased quality where needed
  - Incremental development

• New trends:
  - Dynamic language on mainstream platforms,
    e.g., Python, Ruby, Groovy, Scala (JRE); F# and C# 4.0 (.NET)
  - Hybrid typing, e.g. VisualBasic, Perl 6, C# 4.0
A Plea for Pluggable Programming Language Features

Bernhard G. Humm
Darmstadt University of Applied Sciences
Darmstadt, Germany
Bernhard.Humm@h-da.de