How do Python Programmers Use Python?

Python Dynamicity & Other Ideas

Beatrice Åkerblom

Department of Computer and Systems Sciences
Stockholm University
beatrice@dsv.su.se
Statically typed languages

Lawful evil

Chaotic good

Dynamically typed languages

Lawful good

Chaotic evil
“Historically” in Language Research

• Type inference (Smalltalk, Various Python projects, Diamondback Ruby)
• Gradual typing (e.g. Siek, Taha)
• Soft typing (e.g. Fagan)
• Pluggable types (e.g. Bracha)

• Generally tries to make dynamic languages more “controllable” and predictable, that is static
• Assumptions are made about how programs are developed
Approaches used before

Selected examples:

• “Usually, no further properties are defined after the initialization and the type of the properties rarely changes.”
  -- Peter Thiemann

• “Giving people a dynamically-typed language does not mean that they write dynamically-typed programs”
  -- John Aycock

• “Yet while the presence of such abundant dynamism makes traditional static optimization impossible, in most programs, there is surprisingly little dynamism present.”
  -- Michael Salib
Approaches used before

Selected examples:

- “Usually, no further properties are defined after the initialization and the type of the properties rarely changes.”
  -- Peter Thiemann

- “Giving people a dynamically-typed language does not mean that they write dynamically-typed programs”
  -- John Aycock

- “Yet while the presence of such dynamism makes traditional static optimization impossible, in most programs, there is surprisingly little dynamism present.”
  -- Michael Salib
When/Where, How & Why (if at all) is the dynamic power of dynamic languages used in real applications?
What’s Dynamic?

• Dynamic features - use of introspection, reflection, dynamic code evaluation

• Duck typing - polymorphism without need for inheritance or declared interfaces

• Dynamic objects - how dynamic are class and object structures
What’s Dynamic?

• Dynamic features - use of introspection, reflection, dynamic code evaluation

• Duck typing - polymorphism without need for inheritance or declared interfaces

• Dynamic objects - how dynamic are class and object structures

what is the program?
do our objects reflect the class definitions?
how dynamic are variable accesses, etc?
how common is dynamic code generation?
What’s Dynamic?

- Dynamic features - use of introspection, reflection, dynamic code evaluation
- Duck typing - polymorphism without need for inheritance or declared interfaces
- Dynamic objects - how dynamic are class and object structures

do variables change type? will different paths lead to different types? how polymorphic are method calls? can common supertypes be found?
What’s Dynamic?

• Dynamic features - use of introspection, reflection, dynamic code evaluation

• Duck typing - polymorphism without need for inheritance or declared interfaces

• Dynamic objects - how dynamic are class and object structures

how stable is the OO (objects, classes, inheritance structures) of Python programs? do we find interface-like structures in Python programs?
Why is this important?

We’ll be able to:

• know how much of a “typical” Python program could (or could not) be annotated with types

• know how well Python source code does represent the running program

• know to what extent we need to support dynamic behaviour e.g. when building tools or new language constructs for Python

• emphasize the focus on how Python is used when designing new constructs
Different Sources, different methods

• Programs (Quantitative)
  – Static analysis (what is the program?)
  – Dynamic analysis: Measure behaviour at runtime, e.g. use of language constructs, inheritance hierarchies, polymorphic call sites, etc.

• Code snippets (Qualitative)
  – Search for language constructs usage patterns
  – Read to understand how/why

• Programmers (Sociological)
  – Interview
  – Observe
What Have We Done?

• Modified the Python 2.6 interpreter to log information about running programs
  – class creation
  – method and function calls
  – instance member access
  – use of dynamic features
• Python programs selected from Source Forge
• Programs run on a Debian machine
  – interactive
  – tests
  – examples
• Program runs documented
  – tests
  – recordings
  – use cases
Dynamic Features in Python Programs

- Anamos, Bleachbit, Comix, ConvertAll, Exaile, Kodos, Mcomix, Pysolfc, Rednotebook, Retext, Sbackup, Solfege, Task coach, Torrent Search, Wikidpad, Zmail

- hasattr, eval, reload, getattr, __delattr__, __getattr__, execfile, __getattribute__, del attribute, __import__, exec, setattr, vars, __setattr__, delattr

0: Id-nummer
1: the path, filename and row number from which the call was made,
2: Caller id.
3: Caller type.
4: Target Id
5: Target type
6: Feature name
7: Argument types
8: Results
What about Holkner & Harland’s “Evaluating the dynamic behaviour of Python applications”?
Number of Features Used by Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bleachbit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convertall</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elltube</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kodos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mcomix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photofilmstrip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pysol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiotray</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rednotebook</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retext</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sbackup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solfege</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Taskcoach</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TorrentSearch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wikidpad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zmail</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 = hasattr, 2 = getattr, 3 = .import_, 4 = setattr, 5 = exec, 6 = _getattr_, 7 = del attribute, 8 = eval, 9 = vars, 10 = _setattr_, 11 = delattr, 12 = _delattr_, 13 = _getattr_, 14 = execfile, 15 = reload
Distribution of Dynamism for All Traces

- Distribution of dynamic features over libraries and program-specific code during start-up and run-time

<table>
<thead>
<tr>
<th>Run-time</th>
<th>Startup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 16%</td>
<td>Program 15%</td>
</tr>
<tr>
<td>Library 29%</td>
<td>Library 40%</td>
</tr>
</tbody>
</table>
Number of Programs Where Features Were Traced
Median, Average, Minimum and Maximum for All Features

<table>
<thead>
<tr>
<th>Per program dynamic feature usage</th>
<th>Median</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entire programs</td>
<td>5.8 K</td>
<td>390 K</td>
<td>214</td>
<td>6.7 M</td>
</tr>
<tr>
<td>Library start-up</td>
<td>674</td>
<td>4.5 K</td>
<td>81</td>
<td>56 K</td>
</tr>
<tr>
<td>Library run-time</td>
<td>883</td>
<td>350 K</td>
<td>0</td>
<td>6.6 M</td>
</tr>
<tr>
<td>Program-specific start-up</td>
<td>508</td>
<td>3.2 K</td>
<td>0</td>
<td>33 K</td>
</tr>
<tr>
<td>Program-specific run-time</td>
<td>154</td>
<td>33 K</td>
<td>0</td>
<td>610 K</td>
</tr>
</tbody>
</table>
Polymorphism in Python Programs


0. Event ID
1. Source file path
2. Caller ID (current this at the call-site)
3. Caller type
4. Target ID (the receiver of the method call)
5. Class name of target + : + class id
6. Name of called function/method
7. Argument types
8. Call line
9. A list of all super classes of the target type
Questions Asked

- How many unique call-sites?
- How many call-sites are monomorphic?
  - Trivially monomorphic vs. monomorphic
- How many polymorphic call-sites?
- Distribution of the degree of polymorphism seen

- For call-sites that saw several different types as receiver, what were the types and do they share a common supertype containing the method called?
Monomorphic Call Sites

- Trivially monomorphic: We have only recorded one single execution of this call site
- Monomorphic: We have recorded more than one execution of this call site, and the types seen were always the same

Polymorphic 4%  Trivially mono 51%

Truly mono 45%
How Polymorphic are Python Call Sites?

Programs listed ordered by size from smallest to largest
How Polymorphic are Python Call Sites?

Number of call-sites in all programs plotted for number of receiver types (2-n)
References

• Phillip Heidegger and Peter Thiemann, “Recency types for analyzing scripting languages”, ECOOP 2010.