Configuration Migration in Evolving Software Product Lines

Leonardo Passos  
Generative Software Development Lab.  
University of Waterloo

June, 2012
Evolving SPLs (ESPLs)

- In evolving SPLs (e.g.: Linux), users migrate configurations.

- Two types of migration:
  - Forward
  - Backward
Why do forward migration?
In Linux

v2.6.32  →  v2.6.33
Why do forward migration?

- New functionality

\[ \text{card}(\text{features(v2.6.33)} \setminus \text{features(v2.6.32)}) - \text{merged} = 274 \text{ NEW} \]

[arch = x86_64]

v2.6.33: RealTek 8192 Wireless Driver
(commit: 8fc8598e61f6f384f3eaf1d9b09500c12af47b37)
Why do forward migration?

- Reliability
  - Newer kernels have fixed bugs

  v2.6.33: fixed contractory cache line size config options (commit: 350f8f5631922c7848ec4b530c111cb8c2ff7caa)

- Performance
  - Newer kernels have redundant code/features removed

  v2.6.33: removal of the AS IO scheduler (⊆ CFQ IOS) (commit: 492af6350a5ccf087e4964104a276ed358811458)
Why do backward migration?
In Linux

v2.6.32  →  v2.6.33
Why do backward migration?

- Old (deprecated) functionality
  - ANDROID drivers are not in v2.6.33, but are in v2.6.32

- Reliability: newer releases introduce new bugs
  - https://bugs.launchpad.net/bugs/bugtrackers/linux-kernel-bugs

- Performance: older features may have better performance
  - Bad SSD performance with recent kernels
    https://lkml.org/lkml/2012/1/30/202
What Changes in ESPLs? (Fwd. migration)

- The feature model (FM)
- Features' constraints + CTCs
- Code + build artifacts
No Traceability

- SPLs may not keep traceability links for its features between different versions.

(SPLs without feature traceability)
No Traceability

- Forward migration: v2.6.32 to v2.6.33: 43 unmapped features

- Release window fragment (x86_64):

```
<table>
<thead>
<tr>
<th>v2.6.37</th>
<th>v2.6.38</th>
<th>v2.6.39</th>
<th>v3.0</th>
<th>v3.1</th>
<th>v3.2</th>
<th>v3.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7256 features)</td>
<td>(7419 features)</td>
<td>(7586 features)</td>
<td>(7703 features)</td>
<td>(7777 features)</td>
<td>(7909 features)</td>
<td>(8036 features)</td>
</tr>
</tbody>
</table>
```
No traceability means...
No traceability means

- No control whether functionality is being lost and how much of it is lost.

It does not necessarily mean removed!
No Traceability Means

- No compatibility check (forward migration):
  - Full compatibility: can all products in $old$ can be generated in $new$?
  - Partial compatibility 1: can all products in $old$ that consist only of mapped features be generated in $new$?
  - Partial compatibility 2: can $new$ generate products containing only features in $old$, but no covered by $old$?
  - Instance compatibility: is the $old$ configuration compatible with the $new$ SPL?
  - Approximate instance compatibility: is there a configuration in $new$ that approximates to the $old$ configuration?
Proposal

1) Feature traceability
   - Retrieval of feature mapping from commit logs
   - For SPLs with no such data, create such mapping possibly based on
     a) Feature name
     b) Feature description
     c) Feature constraints + CTCs

2) Formal framework for forward migration compatibility analysis (full/partial/instance/approximate compatibility).

3) Main case study: Linux
Questions