Towards a Catalog of Variability Evolution Patterns – The Linux Kernel Case

Leonardo Passos  
University of Waterloo  
lpassos@gsd.uwaterloo.ca

Krzysztof Czarnecki  
University of Waterloo  
kczarneck@gsd.uwaterloo.ca

Andrzej Wasowski  
IT University of Copenhagen  
wasowski@itu.dk
In evolving variant-rich software...
In evolving variant-rich software...

• New features are added
In evolving variant-rich software...

- New features are added
- Features are removed
In evolving variant-rich software...

- New features are added
- Features are removed
  - feature is no longer supported: complete removal

Examples:
- merge
- split
- rename
- Constraints are changed, etc.
In evolving variant-rich software...

- New features are added
- Features are removed
  1. feature is no longer supported: complete removal
  2. feature continues to be supported, but its abstraction is no longer present (disappears from the variability model).
In evolving variant-rich software...

- New features are added
- Features are removed
  1. feature is no longer supported: complete removal
  2. feature continues to be supported, but its abstraction is no longer present (disappears from the variability model).

Examples:
In evolving variant-rich software...

- New features are added
- Features are removed
  1. feature is no longer supported: complete removal
  2. feature continues to be supported, but its abstraction is no longer present (disappears from the variability model).

Examples:
- merge
In evolving variant-rich software... 

- New features are added
- Features are removed
  1. feature is no longer supported: complete removal
  2. feature continues to be supported, but its abstraction is no longer present (disappears from the variability model).

Examples:

- merge
- split
In evolving variant-rich software...

- New features are added
- Features are removed
  1. feature is no longer supported: complete removal
  2. feature continues to be supported, but its abstraction is no longer present (disappears from the variability model).

Examples:
- merge
- split
- rename
In evolving variant-rich software...

- New features are added
- Features are removed

  1. feature is no longer supported: complete removal

  2. feature continues to be supported, but its abstraction is no longer present (disappears from the variability model).

Examples:

- merge
- split
- rename

- Constraints are changed, etc.
Example
(from Linux)
Ralink Drivers

RT2860
Ralink Drivers

RT2860

Complete removal?
Existing evolution studies tend to focus on the variability model alone
That doesn’t tell the whole story...
Ralink Drivers

... RT2860 RT3090 ...

Configuration space
Ralink Drivers

... RT2860 RT3090 ...

Implementation space
Spaces are connected...
Ralink Drivers

... RT2860 RT3090 ...

...
Ralink Drivers

... RT2860  RT3090 ...

7/28
Ralink Drivers

... RT2860 RT3090 ...

--

...
Ralink Drivers

RT2860

RT3090

...
With the three spaces in mind, the real picture of . . .
Ralink Drivers

RT2860

RT3090

is

is
Ralink Drivers

... RT2860 RT3090 ...

...
Ralink Drivers

RT2860

RT3090

RT3090 is merged into RT2860

8/28
RT3090 is merged into RT2860
We want to know...
How do the three spaces evolve together in real world variant rich software?
How do the three spaces evolve together in real world variant rich software?

Focus: features that disappear from the configuration space
Two goals

Understand the evolution of the three spaces in a real-word variant rich software
Two goals

Understand the evolution of the three spaces in a real-word variant rich software

Document our understanding in the form of evolution patterns (preliminary).
Our subject of analysis
Qualities of Linux as a subject of study

• Mature: over 20 years since its first release
Qualities of Linux as a subject of study

- Mature: over 20 years since its first release
- Complex: over 6,000 features
Qualities of Linux as a subject of study

• Mature: over 20 years since its first release

• Complex: over 6,000 features

• Changes are kept in a publicly available SCM Repository (git)
Qualities of Linux as a subject of study

- Mature: over 20 years since its first release
- Complex: over 6,000 features
- Changes are kept in a publicly available SCM Repository (git)
- Continuous development
Qualities of Linux as a subject of study

- Mature: over 20 years since its first release
- Complex: over 6,000 features
- Changes are kept in a publicly available SCM Repository (git)
- Continuous development
- Contains multiple spaces:
Qualities of Linux as a subject of study

- Mature: over 20 years since its first release
- Complex: over 6,000 features
- Changes are kept in a publicly available SCM Repository (git)
- Continuous development
- Contains multiple spaces:
  - configuration space: Kconfig
Qualities of Linux as a subject of study

- Mature: over 20 years since its first release
- Complex: over 6,000 features
- Changes are kept in a publicly available SCM Repository (git)
- Continuous development
- Contains multiple spaces:
  - configuration space: Kconfig
  - compilation space: Makefile
Qualities of Linux as a subject of study

- Mature: over 20 years since its first release
- Complex: over 6,000 features
- Changes are kept in a publicly available SCM Repository (git)
- Continuous development
- Contains multiple spaces:
  - configuration space: Kconfig
  - compilation space: Makefile
  - implementation space: C code
Variability evolution patterns from Linux
Data collection & Analysis

• Data collection is limited to three pairs of stable kernel releases in x86_64

• For each pair, we considered only the features that disappeared from the configuration space

• Manual analysis of 140 removals from a total of 220 (63%)
Infrastructure

- Extraction and reuse of Kconfig parsing infrastructure from Linux itself
  - allow us to compute disappearing features among each release kernel
- Conversion of Linux patches from git into a relational database
  - allow us to quickly identify which commit erases a feature from the configuration space
- git log + gitk, grep: visualize and search logs
Extracting patterns is hard!

Difficulties in analyzing patches when collecting patterns:

- unrelated changes (noise)
- technical comments (too much jargon)
- extensive set of changes
- everything is recorded in the SCM as addition/removal of lines (too low level)
Four identified patterns

• Optional feature to implicit mandatory

• Computed attributed feature to code

• Merge features by module aliasing

• Optional feature to kernel parameter

Template: structure, instance and discussion
Four identified patterns

- Optional feature to implicit mandatory
- Computed attributed feature to code
- Merge features by module aliasing
- Optional feature to kernel parameter

Template: structure, instance and discussion
Optional feature to implicit mandatory
if Y, compile Y.c into Y.o
    compile X.c into X.c
Y.c #ifdef Y
    ...
#endif
CTC

X

... ...
Y

(Before)

if Y, compile Y.c into Y.o
    compile X.c into X.c
Y.c #ifdef Y
    ...
#endif
CTC

X

... ...
Y

(After)

instance: X = OCFS, Y = OCFS Access Control List
if Y, compile Y.c into Y.o
compile X.c into X.c

Y.c

Y

CTC

Y

#ifdef Y
    ...
#endif

X

X

(X,Y)

CTC[X,Y]

#ifdef X
    if X,
    compile Y.c into Y.o
    compile X.c into X.c

Y.c

(Y.c)

(Before)

(After)
Structure & Instance

(X)

if Y,
    compile Y.c into Y.o
    compile X.c into X.c
Y.c #ifdef Y
    ...
#endif
CTC

(X Y)

if Y, if X,
    compile Y.c into Y.o
    compile X.c into X.c
Y.c #ifdef X
    ...
#endif
CTC_{X \setminus Y}

Instance: X = OCFS, Y = OCFS Access Control List
Structure & Instance

```
if Y,
    compile Y.c into Y.o
    compile X.c into X.c
Y.c #ifdef Y
    ...
#endif 
CTC
```

(Before)

```
if X,
    compile Y.c into Y.o
    compile X.c into X.c
Y.c #ifdef X
    ...
#endif 
CTC_{X\setminus Y}
```

(After)
Structure & Instance

Instance: $X = \text{OCFS}, \ Y = \text{OCFS Access Control List}$
Pattern should be used when:

- users should not be given the freedom to configure $Y$
  - e.g.: they may inadvertently forget to select it, as in Access Control List ($Y$)

- $Y$ is a critical feature that makes sense to exist in the software, given the presence of its parent $X$
Our patterns have direct implications...
Direct implications

- Existing evolution studies (She et al. at Vamos’10, Lotufo et. al. at SPLC’10) focus on the variability model alone: our patterns show that features can be erased from the configuration space, while still present in the implementation space.

- Our patterns capture situations not covered by the existing SPL evolution theory (Borba et al. at ITAC’10)
  - compatibility of product is not guaranteed (evolution is not safe)
Conclusions
Conclusions

• Evolution must focus on all spaces

• We presented 4 patterns extracted from Linux

• Our patterns explain the evolution of features removed from the configuration space

• They show evolution steps not captured in previous studies (both theoretical and empirical).
Future work
Future work

- Collect patterns not restricted to removals
- Measure frequency
- Study other systems
Thanks for listening!