Feature-Oriented Software Evolution
(Vision paper)

Leonardo Passos¹
Andrzej Wąsowski³

Krzysztof Czarnecki¹
Christian Käster⁴

Sven Apel²
Jianmei Guo¹
Claus Hunsen²

¹University of Waterloo  ²University of Passau  ³IT University  ⁴CMU

The Seventh International Workshop on Variability Modelling of Software-intensive Systems
Software evolves...
Example

Automotive embedded software:

• Changing regulations
  ◦ ABS is now mandatory in the EU

• Market differentiating enhancements
  ◦ Electronic stability control (ESC) improves ABS by preventing skidding

• New technology availability
  ◦ Laser-based distance sensors are more precise than radio-based ones
Example

Automotive embedded software:

• Changing regulations
Example

Automotive embedded software:

• Changing regulations
  ◦ ABS is now mandatory in the EU
Example

Automotive embedded software:

• Changing regulations
  ◦ ABS is now mandatory in the EU

• Market differentiating enhancements
Example

Automotive embedded software:

- Changing regulations
  - ABS is now mandatory in the EU

- Market differentiating enhancements
  - Electronic stability control (SC) improves ABS by preventing skidding
Example

Automotive embedded software:

- Changing regulations
  - ABS is now mandatory in the EU

- Market differentiating enhancements
  - Electronic stability control (SC) improves ABS by preventing skidding

- New technology availability
Example

Automotive embedded software:

- Changing regulations
  - ABS is now mandatory in the EU

- Market differentiating enhancements
  - Electronic stability control (SC) improves ABS by preventing skidding

- New technology availability
  - Laser-based distance sensors are more precise than radio-based ones
Understanding the evolution in place is not easy...
Scenario

ABS + SC
Scenario

- Integration can scatter different artifacts
Scenario

- Integration can scatter different artifacts
- Different levels of abstractions not mastered by all stakeholders
Scenario

- Integration can scatter different artifacts
- Different levels of abstractions not mastered by all stakeholders
Scenario

- Integration can scatter different artifacts
- Different levels of abstractions not mastered by all stakeholders

Management level

System level

Code level

Developers
Scenario

• Integration can scatter different artifacts
• Different levels of abstractions not mastered by all stakeholders

⇐ System engineers
Scenario

- Integration can scatter different artifacts
- Different levels of abstractions not mastered by all stakeholders

← Project managers
In practical settings...
In practical settings...

Complex and large software systems have:
In practical settings... 

Complex and large software systems have:

- Diverse set of stakeholders
In practical settings...

Complex and large software systems have:

- Diverse set of stakeholders
- Diverse set of artifacts
In practical settings...

Complex and large software systems have:

- Diverse set of stakeholders
- Diverse set of artifacts
- Different stakeholders have particular “views” over the software
In practical settings...

Complex and large software systems have:

- Diverse set of stakeholders
- Diverse set of artifacts
- Different stakeholders have particular “views” over the software

Stakeholders need a common meeting point
Otherwise... (no common meeting point)
Otherwise...  
(no common meeting point)  

Ineffective communication
Otherwise... 
(no common meeting point)

Ineffective communication  Software flaws
Otherwise... 
(no common meeting point)

Ineffective communication    Software flaws

Architecture decay
Otherwise...  
(no common meeting point)

Ineffective communication  Software flaws

Architecture decay  Higher maintenance costs
Hypothesis
Hypothesis

Managing evolution at the level of features can address the challenges describe above
Hypothesis

Arguments favouring the hypothesis:
Hypothesis

Arguments favouring the hypothesis:

- Feature = cohesive requirements bundle
Hypothesis

Arguments favouring the hypothesis:

• Feature = cohesive requirements bundle

• Requirements are a common point among all stakeholders
Hypothesis

Arguments favouring the hypothesis:

• Feature = cohesive requirements bundle

• Requirements are a common point among all stakeholders

• Features are more coarse-grained than individual requirements
Hypothesis

Arguments favouring the hypothesis:

• Feature = cohesive requirements bundle

• Requirements are a common point among all stakeholders

• Features are more coarse-grained than individual requirements
  ◦ Facilitates understanding
Hypothesis

Arguments favouring the hypothesis:

• Feature = cohesive requirements bundle

• Requirements are a common point among all stakeholders

• Features are more coarse-grained than individual requirements
  ○ Facilitates understanding

• Evolution can be put in simple terms
Hypothesis

Arguments favouring the hypothesis:

• Feature = cohesive requirements bundle
• Requirements are a common point among all stakeholders
• Features are more coarse-grained than individual requirements
  ○ Facilitates understanding
• Evolution can be put in simple terms
  ○ Add new feature, retire old ones, etc.
Our vision

(Assuming the validity of our hypothesis)
Feature-oriented evolution based on:

Tracing
Feature-oriented evolution based on:

Tracing

Analyses
Feature-oriented evolution based on:

- Tracing
- Analyses
- Recommendations
Purpose of our work

Research agenda based on our vision for feature-oriented software evolution

This presentation covers part of that agenda (see paper for more details)
Motivating example
Motivating example

Feature model

Car

BRA

SC

YRS

Conv

ABS

YRS-M₂

YRS-M₁

SC ⇒ ABS

Conv ⇒ ¬ SC

YRS ⇔ SC
Motivating example

Feature model

Car

BRA

SC

YRS

Conv

ABS

YRS-M_2

YRS-M_1

SC \implies \text{ABS}

\text{Conv} \implies \neg \text{SC}

YRS \iff \text{SC}

Merge + clone yaw rate prediction
Motivating example

Feature model

Car

SC  ⇒  ABS
Conv  ⇒  ¬ SC
YRS  ⇔  SC

BRA

SC

YRS

Conv

ABS

YRS-M_{2}
Motivating example

Feature model

Merge YRS-M₂ into YRS + rename YRS to YS
Motivating example

Feature model

Car
  → BRA
  ↓
  ↓
  SC
  →
  ↓
  YS

SC → ABS
Conv → SC
YS ↔ SC

Motivating example
Tracing
Tracing

(t₁)  YRS-M₁  YRS-M₂

(t₂)  YRS-M₂

(t₃)  YS
Tracing

(t₁) YRS-M₁ YRS-M₂

(t₂) YRS-M₂

(t₃) YS

Bug found in YS
Tracing

Does the bug exist in YRS-M$_2$ ($t_2$)?
Does the bug exist in YRS-M$_{1/2}$ ($t_1$)?
Tracing

(t₁)  (t₂)  (t₃)

YRS-M₁  YRS-M₂  YRS-M₂  YS

Does the bug exist in both \( t_1 \) and \( t_2 \)?
Tracing

Answering requires tracing the evolution of single features
Tracing

- Traceability has to be recovered from a multi-space setting:
Tracing

- Traceability has to be recovered from a multi-space setting:
  - Recover traceability of different artifacts (e.g.: FM, Build files, C code)
Tracing

- Traceability has to be recovered from a multi-space setting:
  - Recover traceability of different artifacts (e.g.: FM, Build files, C code)
  - Integrate the evolution history of those artifacts over time
Tracing

- Traceability has to be recovered from a multi-space setting:
  - Recover traceability of different artifacts (e.g.: FM, Build files, C code)
  - Integrate the evolution history of those artifacts over time
  - Draw an evolution history (timeline)
Tracing

- Traceability has to be recovered from a multi-space setting:
  - Recover traceability of different artifacts (e.g.: FM, Build files, C code)
  - Integrate the evolution history of those artifacts over time
  - Draw an evolution history (timeline)
Tracing
(Research questions)
Tracing (Research questions)

- Tracing certain artifacts can be daunting
Tracing (Research questions)

- Tracing certain artifacts can be daunting
  - Individual build rules in build files (e.g., make is Turing-complete)
Tracing (Research questions)

- Tracing certain artifacts can be daunting
  - Individual build rules in build files (e.g., make is Turing-complete)
  - Fine-grained variability analysis in code is costly
Tracing (Research questions)

- Tracing certain artifacts can be daunting
  - Individual build rules in build files (e.g., make is Turing-complete)
  - Fine-grained variability analysis in code is costly

RQ: How to recover traceability links in build files and source code in variability-aware systems?
Tracing (Research questions)

- Tracing certain artifacts can be daunting
  
  - Individual build rules in build files (e.g., *make* is Turing-complete)
  
  - Fine-grained variability analysis in code is costly

**RQ:** How to recover traceability links in build files and source code in variability-aware systems?

**RQ:** Once recovered, how to update them to reflect the temporal evolution in place?
Tracing (Research questions)

• Different artifacts = different sources to draw the evolution in place
Tracing (Research questions)

- Different artifacts = different sources to draw the evolution in place
  - Mailing lists
Tracing (Research questions)

- Different artifacts = different sources to draw the evolution in place
  - Mailing lists
  - Commit patches and log messages
Tracing (Research questions)

- Different artifacts = different sources to draw the evolution in place
  - Mailing lists
  - Commit patches and log messages
  - Bug reports in bug tracking systems
Tracing (Research questions)

- Different artifacts = different sources to draw the evolution in place
  - Mailing lists
  - Commit patches and log messages
  - Bug reports in bug tracking systems

**RQ: Which sources are trustworthy?**
Analyses
Analyses
(Back to the motivating example)
Analyses

After the evolution of the SPL, stakeholders noticed that:

• Maintenance is taking longer
• Productivity has decreased
• Bugs are starting to rise

Well-known phenomena of software aging
Analyses

After the evolution of the SPL, stakeholders noticed that:

- Maintenance is taking longer
Analyses

After the evolution of the SPL, stakeholders noticed that:

• Maintenance is taking longer
• Productivity has decreased
Analyses

After the evolution of the SPL, stakeholders noticed that:

- Maintenance is taking longer
- Productivity has decreased
- Bugs are starting to rise
Analyses

After the evolution of the SPL, stakeholders noticed that:

• Maintenance is taking longer
• Productivity has decreased
• Bugs are starting to rise

Well-known phenomena of *software aging*
Analyses

We envision three analyses to prevent aging:
Analyses

We envision three analyses to prevent aging:

- Consistency checking analysis
Analyses

We envision three analyses to prevent aging:

• Consistency checking analysis

• Change impact analysis
Analyses

We envision three analyses to prevent aging:

• Consistency checking analysis
• Change impact analysis
• Architectural analysis
Analyses
(Consistency checking)
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts

```c
// ifdef Conv
// switch
// to Conv
// if ABS
// fails
#endif

sensor_data_t data;
#ifdef SC
    data = get_value(data);
#endif
default:
    if (data->check_oversteering())
        react_oversteering();
#endif

#ifdef SC && YRS_M1
    double predicted_value;
#else
    int p = 0;
#endif

predicted_value = p->get();
```
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts

```c
#ifdef Conv
   // switch
   // to Conv
   // if ABS
   // fails
#endif

sensor_data_t data ;
#ifdef SC
   data = get_value(data) ;
#endif
if (data->check_oversteering())
   react_oversteering() ;

#ifdef SC && YRS_M1
   double predicted_value
   predictor_t p ;
#else
   int p = 0;
#endif
predicted_value=p->get() ;
```

Dead code
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts

Null pointer exception
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts

```c
... #ifdef Conv
   // switch
   // to Conv
   // if ABS
   // fails
#else
#endif
...

sensor_data_t data ;
#ifdef SC
   data = get_value(data) ;
#endif
if (data->check_oversteering())
    react_oversteering() ;
...

#ifdef SC && YRS_M1
   double predicted_value
   ...
#endif
...

 prepaid_value=p->get() ;
```
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts

```
abs.c (1)                      abs.c (2)                      abs.c (3)                      abs.c (4)
... #ifdef Conv
    // switch
    // to Conv
    // if ABS
    // fails
#endif ...
... sensor_data_t data ;
    #ifdef SC
        data = get_value(data) ;
        #endif
    if (data->check_oversteering())
        react_oversteering() ;
    ...
... #ifdef SC && YRS_M1
    double predicted_value ...
    #endif
... #ifdef SC && YRS_M1
    predictor_t p ;
    #else
        int p = 0;
    #endif
    ...
    predicted_value=p->get() ;
```
Analyses (Consistency checking)

Goal: prevent inconsistencies in different artifacts

Other types of analysis exist: e.g., model-checking
Consistency checking
(Research questions)
Consistency checking (Research questions)

- Variability aware-analysis is costly.
Consistency checking (Research questions)

- Variability aware-analysis is costly.

**RQ:** Do existing approaches for variability-aware type-checking, flow-analysis and model-checking scale to large systems?
Consistency checking (Research questions)

• Variability aware-analysis is costly.

  RQ: Do existing approaches for variability-aware type-checking, flow-analysis and model-checking scale to large systems?

• Existing flow-analysis is intra-procedural.
Consistency checking (Research questions)

• Variability aware-analysis is costly.

  RQ: Do existing approaches for variability-aware type-checking, flow-analysis and model-checking scale to large systems?

• Existing flow-analysis is intra-procedural.

  RQ: How to adapt existing inter-procedural analyses to handle variability?
Analyses
(Impact analysis)
Impact analysis

Goal: assess impact of changes
Impact analysis

Goal: assess impact of changes

Scenario:

• To identify bugs, stakeholders in our SPL have created formal specifications of the system’s features

• Support for cruise control (CC)
Impact analysis

Goal: assess impact of changes

Scenario:

• To identify bugs, stakeholders in our SPL have created formal specifications of the system’s features

• Support for cruise control (CC)
Impact analysis

Stability-control behaviour property: *No subsystem increases acceleration when SC is engaged*
Impact analysis

Stability-control behaviour property: *No subsystem increases acceleration when SC is engaged*

1. CC cruise speed is set
2. Engage CC
3. Drivers loses control
4. Engage SC
5. CC accelerates to achieve cruise speed

Adding CC violates the given property (Impact analysis aims to detect that promptly)
Impact analysis

Stability-control behaviour property: *No subsystem increases acceleration when SC is engaged*

1. CC cruise speed is set
2. Engage CC
3. Drivers lose control
4. Engage SC
5. CC accelerates to achieve cruise speed

Adding CC violates the given property
Impact analysis

Stability-control behaviour property: *No subsystem increases acceleration when SC is engaged*

1. CC cruise speed is set
2. Engage CC
3. Drivers loses control
4. Engage SC
5. CC accelerates to achieve cruise speed

Adding CC violates the given property
(Impact analysis aims to detect that promptly)
Impact analysis (Research questions)

- Currently, consistency between implementation assets (code) and the system's specified property is mostly intractable.
Impact analysis (Research questions)

- Currently, consistency between implementation assets (code) and the system’s specified property is mostly intractable.

RQ: How to verify that the system implementation does not break its specified properties?
Analyses
(Architectural analysis)
Architectural analysis

• Feature model = view of the system architecture
• From the recovered traces, one can track the “health of the system”
• Different indicators can be collected to assess the system evolution:
  ◦ code metrics
  ◦ process metrics
  ◦ feature-based metrics
  ◦ feature-model based metrics
  ◦ product-line based metrics
Architectural analysis

- Evidence relating scattering and defects is rather preliminary.
Architectural analysis

- Evidence relating scattering and defects is rather preliminary.

**RQ:** Can we provide more evidence for the relationship between scattering and defects?
Recommendations
Recommendations + research question

Suggestions for:
Recommendations + research question

Suggestions for:

• Consistency analysis
Recommendations + research question

Suggestions for:

- Consistency analysis
- Impact analysis
Recommendations + research question

Suggestions for:

• Consistency analysis

• Impact analysis

• Architectural analysis
Recommendations + research question

Consistency:

- Fix recommendations for different artifacts types

RQ: How to devise a fixing recommender integrating different artifacts, with different abstraction levels?

RQ: Which feature-based metrics are good defect predictors?
Recommendations + research question

Consistency:

- Fix recommendations for different artifacts types

RQ: How to devise a fixing recommender integrating different artifacts, with different abstraction levels?
Recommendations + research question

Consistency:

• Fix recommendations for different artifacts types

  RQ: How to devise a fixing recommender integrating different artifacts, with different abstraction levels?

Impact analysis:

• Point which features are more likely to have defects after a change
Recommendations + research question

Consistency:

• Fix recommendations for different artifacts types

   **RQ:** How to devise a fixing recommender integrating different artifacts, with different abstraction levels?

Impact analysis:

• Point which features are more likely to have defects after a change

   **RQ:** Which feature-based metrics are good defect predictors?
Recommendations + research question

Architectural analysis:
Recommendations + research question

Architectural analysis:

• Propose merges (features are too similar)
Recommendations + research question

Architectural analysis:

- Propose merges (features are too similar)
- Suggest feature retirement
Recommendations + research question

Architectural analysis:

• Propose merges (features are too similar)

• Suggest feature retirement

• Suggest which features to modularize
Recommendations + research question

Architectural analysis:

• Propose merges (features are too similar)

• Suggest feature retirement

• Suggest which features to modularize

RQ: Which scenarios should be supported (are required in practice)?
Conclusion

- We hypothesized that feature-oriented evolution can mitigate existing challenges in evolving large-complex systems.

- From that hypothesis, we presented our vision based on tracing, analyses and recommendations.

- We are have started working on the realization of that vision.
Thanks for listening!