JOHANNES KEPLER UNIVERSITÄT LINZ

software competence center hagenberg

Configurable Systems & Testing Survey

Glossary:

This part summarizes some terms used in the survey. Please do not hesitate to ask questions to further clarify any term or question.

- **Variability**: The ability of a software system or artifact to be efficiently extended, modified, customized, or configured.
- **Configurable system**: typically consists of configurable artifacts (e.g., code, models, requirements) that use configuration options and a specification of configuration options.
- **Variability-intensive systems**: can be any *Configurable system*, with a large degree of *variability* / configurability.
- **Software Product Lines**: are a form of configurable systems. They are families of related software variants, where each variant represents a configuration.
- **Software Product Families**: are synonymous with *Software Product Lines*.
- **Configuration option**: represents the configurability of a specific aspect of the system (might also be called feature, setting, decision, ...). For example the type of sound system in a car, or the number of loudspeaker in the car.
- **Configuration**: is a specific valid set of *configuration options* with specific values assigned.
- **Variability model**: is typically used as a configuration specification of *configuration options* and their dependencies. Helps to reason over the specification (e.g., number of possible configurations) or to assure the quality of the specification (e.g., maintainability or absence of anomalies).
- **Feature model**: is a type of variability model, which captures *configuration options* (a.k.a. features) and the relationships among them, typically in a tree structure.
- **Decision model**: is a type of variability model, which represent *configuration options* as decisions with a range of values and dependencies between them. Can be represented in a table or in other textual notation.
- **Clone-and-Own**: refers to an ad-hoc process of manually cloning code and adapting it to implement different *configurations*.
- **Object-orientation**: Object-oriented programming is a programming paradigm based on the concept of "objects" as representation of abstract or real-world objects (e.g., car), together properties (e.g., color) and procedures (e.g., start()). Properties and procedures from one object can be "inherited" by other objects.
- **Conditional compilation**: Provides methods so the generated executable program includes only a subset of the different code sections or libraries of the developed program (e.g., C '*#ifdef*' preprocessor annotations). This can be used to generate software for different configurations.
- **Conditional execution**: The program controls what parts are executed (e.g., via if-Statements). This can be used to configure software, by only executing program parts that are relevant for a *configuration*.
- **Platform team**: leads decisions about *variability* of the system, which can affect many to all components. For instance, domain-engineering decisions such as what should be configurable and with which options are in the responsibility of such a team.

Step 1. Variability management

This part contains questions about your general handling of variability.

1.1. How is the software of your system configured (variability mechanisms)?	
A: *Conditional compilation* (e.g., #IFDEFs)	B: *Conditional execution* (e.g., If-Statements)
C: Modularization (e.g., Build system)	D: *Clone-and-Own*
E: *Object-oriented* mechanisms (polymorphism, inheritance, etc.)	F: Configuration parameters (files) (e.g., properties)

1.2. Which challenges/difficulties do you find in using *variability* mechanisms?	
A: Identifying the products affected by a change	B: Implementing the same change in several variants
C: Ensuring their maintenance	D: Generating new variants of similar products
E: Other:	

1.3. How are you currently specifying (documenting) *variability*?	
A: Simple list of *options* in a unstructured document	B: Structured list of *options* / variants in spreadsheets (e.g., Excel)
C: In source code (e.g., via comments)	D: *Configuration* tool with constraints/dependencies
E: *Variability model* (such as *feature model*, *decision model*)	

1.4. Who develops and maintains the *variability* information/documentation (see 1.3)?	
A: A dedicated *platform team*	B: The software / system development department
C: Other (e.g. sales, requirements departments, business management) Please Elaborate:	

1.5. In case, you do not use a variability model: Why do you not use a variability model?
A: We did not know about them
B: The model types we are aware of are not powerful enough to represent our configurability
C: It was/is not necessary to create a *variability model*, in our opinion. Why?:
D: We would like to introduce a *variability model* in the future
E: Other:

1.6. Do you use any tools to create and work with *variability models* and/or create product configurations? ***A tool in this context could be any software that automates something in the management of variability or the configuration of the system (e.g. configurator tool, DSL, ...).***	
◯ A: Yes. Which?:	◯ B: No. Why?:

1.7. Which of the following terms were you aware of before this session?	
A: *Software Product Lines*	B: *Software Product Families*
C: *Variability-intensive systems*	D: *Variability Models*
E: *Feature Models*	F: *Decision Models*

1.8. What support are you missing from existing variability management tools? What support for variability management would you need?

Step 2. Test Reuse

This part contains questions about testing of your configurable system.

2.1. Do you test different configurations?	
◯ A: Yes	◯ B: No

2.2. How do you design tests for different configurations?
A: We do not design tests specific for *configurations*, they work the same regardless of *configuration*
B: We design tests to run on a specific *configuration* and do not reuse tests for other *configurations*
C: We reuse test scenarios, but adapt the tests to work on different *configurations* (*Clone-and-Own*)
D: We develop tests configurable, so they are automatically adapted to the *configuration* How?:

2.3. How are system *configurations* set up for testing?
◯ A: The system can be automatically configured by software
◯ B: The system requires manual configuration
◯ C: Software can be configured automatically, but requires manual hardware setup

2.4. How are tests selected for different configurations?
A: All tests are executed on all tested *configurations*
B: Test scenarios are executed on *configurations* that are known to influence the test scenario
C: Some base functionality tests are executed on all tested *configurations*
D: Tests are selected automatically based on coverage and changes over *configurations*

Step 3. Context

This part contains questions about your experience and the domain of your configurable system.

3.1. What is the domain of your configurable system? (e.g. automotive, telecommunication, medical)

3.2. What have been your roles in the development of highly configurable systems?		
Developer	Modeler	Team leader
Project manager	Domain expert	Researcher
Product manager	Marketing expert	Product owner
System owner	System architect	Software architect
Other:		

3.3. How many years of industrial experience do you have with highly configurable systems?		
○ <1 year	○ 1-2 years	○ 3-5 years
○ 5-10 years	○ 11-20 years	○ >20 years