<table>
<thead>
<tr>
<th>Éligible à une allocation de type :</th>
<th>UCA-EDSTIC-EUR-DS4H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titre du sujet :</td>
<td>Identifying, Visualizing, and Managing Variability Implementations in Very Large Variability-rich Software Systems</td>
</tr>
<tr>
<td>Mention de thèse :</td>
<td>Informatique</td>
</tr>
<tr>
<td>HDR Directeur de thèse inscrit à l'ED STIC :</td>
<td>Philippe Collet</td>
</tr>
</tbody>
</table>

**Co-encadrant de thèse éventuel :**

<table>
<thead>
<tr>
<th>Nom :</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prénom :</td>
<td></td>
</tr>
<tr>
<td>Email :</td>
<td></td>
</tr>
<tr>
<td>Téléphone :</td>
<td></td>
</tr>
</tbody>
</table>

**Email de contact pour ce sujet :** philippe.collet@univ-cotedazur.fr

**Laboratoire d’accueil :** I3S

**Description du sujet :**

**Context & Challenges**

Many industry sectors are facing an increasing demand to develop customized or individualized software-intensive systems. They must then manage the variability of their systems. Variability is the property of a software system to be changed or customized with regards to some specific needs of its surrounding environments (users, customers, hardware platforms, etc.). This variability can be managed by heavyweight software product lines [1], external configuration systems, or ad hoc approaches, e.g. cloning and adapting the main software project for each new customer need [2]. But in the majority of real cases, this variability is managed in the code through a wide variety of implementation mechanisms, e.g., macros, parameters in classes or methods, attributes, inheritance, or design patterns, all defining variant points in the code.
Knowing the exact location of these variation points is crucial to manage variability, e.g., to reason on it statically to check consistency with other models [4,5], to explore or optimize a design space representing a large set of variants (from thousands to billions potential variants) [6]. In this context, identifying the variability inside large existing code bases is very complex, or just not feasible, for developers, while state-of-the-art analysis techniques based on similarity detection or annotations do not scale [2,3].

This PhD thesis will thus address the four following scientific locks:

- How to precisely identify variability implementation techniques in large code bases?
- How to provide adapted and scalable visualization means to help software developers and architects to browse and exploit the identified points of interest wrt variability?
- How to check in a scalable way the identified variation points with other high-level domain models such as feature models?
- How to handle evolving code over time wrt the three previous locks (identification, visualization, consistency)?

Approach and expected results

To tackle these challenges, the PhD work will first rely on original approaches to identify the implementation techniques, such as some properties of the Alexander's theory of centers [7] (local symmetries [8]), and some quality and variability-specific metrics [9]. Visualization [10] and consistency checking [4,5,6] techniques will be devised and validated incrementally, relying on experiments and applications to very large open-source software systems (code bases of largest projects from the Apache consortium and from the Mozilla foundation are expected targets, with hundreds of thousands to millions lines of code per project). Once the first identification operators being partially validated on several projects, their definition will be extended to cope with the time dimension. The definition, prototype tool for identification, and visualizations are then expected to be incrementally augmented and validated on relevant sets of very large code bases.

The fundamental outcomes of the PhD will be (i) the definition of a well-grounded set of identification operators (with a formal model) for the considered variability implementation techniques, and their evolution over time, (ii) algorithms to enable the usage of these operators in a scalable way over very large code bases, (iii), complementary visualization techniques adapted to the nature and volume of the data. These results will be strengthened by the experimental and empirical validations exploiting the prototyped tool to assess the relevance and scalability of the contributions.

References


**English version:**

**Context & Challenges**

Many industry sectors are facing an increasing demand to develop customized or individualized software-intensive systems. They must then manage the variability of their systems. Variability is the property of a software system to be changed or customized with regards to some specific needs of its surrounding environments (users,
customers, hardware platforms, etc.). This variability can be managed by heavyweight software product lines [1], external configuration systems, or ad hoc approaches, e.g., cloning and adapting the main software project for each new customer need [2]. But in the majority of real cases, this variability is managed in the code through a wide variety of implementation mechanisms, e.g., macros, parameters in classes or methods, attributes, inheritance, or design patterns, all defining variant points in the code assets [3].

Knowing the exact location of these variation points is crucial to manage variability, e.g., to reason on it statically to check consistency with other models [4,5], to explore or optimize a design space representing a large set of variants (from thousands to billions potential variants) [6]. In this context, identifying the variability inside large existing code bases is very complex, or just not feasible, for developers, while state-of-the-art analysis techniques based on similarity detection or annotations do not scale [2,3].

This PhD thesis will thus address the four following scientific locks:
- How to precisely identify variability implementation techniques in large code bases?
- How to provide adapted and scalable visualization means to help software developers and architects to browse and exploit the identified points of interest wrt variability?
- How to check in a scalable way the identified variation points with other high-level domain models such as feature models?
- How to handle evolving code over time wrt the three previous locks (identification, visualization, consistency)?

Approach and expected results

To tackle these challenges, the PhD work will first rely on original approaches to identify the implementation techniques, such as some properties of the Alexander's theory of centers [7] (local symmetries [8]), and some quality and variability-specific metrics [9]. Visualization [10] and consistency checking [4,5,6] techniques will be devised and validated incrementally, relying on experiments and applications to very large open-source software systems (code bases of largest projects from the Apache consortium and from the Mozilla foundation are expected targets, with hundreds of thousands to millions lines of code per project). Once the first identification operators being partially validated on several projects, their definition will be extended to cope with the time dimension. The definition, prototype tool for identification, and visualizations are then expected to be incrementally augmented and validated on relevant sets of very large code bases.

The fundamental outcomes of the PhD will be (i) the definition of a well-grounded set of identification operators (with a formal model) for the considered variability implementation techniques, and their evolution over time, (ii) algorithms to enable the usage of these operators in a scalable way over very large code bases, (iii) complementary visualization techniques adapted to the nature and volume of the data. These results will be strengthened by the experimental and empirical validations exploiting the prototyped tool to assess the relevance and scalability of the contributions.
References


