



HAL
open science

Generative AI for Reengineering Variants into Software Product Lines: An Experience Report

Mathieu Acher, Jabier Martinez

► To cite this version:

Mathieu Acher, Jabier Martinez. Generative AI for Reengineering Variants into Software Product Lines: An Experience Report. SPLC 2023 - 27th ACM International Systems and Software Product Lines Conference, Aug 2023, Tokyo, Japan. pp.1-9, 10.1145/3579028.3609016 . hal-04160693

HAL Id: hal-04160693

<https://inria.hal.science/hal-04160693v1>

Submitted on 12 Jul 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Generative AI for Reengineering Variants into Software Product Lines: An Experience Report

Mathieu Acher

Univ Rennes, IRISA, Inria, CNRS, IUF
Rennes, France
mathieu.acher@irisa.fr

Jabier Martinez

Tecnalia, Basque Research and
Technology Alliance (BRTA)
Pessac, France
jabier.martinez@tecnalia.com

ABSTRACT

The migration and reengineering of existing variants into a software product line (SPL) is an error-prone and time-consuming activity. Many extractive approaches have been proposed, spanning different activities from feature identification and naming to the synthesis of reusable artefacts. In this paper, we explore how large language model (LLM)-based assistants can support domain analysts and developers. We revisit four illustrative cases of the literature where the challenge is to migrate variants written in different formalism (UML class diagrams, Java, GraphML, statecharts). We systematically report on our experience with ChatGPT-4, describing our strategy to prompt LLMs and documenting positive aspects but also failures. We compare the use of LLMs with state-of-the-art approach, BUT4Reuse. While LLMs offer potential in assisting domain analysts and developers in transitioning software variants into SPLs, their intrinsic stochastic nature and restricted ability to manage large variants or complex structures necessitate a semi-automatic approach, complete with careful review, to counteract inaccuracies.

ACM Reference Format:

Mathieu Acher and Jabier Martinez. 2023. Generative AI for Reengineering Variants into Software Product Lines: An Experience Report. In *27th ACM International Systems and Software Product Line Conference - Volume B (SPLC '23)*, August 28-September 1, 2023, Tokyo, Japan. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3579028.3609016>

1 INTRODUCTION

Reengineering variants into a Software Product Line (SPL) is a complex problem [40]. Several industrial cases have been reported in the scientific literature [41] making evident that it is a global and challenging issue for companies seeking to embrace systematic reuse and an overall increase in the quality of their product families.

Numerous automatic techniques have been proposed to analyze the variants, identify the common and variable parts, and linking these parts to features that can be leveraged by an SPL platform [40]. This is related to the n-way merging problem with its compare, match, and compose steps [54]. There are also frameworks which objective is to integrate and seamlessly chain different techniques and to allow the analysis of variants implemented with different programming or modelling languages [44].

SPLC '23, August 28-September 1, 2023, Tokyo, Japan

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *27th ACM International Systems and Software Product Line Conference - Volume B (SPLC '23)*, August 28-September 1, 2023, Tokyo, Japan, <https://doi.org/10.1145/3579028.3609016>.

Generative AI approaches can be considered a disruptive technology that is questioning how different software engineering tasks are being performed so far. It is important to understand the advantages and limitations to both leverage them and avoiding incorrect expectations. The question that motivates this preliminary analysis is to understand to which extent generative AI technologies can help in the tasks of reengineering variants into SPLs. We defend the idea that *large language models (LLMs)* can be leveraged to support the migration of variants into SPLs.

LLMs are capable of tackling problems from linguistics, maths, or commonsense reasoning. BERT [23], GPT-3 [13], PaLM [18], to name a few, are scaling to support a variety of tasks such as text generation, question-answering, or text classification [24, 28, 33, 57]. In software engineering, code assistants based on LLMs have been proposed like Alphacode, CodeParrot or Codex [7, 12, 17, 19, 37, 38] and are now deployed at scale for supporting programmers, such as GitHub Copilot [29]. Based on *prompts* (see green boxes hereafter), composed of an instruction written in natural language that may include existing code, LLMs produce programs and possibly explanations (red boxes hereafter).

Our contribution is to show how LLMs can be concretely and originally used for synthesizing an SPL out of variants. We mainly used and report on sessions with ChatGPT-4 [51], using the Web chat interface of OpenAI. To the best of our knowledge, generative AI and LLMs have not been considered in this context. Machine learning has been applied in SPL and variability engineering, with different use cases [3, 30, 32, 34, 35, 48, 49, 56, 59–64]. The use of natural language processing (NLP) within SPL engineering has caught attention, e.g., for domain analysis and requirements engineering [2, 10, 14–16, 22, 27, 31, 47, 50, 52, 53], but not through LLMs. An exception is a recent paper accepted at SPLC 2023 [1] about *programming* variability with LLM. However, the problem of synthesizing an SPL out of variants is not considered. Our objective is also to call the SPL community to start the discussion and conduct more experiments on this topic. We publicly release the material used for the experience report in this repository: <https://github.com/acherm/variantsGPT/>

2 BACKGROUND

Reengineering variants into an SPL: Clone-and-own is a reuse practice that consists of taking advantage of a similar existing product to adapt it to new requirements through the addition or removal of features. This ad-hoc reuse is accepted in industrial settings because it is simple, available and independent of the assigned developers [25]. However, in the long term, this practice to create variants is not appropriate as the high number of variants and features makes maintenance and management of the whole

product family very difficult. At this point, an SPL approach, in which features are explicitly defined to represent the variability and commonalities between assets needs to be adopted. According to an industrial survey [11], more than fifty percent of the companies that adopt an SPL start from a set of existing variants previously created through ad-hoc reuse techniques.

The process where existing variants are leveraged to adopt an SPL is known as extractive SPL adoption [36]. One of the first steps consists in identifying and locating the features spread among the variants. If the features are not completely known in advance, feature identification and naming must be performed. Feature location techniques rely on heuristics that are sensitive to multiple factors, particularly the quantity and diversity of existing variants [42, 45], as well as the presence of changes not propagated to the rest of variants [26]. These factors directly impact the effectiveness and metrics such as accuracy, precision, and recall. After that, in many cases, the aim is to create a “single-copy” representation of the variants, i.e., a configurable code base containing all the possible features with feature annotations. This maximal single-copy system is also known as the 150% representation, and it is used during the SPL derivation process to automatically create variants through the removal of the undesired features. N-way merging of variants to create 150% representations is a common practice to adopt an SPL using annotative approaches [55].

In order to establish a variability model, it is necessary to undertake the process of discovering feature constraints and then synthesize an understandable abstraction for domain experts. This analysis phase of the reengineering process [4] is very relevant, but in this work we focus on the detection and transformation phases.

Tool support with BUT4Reuse: Bottom-Up Technologies for Reuse (BUT4Reuse) [44] is a framework to support users during the technical activities of extractive SPL adoption such as feature identification, naming, and location, and the construction of reusable assets for SPL implementation (e.g., creation of 150% representations). Its main advantages are its capacity to be generic and extensible, the significant set of available extensions, and a level of abstraction that allows to seamlessly chain different activities compared to tools that exclusively focus on one of them and most of the time with a unique technique. The framework is generic in the sense that it supports different programming and modelling languages through the concept of Adapters. There are Adapters already available for widely-used languages such as Java, C++, EMF-based models including UML, languages to describe graphs, and new Adapters can be integrated to support the types of artefacts of interest for each case. BUT4Reuse is extensible in the sense that different techniques can be integrated for all the activities, from feature identification, naming, and location, to feature constraints discovery, feature model synthesis, visualization etc. The list and description of the 17 currently available adapters and 23 extensions is available [44].

3 RESEARCH METHODOLOGY

In the remainder of the paper, we consider five cases, involving different variants, kinds of artifacts, and reengineering scenarios.

For each case, we report on sessions with ChatGPT-4¹ [51]. We used the Web chat interface of OpenAI, not the API. The hyperparameters such as temperature cannot be controlled in this setting, and we had to rely on default values.

We started a new chat for each session and case. We have also repeated some sessions for the same case with the same prompt or slight variations of the prompts. The intent was to investigate the robustness of LLM or a potential missing opportunity. In any case, we restarted a new, fresh session between each repetition of a case or between each case. The prompts, written in a green box with ‘(User)’ hereafter, are reported in the different sections and there was no additional context. The interested reader can find more details about sessions in the repository <https://github.com/acherm/variantsGPT/>.

The session per case involves different interactions and prompts until the presented results were judged satisfactory. Within a session, we, as users, asked for different tasks such as:

- domain analysis, summary of commonalities and differences, as plain languages or as tables, identification of features
- synthesis of an integrated, 150% model or code (and possibly a visualization)
- synthesis of a template-based generator
- refactoring of models or code
- corrections of code, model, or explanations that were perceived as inaccurate or simply unusable.

Hence, sessions were highly interactive. The first author of the paper wrote the prompts and made the interactive sessions. The two authors have reviewed and assessed the different sessions, throughout and after the experience.

4 CASE 1 AND 2: BANKING SYSTEMS VARIANTS IN JAVA AND UML

We start with Case 1 and 2. Our first challenge involves three variants for different banking systems used in previous works [5, 6, 65, 66]. In this scenario, we have created three model variants through ad-hoc reuse with variations on the limit of bank withdrawal, the consortium entity and currency conversion. The challenge is to factor out the different variants and re-engineer them as an SPL. We consider two cases: (1) the first case involves variants implemented in Java; (2) the second case involved variants implemented in UML class diagrams. Figure 1 shows a visual representation of the UML class diagrams, using PlantUML notation. Listing 1 provides an excerpt of the Java code corresponding to the first variant of Figure 1a.

4.1 Case 1: Java variants

In this first setting, we have converted the three UML variants depicted in Figure 1 as Java code. The implementation resides in a unique file with the different classes, fields, and methods. Methods returning void have an empty body, while methods returning a double have a basic default implementation, the same for all variants. The overall challenge is to build an SPL out of the 3 Java variants.

(User) I have three variants of a Java implementation of a banking system. The first one is:

¹In the rest, we use ChatGPT-4, ChatGPT and LLM alternately, but the three terms refer to the same thing for this paper.

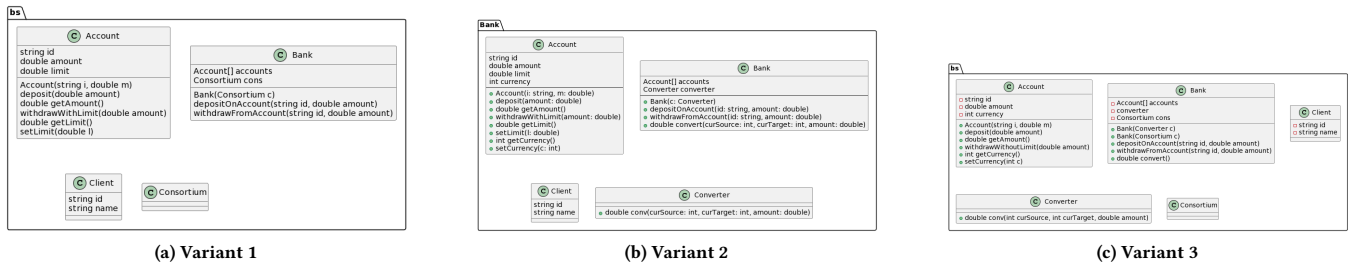


Figure 1: Variants of Class diagrams

```
import java.util.List;

public class Variant1 {

    class Account {
        private String id;
        private double amount;
        private double limit;

        public void deposit(double amount) {}
        ...
    }

    class Client {
        private String id;
        private String name;
    }

    class Consortium {
        // Empty class as per the description.
    }
}
```

Listing 1: Java (first variant, corresponding to Figure 1a)

(User) The second variant is: ... The the third variant is: ... Considering the three variants, can you summarize: (1) what’s common (2) what varies?

(LLM) Sure, I can summarize the common elements and the differences between the three Java variants:
Common Elements
 All variants have Account, Bank, and Client classes. Each Account has a String id and a double amount. The Account class in each variant has deposit, getAmount methods. Each Bank class has methods for depositOnAccount and withdrawFromAccount. Each Client class has a String id and String name.
Differences ... Variant3: The Account class has an int currency, getCurrency, and setCurrency methods similar to Variant2. But it introduces withdrawWithoutLimit instead of withdrawWithLimit. The Bank class has a List of Account objects, a Converter, a Consortium, and a convert method. This is a mix of features from both Variant1 and Variant2. The Consortium class returns, but it’s now along with the Converter class from Variant2.

In general, the response is quite precise, but it should not be assumed as unquestionable. In particular, there are ambiguous explanations (e.g., there is no apparent link between Consortium and

Converter). We continue by asking to build an annotated SPL with templating directives.

(User) Can you use a templating engine (like jinja2) that would allow deriving the three original variants? The principle is to first create a template with all classes, fields, methods and then annotate variant-specific elements through templating expressions and directives

```
import java.util.List;

public class Variant {

    class Account {
        private String id;
        private double amount;

        {% if variant != 3 %}
        private double limit;
        {% endif %}

        {% if variant != 1 %}
        private int currency;
        {% endif %}

        public void deposit(double amount) {}

        public double getAmount() {
            return 0.0;
        }

        {% if variant == 3 %}
        public void withdrawWithoutLimit(double amount) {}
        {% else %}
        public void withdrawWithLimit(double amount) {}
        {% endif %}

        {% if variant != 3 %}
        public double getLimit() {
            return 0.0;
        }

        public void setLimit(double limit) {}
        {% endif %}

        {% if variant != 1 %}
        public int getCurrency() {
            return 0;
        }

        public void setCurrency(int currency) {}
        {% endif %}
    }

    class Bank {
        {% if variant != 2 %}
        private List<Account> accounts;
        {% else %}
        private Account accounts;
        {% endif %}

        {% if variant != 1 %}
        private Converter converter;
        {% endif %}

        {% if variant != 2 %}

```

```

private Consortium cons;
{% endif %}

public void depositOnAccount(String id, double amount) {}

public void withdrawFromAccount(String id, double amount) {}

{% if variant != 1 %}
public double convert(int curSource, int curTarget, double amount) {
    return 0.0;
}
{% endif %}
}

class Client {
    private String id;
    private String name;
}

{% if variant != 1 %}
class Converter {
    public double conv(int curSource, int curTarget, double amount) {
        return 0.0;
    }
}
{% endif %}

{% if variant != 2 %}
class Consortium {
    // Empty class as per the description.
}
{% endif %}
}
    
```

Listing 2: Java (template-based SPL)

The template is correct and variant-specific elements are correctly annotated. We move forward, and challenge the LLM-based assistant to refactor the SPL (and template), this time with a feature-oriented perspective.

(User) Instead of having "variant" as a variable to configure the template, I'd like to have other variables that represent "features" of the banking system... can you propose some features?

```

import java.util.List;

public class Variant {

    class Account {
        private String id;
        private double amount;

        {% if hasWithdrawalLimit %}
        private double limit;
        {% endif %}

        {% if supportsCurrency %}
        private int currency;
        {% endif %}

        public void deposit(double amount) {}

        public double getAmount() {
            return 0.0;
        }

        {% if hasWithdrawalLimit %}
        public void withdrawWithLimit(double amount) {}
        {% else %}
        public void withdrawWithoutLimit(double amount) {}
        {% endif %}

        {% if hasWithdrawalLimit %}
        public double getLimit() {
            return 0.0;
        }
        }

        public void setLimit(double limit) {}
        {% endif %}

        {% if supportsCurrency %}
        public int getCurrency() {
            return 0;
        }
        }
    }
}
    
```

```

public void setCurrency(int currency) {}
{% endif %}
}

class Bank {
    {% if isSingleAccount %}
    private Account accounts;
    {% else %}
    private List<Account> accounts;
    {% endif %}

    {% if supportsCurrency %}
    private Converter converter;
    {% endif %}

    {% if hasConsortium %}
    private Consortium cons;
    {% endif %}

    public void depositOnAccount(String id, double amount) {}

    public void withdrawFromAccount(String id, double amount) {}

    {% if supportsCurrency %}
    public double convert(int curSource, int curTarget, double amount) {
        return 0.0;
    }
    {% endif %}
}

class Client {
    private String id;
    private String name;
}

{% if supportsCurrency %}
class Converter {
    public double conv(int curSource, int curTarget, double amount) {
        return 0.0;
    }
}
{% endif %}

{% if hasConsortium %}
class Consortium {
    // Empty class as per the description.
}
{% endif %}
}
    
```

Listing 3: Improved template-based SPL with created features

The new template allows one to configure variants in terms of 4 features (hasWithdrawalLimit, hasConsortium, supportsCurrency, isSingleAccount). The LLM-based assistant can provide a synthesis of the situation, with a product comparison matrix.

(User) Can you create a table that describes whether feature variables are supported by Variant1, Variant2, and Variant3?

(LLM)

Feature	Variant1	Variant2	Variant3
hasWithdrawalLimit	Yes	Yes	No
supportsCurrency	No	Yes	Yes
isSingleAccount	No	Yes	No
hasConsortium	Yes	No	Yes

Table 1: Variants' and features' matrix

Deriving variants out of SPL: Out of the template of Listing 2 or Listing 3, we can successfully derive and retrieve the original three variants. The identification of features (see Listing 3) allows one to envision to derive new variants, beyond the three original ones. Stated differently, Table 1 could be expanded through new combinations of features and thus variants. Obviously, not all combinations of features are syntactically valid or would make sense from a domain point of view. Hence, we consider it is the responsibility of developers to build a feature model in order to properly

scope the feature configuration space of the SPL. We did not use generative AI in this context. We leave as future work the problem of automating the synthesis of feature model with an assistant.

Syntactical and semantics errors: Throughout our interactions, we were exposed with two kinds of errors.

Misleading variants. The correct template of Listing 2 has been obtained after several iterations and fixes. Indeed, there was a subtle yet severe error during the session with the assistant. The templating code was unsound at some point and needs to be fixed. Specifically, the field limit was not included in Variant2 since the conditional was wrong. It requires to further interact as follows.

(User) I don't understand why:

```
{% if variant != 2 %}
private double limit;
{% endif %}
```

since the field limit is included in Variant2.

The fix was correctly provided after some interactions. This kind of error not only increases the cognitive load and effort. There is also the risk that the error goes unnoticed, thereby misleadingly authorizing unsafe variants that lack certain Java fields.

Syntactical error affecting the whole SPL. In Listing 3, the careful reader can notice that the method `getAmount` has no ending bracket. It is a real bug that causes compilation errors for all variants. The error has been fixed, and a new template has been proposed. However, it requires developers to locate the error (after the observation of compilation failures) and eventually interact again with the assistant.

Summary: The experience for Case 1 is positive, since an annotated SPL can be generated and out of which the original Java variants can be derived. It is also possible to identify relevant and meaningful features that did not exist in the first place, and then envision to expand the set of variants. On a negative side, some subtle errors can be introduced and SPL developers should carefully review the generated code.

4.2 Case 2: UML Class diagrams variants

In this second setting, we consider the three UML variants depicted in Figure 1. The case is similar to the previous one: there are 3 variants representing the same exact banking system. However, the technological space is different and challenges generative AI to correctly understand, match, and merge artifacts written in a different dialect, arguably less popular than Java. In a sense, it is a replication of the case described in Section 4.1.

The original class diagrams are encoded in a technical, XML-like format (an excerpt is given in the prompts hereafter, see Listing 4). The challenge is to build an SPL out of the 3 UML variants.

We initiate the session with the LLM-based assistant, similarly as with the previous session with Java artefacts.

(User) I'll give you three variants of an UML model... the first one is:

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20110701" xmlns:xmi="http://www.omg.org/spec/XMI/20110701" xmlns:uml="http://www.eclipse.org/uml2/4.0.0/uml" xmi:id="_HJ0ewHxvEeOpP-PjhVelzQ" name="Bank">
  <packagedElement xmi:type="uml:Package" xmi:id="_lZw_cHyAEe0S740IN395aA" name="bs">
```

```
<packagedElement xmi:type="uml:Class" xmi:id="_qv-PIHyAEe0S740IN395aA" name="Account">
  <ownedAttribute xmi:id="_xI3YHyAEe0S740IN395aA" name="id" type="__EESKn-OEe047f7pJZDqVw"/>
  <ownedAttribute xmi:id="_D1m20HyBEe0S740IN395aA" name="amount" type="__EESJH-OEe041f7pJZDqVw"/>
  <ownedAttribute xmi:id="_ZPoIYHyBEe0S740IN395aA" name="limit" type="__EESJH-OEe041f7pJZDqVw"/>
  <ownedOperation xmi:id="_idK2AHyBEe0S740IN395aA" name="Account">
    <ownedParameter xmi:id="_lkf1UH-PEe038f7pJZDqVw" name="i" type="__EESKn-OEe047f7pJZDqVw"/>
    <ownedParameter xmi:id="_m38C0H-PEe038f7pJZDqVw" name="m" type="__EESJH-OEe041f7pJZDqVw"/>
  </ownedOperation>
  ...
</uml:Model>
```

Listing 4: UML class diagram of the first variant (excerpt, corresponding to Figure 1a)

(LLM) Based on the provided UML model, we have a banking system with four main classes: Account, Bank, Client, and Consortium. Below is a summary of these classes and their attributes and operations:

```
1. Account Class
- Attributes:
  - id (string)
  - amount (double)
  - limit (double)
...
```

Comments of the assistant show a good syntactical understanding of the UML model.

(User) Here is the second UML model: ... here is the third UML model variant: ... considering the three UML variants, can you summarize: (1) what's common (2) what varies?

(LLM) After analyzing the three UML model variants, here is a summary of what's common and what varies among them: ... All three variants have the following classes: - Account - Bank - Client - Converter These classes share some common attributes and operations: ... The main differences among the three UML model variants are as follows: Variant 1: - The Account class has an additional operation: withdraw. - The Bank class has an additional operation: addClient. - The Client class has an additional attribute: accounts. ...

Contrary to the previous session with Java, the answer is much less accurate. In fact, there are several errors: (1) not all variants have Converter; (2) the Account class does not have a withdraw operation in any variant; (3) the Bank class does not have a addClient operation in any variant; (4) the Client class does not have accounts operation in any variant.

As with previous Java variants, the lack of understanding spreads when asking to produce an annotated SPL.

(User) Can you create an UML model that can represent the three variants? I'd like to have the result in PlantUML ... can you use a templating engine (like jinja2) to annotate variant-specific elements of the PlantUML model?

(LLM) Certainly! Here's an example of how to use the Jinja2 templating engine to annotate variant-specific elements of the PlantUML model:

```

...
class Client {
- id: string
- name: string
- accounts: Account[*]
}
class Converter {
+ conv(curSource: int, curTarget: int, amount: DOUBLE): DOUBLE
}
{% if variant == 2 or variant == 3 %}
class Consortium {
}
{% endif %}
Account -- Bank: "accounts"
Client -- Bank: "clients"
Bank -- Converter: "converter"
{% if variant == 2 or variant == 3 %}
Bank -- Consortium: "cons"
{% endif %}

```

Listing 5: Template of UML class diagram (excerpt)

First, the field `accounts` should not exist in class `Client`. Second, the conditional is wrong: `Consortium` is included in Variant 1, not in Variant 2. Same error applies for `cons` (see Listing 5).

Lessons and hypothesis. Compared to the case of Java, this session is less supportive of the use of generative AI. We repeated the session with different prompts and strategies for interacting with the LLM-based assistant, and came to a similar conclusion with negative experiences. We report on following limits:

- inaccurate analysis of commonality and differences: the previous excerpt of the explanations about commonalities and differences among the three variants exhibits several errors. When asking a comparison table, we noticed similar trends: some elements are considered as common whereas it is not the case; some elements are wrongly considered as specific; some elements are simply made up (new operations, but also new classes like `Manager` that simply do not exist).
- inaccurate integrated model: we asked to build an integrated, 150% model *without annotation*, and some elements were made up or repeated, leading to an inaccurate model.
- inaccurate synthesis of template with wrong conditionals, invented elements, or unnecessary repetition of elements.
- inaccurate visualization: we also asked to produce a 150% model with colors in PlantUML to showcase specific elements. The resulting visualization was incomplete and wrong.

Obviously, the four limitations are related, i.e., a lack of understanding of what is common and varies has a ripple effect on the integrated model, possible template and visualization. Overall, the case of UML variants was less favorable than the case of Java variants, despite being on the same domain and banking system. A hypothesis is that the generative AI technology was less mature to handle the UML language than the Java language. Another hypothesis is that UML-like artefacts contain a different structure that challenge the matching and merging of elements. Yet another possible explanation is that the UML models tend to exhibit much more textual information and tokens. It then becomes more problematic to handle for transformers-like generative AI.

Summary: The experience for Case 2 is negative, since some UML elements can be made up while the analysis of commonalities and differences is inaccurate, leading to unsafe SPL. It is in contrast with Case 1 and Java variants, albeit from the same domain.

5 CASE 3: GRAPHML VARIANTS

In this setting, we consider three variants in GraphML format. The case involves a different formalism (capable of representing a state transition system), language, and technological space (GraphML). It has also been used in the context of education, academic presentation, and BUT4Reuse².

We start the session in a very similar fashion.

(User) I'll give you three variants of a graphml model... Here is the first one: here is the second variant: ... here is the third variant: ...

```

<graphml xmlns="http://graphml.graphdrawing.org/xmlns" ... >
<graph id="G" edgedefault="directed">
  <node id="s0">
    <data key="nodeLabel">s0 </data>
  </node>
  <node id="s1">
    <data key="nodeLabel">s1 </data>
  </node>
  <node id="s2">
    <data key="nodeLabel">s2 </data>
  </node>
  <edge source="s0" target="s1">
    <data key="edgeLabel">findVillain </data>
  </edge>
  <edge source="s1" target="s2">
    <data key="edgeLabel">smash </data>
  </edge>
  <edge source="s2" target="s0">
    <data key="edgeLabel">finishHim </data>
  </edge>
</graph>
</graphml>

```

We then continue with the overarching goal of getting a visualization of a 150% SPL that would represent all variants. We employ a step-by-step strategy, with a series of prompt/answer.

(User) Can you create an integrated model that would represent the three variants? I'd like to have the result in graphml as well ... can you translate it to dot format? ... can you add color to edges to nodes that are specific to variant 1, variant 2, or variant 3? Says variant 1 is blue color, variant 2 is green color, variant 3 is red color, and if it's common to all variants, it remains black

The final result is depicted in Figure 2, after a conversion from DOT to PDF. The 150% model is correct, and it is straightforward to engineer an annotated SPL (e.g., with a template-based approach, as previously illustrated). It is also possible to envision expanding the SPL with new variants.

Summary: The experience for Case 3 is positive, since the resulting SPL correctly represents the original variants.

6 CASE 4: STATE CHART VARIANTS

This case involves vending machines' model variants, encoded as state chart and using SCT format. Originally, six variants were

²<https://github.com/but4reuse/but4reuse/wiki/Examples>

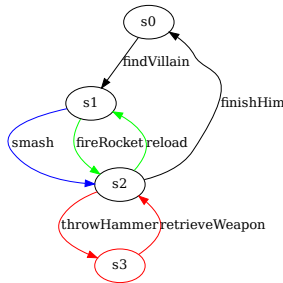


Figure 2: 150% model with colors (out of DOT format)

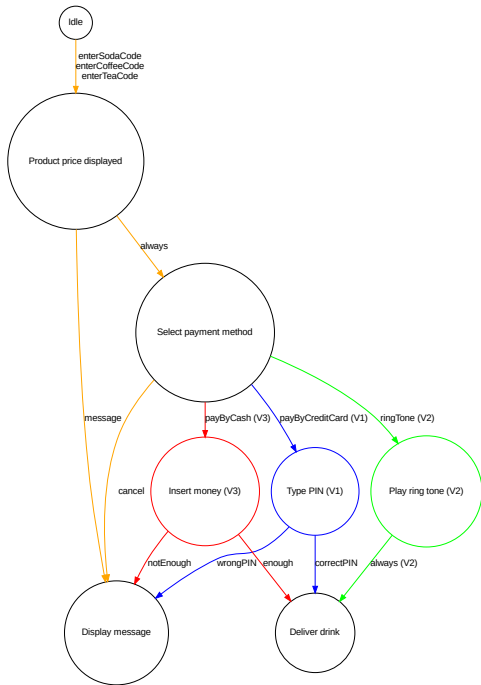


Figure 3: 150% state chart model

used. A first observation and lesson learned is that the size of the context³ (and thus prompts) precludes to consider all variants and information. Furthermore, the SCM format is composed of a state chart and a layout description. We did not consider the layout part to reduce the size of information and tokens fed to LLM. Using ChatGPT-4, we were thus forced to only consider four variants.

We followed a similar methodology as for previous cases. Due to space limitations, we report on key results and observations. Figure 3 shows a 150% model augmented with colors and depicted as PDF. ChatGPT-4 generated the DOT file after some prompts and interactions.

Limitations. There are serious errors in Figure 3. First, ChatGPT-4 takes the liberty of merging edges and transitions (for example, enterSodaCode, enterCoffeeCode, enterTeaCode). Second, some presence conditions are misleading (e.g., always transition between Product price displayed and Select payment method is not specific to a variant). Third, some transitions of the original variants are not

³LLMs are usually trained with relatively small context windows and limited number of tokens [20, 21]. LLMs are typically based on transformers, and the underlying self-attention complexity is quadratic with the length of the input sequence (prompt).

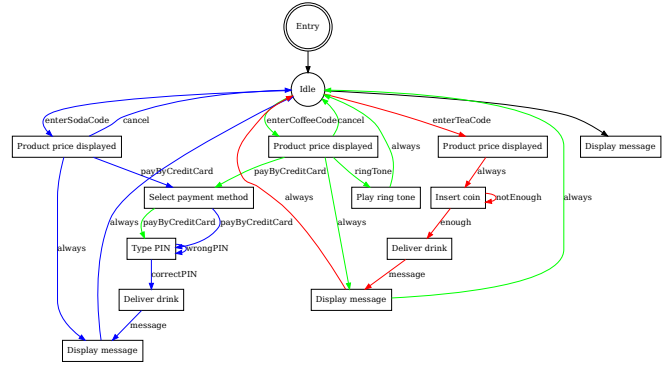


Figure 4: Another 150% state chart model (only 3 variants; replicated session)

included. When repeating our experiments with the same prompts or variations thereof, we observed similar limitations. There were sessions where the 150% model contains duplicated transitions and nodes (states). It leads to more accurate models (e.g., some transitions were this time included). However, the resulting models tend to be hard to maintain; the semantics becomes ambiguous (same transition out of the same named states) and can be problematic to handle by modelling tools. Such an example is given in Figure 4, considering 3 variants instead of 4.

Summary: The experience for Case 4 is negative, since the semantics of the SCM formalism is not properly handled and important information is either lost or wrongly expressed, leading to misleading SPL.

7 CASE 5: PNG VARIANTS

In this setting, we have four images created through the copy-paste-modify reuse paradigm. This case has been considered as part of BUT4Reuse⁴. The images are in Portable Network Graphics (PNG) format. Each PNG file begins with an 8-byte signature and contains a series of chunks, which are binary data blocks. As a result, producing a "diff" among binaries' formats is likely to produce irrelevant information. To avoid directly operating over PNG, BUT4Reuse proposes to use an adapter based on pixels where the similarity between pixels of variants is based on checking that they share the same x y position and a color similarity formula. With current LLMs, we confirm (1) the impossibility to analyze PNG, binary files and compute a relevant diff – there are no magical solutions involved; (2) the provision of pixels location to LLM does not scale: there is simply too much information to fed in; (3) LLM is not helpful to find meaningful features: the information is too low-level and there is no semantic to infer.

Summary: The experience for Case 5 is that LLM cannot be applied "as such" for PNG, since the information is low-level and without an exploitable structure. This experience illustrates the case of binary files and the need of humans providing the desired abstraction level and means for the analysis as in traditional reengineering approaches.

⁴<https://github.com/but4reuse/but4reuse/wiki/Tutorial>

8 DISCUSSION

Threats to validity. The major threat to validity is related to the prompts used in our experiments. Although we believe the used prompts conveyed our intentions, and we attempted to enhance them based on response quality, we cannot be certain that further specifications or clarifications in the prompt would have yielded improved results. We also suspect different LLMs (or LLMs with different hyperparameters like temperature) could lead to different outcomes.

Size of the input variants: Current LLMs cannot handle a large code base or numerous variants, since both can increase the number of tokens supported during inference. It is an open question how LLMs can manage long-term working memory and large context window [20, 21]. Though progress can be envisioned, this is currently a severe limitation for real-world usage.

Variants as binary files or complex structures: As demonstrated in the case of images in PNG format, LLMs were incapable of synthesizing differences and commonalities due to the non-exploitable nature of the information at the binary file level. This limitation stands in contrast to the effectiveness of BUT4Reuse Adapters in transforming input variants into usable structures. The key idea lies in operating at a different level of abstraction, specifically analyzing the commonalities and differences within the pixels themselves.

Consistency among different executions: BUT4Reuse put emphasis on obtaining deterministic results across different executions. With ChatGPT, the results vary depending on the prompt, context, and other factors, so its users should know that partially similar responses could be expected across different executions.

Genericity in terms of variants' programming or modelling language: ChatGPT has been probably trained with almost all existing programming languages where source code was available. While the amount of information will vary depending on if it is a widely used programming or modelling language or a niche language, there is no need to adapt or provide information about the language itself for comparing variants. Despite that the reported Adapters for BUT4Reuse were relatively easy and not very time-consuming to develop (they usually rely on existing parsing libraries or there is already a generic modelling artefacts adapter that can be specialized), it is a clear advantage not having to design and provide an Adapter. When asking ChatGPT about how it calculated the common and variable parts, for instance for the Java variants, we found a similar reasoning as the one that will follow the Adapter decomposing the source code in Classes, Methods, and Fields, and then using set theory to determine if it is common or variable.

Similarity function of the Adapter: The most challenging function of a programming or modelling language Adapter is the similarity between elements. While the Adapter will allow having more control on the similarity function among elements, LLMs hide the complexity of its development and can potentially internally define more complex semantic similarity relations. In fact, one of the most relevant concerns about id-based matching is how easy it fails to refactorings in the source code. When trying some examples from a dataset of equivalent and non-equivalent program pairs (EqBench) [9] we can see that the performance of ChatGPT is not optimal, at least not from its direct response. For instance, if we

provide the Java pair in *pow test Eq* and asks to check if they are equivalent:

(LLM) After comparing the two provided methods, I can confirm that they are not equal. The difference lies in the condition within the second if statement in the newV class:

In the oldV class: $if(y > 8)$

In the newV class: $if(-y < -8)$

The change is in the comparison operators and the negation of y . These alterations will result in different conditions being evaluated in the code.

(User) Are you sure that this alteration is not the same condition?

(LLM) Apologies for the confusion. Upon further review, I can see that the altered condition in the newV class is, in fact, equivalent to the original condition in the oldV class.

This kind of experiments raise questions about the capacity of LLMs to correctly detect source code clones when responding to logical issues that can be more complicated than this example.

Extensible for visualizations: Visualization paradigms in the evolution of variants can be very diverse [46]. BUT4Reuse is extensible for integrating new visualizations, but they need to be implemented and integrated. While the currently available visualizations are specialized for the reengineering activities, with ChatGPT it is possible to ask to manipulate and show the results in different formats such as tables, graphs, and many other approaches, thus potentially focusing on any kind of perspective for the analysis task at hand.

Feature naming: BUT4Reuse integrates the VariClouds visualization for feature naming and to automatically set or suggest feature names [43]. ChatGPT is a powerful tool for summarizing source code so feature naming and creating feature descriptions through LLM could easily outperform previous approaches.

Benchmarking: The state-of-the-art in benchmarking approaches and techniques for the evolution of variants has been analyzed [58] showing significant gaps and evidence that the field could benefit from the design of new benchmarks respecting a minimum of established requirements. The existence of benchmarks are specially useful with the emergence of new techniques to quickly and unambiguously identify the strengths and limitations compared to previous approaches. Besides that, in the case of LLMs we are confronted with the limitations on the size of the information we can provide to ChatGPT while prompting, as most of the benchmarks are real systems of relatively big size.

Variants synchronization and filtering prior to SPL extraction: Fenske et al. [26] used clone detection techniques to try to synchronize variants towards simpler SPL extractions. Other more drastic solutions considered filtering outlier variants to facilitate this process [8, 39]. It will be worth exploring to which extent ChatGPT could help in that task. For instance, we repeated the Java Banking systems experiment providing again the three variants but Variant1 with the class, method, and field names in Spanish instead of English. Apparently the common and variable parts seem to take into account that they are the same even if the name was a translation, but with more difficulties and further incorrect explanations. However, it was possible to ask ChatGPT to translate Variant1 using the

naming conventions in Variant2 and Variant3 while keeping the elements of Variant1. The results were fine except for a small detail (“depositIntoAccount” should have been “depositOnAccount”) that were recognized by ChatGPT as a confusion.

Complementary: Instead of thinking of ChatGPT as a replacement of BUT4Reuse or similar frameworks, the integration of both could be a more beneficial solution for the moment. For instance, the framework can create template-based prompts to get the ChatGPT results for tasks where the value has been proven.

9 CONCLUSION

We showed how LLMs can assist domain analysts and developers in migrating variants as software product lines. The inaccuracies during different steps suggest a semi-automatic approach, where it is mandatory to carefully review the generated code. We identify major barriers: LLMs are by construction stochastic and sensitive to prompt variations and context. The size of the input is also limited: It is difficult for these techniques to handle numerous variants, large code bases, or the underlying structures of binary files.

The directions of further work are multiple. This report is a call to the SPL community to start thinking about the synergies of LLMs and reengineering variants. We believe that the research direction that could have the most practical impact is the integration of LLMs with currently available tools, as well as supervision support.

Acknowledgements We would like to acknowledge the anonymous reviewers of VariEvolution 2023 for their valuable input and feedback.

REFERENCES

- [1] Mathieu Acher, Jean-Marc Jézéquel, and José A. Galindo. 2023. On Programming Variability with Large Language Model-based Assistant. In *Proceedings of the 27th International Systems and Software Product Line Conference (SPLC 2023) - Research Track (SPLC '23)*. ACM, New York, NY, USA.
- [2] Vander Alves, Christa Schwanninger, Luciano Barbosa, Awais Rashid, Peter Sawyer, Paul Rayson, Christoph Pohl, and Andreas Rummeler. 2008. An Exploratory Study of Information Retrieval Techniques in Domain Analysis. In *SPLC'08*. IEEE, 67–76.
- [3] Juliana Alves Pereira, Hugo Martin, Mathieu Acher, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning Software Configuration Spaces: A Systematic Literature Review. *Journal of Systems and Software (JSS)* (June 2021). <https://doi.org/10.1145/nnnnnnn.nnnnnnn>
- [4] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.
- [5] Wesley K. G. Assunção, Silvia R. Vergilio, and Roberto E. Lopez-Herrejon. 2023. ModelVars2SPL: From UML Class Diagram Variants to Software Product Line Core Assets. In *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*. Springer International Publishing, 221–250. https://doi.org/10.1007/978-3-031-11686-5_9
- [6] Colin Atkinson, Christian Bunse, and Joachim Bayer. 2002. *Component-based product line engineering with UML*. Pearson Education.
- [7] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. *Program Synthesis with Large Language Models*. Technical Report arXiv:2108.07732. arXiv. <https://doi.org/10.48550/arXiv.2108.07732> [cs] type: article.
- [8] Önder Babur. 2016. Statistical analysis of large sets of models. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. ACM, 888–891. <https://doi.org/10.1145/2970276.2975938>
- [9] Sahar Badihi, Yi Li, and Julia Rubin. 2021. EqBench: A Dataset of Equivalent and Non-equivalent Program Pairs. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021*. IEEE, 610–614. <https://doi.org/10.1109/MSR52588.2021.00084>
- [10] Ebrahim Bagheri, Faezeh Ensan, and Dragan Gasevic. 2012. Decision support for the software product line domain engineering lifecycle. *Automated Software Engineering* 19, 3 (2012), 335–377.
- [11] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A survey of variability modeling in industrial practice. In *The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa, Italy, January 23 - 25, 2013*. ACM, 7:1–7:8.
- [12] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*. <https://arxiv.org/abs/2204.06745>
- [13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [14] Jessie Carbonnel. 2018. *L'analyse formelle de concepts: un cadre structurel pour l'étude de la variabilité de familles de logiciels*. Ph.D. Dissertation. Université Montpellier.
- [15] Jessie Carbonnel, Marianne Huchard, André Miralles, and Clémentine Nebut. 2017. Feature model composition assisted by formal concept analysis. In *ENASE: Evaluation of Novel Approaches to Software Engineering*. SciTePress, 27–37.
- [16] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. 2005. An approach to constructing feature models based on requirements clustering. In *RE'05*. 31–40. <https://doi.org/10.1109/RE.2005.9>
- [17] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. *Evaluating Large Language Models Trained on Code*. Technical Report arXiv:2107.03374. arXiv. <https://doi.org/10.48550/arXiv.2107.03374> arXiv:2107.03374 [cs] type: article.
- [18] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).
- [19] Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. PyMT5: multi-mode translation of natural language and Python code with transformers. <https://doi.org/10.48550/arXiv.2010.03150> arXiv:2010.03150 [cs]
- [20] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019).
- [21] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.
- [22] Jean-Marc Davril, Edouard Delfosse, Negar Harii, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. 2013. Feature Model Extraction from Large Collections of Informal Product Descriptions. In *ESEC/FSE*.
- [23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [24] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathy Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2021. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts. arXiv:2112.06905 [cs.CL] <https://arxiv.org/abs/2112.06905>
- [25] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013*. IEEE Computer Society, 25–34.
- [26] Wolfram Fenske, Jens Meinicke, Sandro Schulze, Steffen Schulze, and Gunter Saake. 2017. Variant-preserving refactorings for migrating cloned products to a product line. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*. IEEE Computer Society, 316–326. <https://doi.org/10.1109/SANER.2017.7884632>
- [27] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Felice dell'Orletta. 2013. Mining commonalities and variabilities from natural language documents. In *SPLC*.

- [28] Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting Numerical Reasoning Skills into Language Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 946–958. <https://doi.org/10.18653/v1/2020.acl-main.89>
- [29] Github. 2021. *GitHub Copilot · Your AI pair programmer*. <https://copilot.github.com>
- [30] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *ASE*.
- [31] Nili Itzik and Iris Reinhartz-Berger. 2014. SOVA - A Tool for Semantic and Ontological Variability Analysis. In *Joint Proceedings of the CAiSE 2014 Forum and CAiSE 2014 Doctoral Consortium*. 177–184. <http://ceur-ws.org/Vol-1164/PaperDemo06.pdf>
- [32] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 497–508.
- [33] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How Can We Know What Language Models Know? *Transactions of the Association for Computational Linguistics* 8 (2020), 423–438. https://doi.org/10.1162/tacl_a_00324 arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00324/1923867/tacl_a_00324.pdf
- [34] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-Based Sampling of Software Configuration Spaces. In *Proceedings of the International Conference on Software Engineering (ICSE)*.
- [35] Thomas Krismayer, Rick Rabiser, and Paul Grünbacher. 2017. Mining Constraints for Event-Based Monitoring in Systems of Systems. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 826–831.
- [36] Charles W. Krueger. 2001. Easing the Transition to Software Mass Customization. In *Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers (Lecture Notes in Computer Science, Vol. 2290)*. Springer, 282–293.
- [37] Thomas Wolf Lewis Tunstall, Leandro von Werra. 2022. *Natural Language Processing with Transformers, Revised Edition [Book]*. <https://www.oreilly.com/library/view/natural-language-processing/9781098136789/> ISBN: 9781098136796.
- [38] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-Level Code Generation with AlphaCode. <https://doi.org/10.48550/arXiv.2203.07814> arXiv:2203.07814 [cs]
- [39] Crescencio Lima, Wesley K. G. Assunção, Jabier Martinez, William D. F. Mendonça, Ivan do Carmo Machado, and Christina von Flach G. Chavez. 2019. Product line architecture recovery with outlier filtering in software families: the Apo-Games case study. *J. Braz. Comput. Soc.* 25, 1 (2019), 7:1–7:17. <https://doi.org/10.1186/s13173-019-0088-4>
- [40] Roberto E. Lopez-Herrejon, Jabier Martinez, Wesley Klewerton Guez Assunção, Tewfik Ziadi, Mathieu Acher, and Silvia Regina Vergilio (Eds.). 2023. *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*. Springer International Publishing. <https://doi.org/10.1007/978-3-031-11686-5>
- [41] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017*. ACM, 38–41. <https://doi.org/10.1145/3109729.3109748>
- [42] Jabier Martinez, Nicolas Ordoñez, Xhevahire Tërnav, Tewfik Ziadi, Jairo Aponte, Eduardo Figueiredo, and Marco Túlio Valente. 2018. Feature location benchmark with argoUML SPL. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*. ACM, 257–263. <https://doi.org/10.1145/3233027.3236402>
- [43] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Name suggestions during feature identification: the varioclouds approach. In *Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016*. ACM, 119–123.
- [44] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2023. Bottom-Up Technologies for Reuse: A Framework to Support Extractive Software Product Line Adoption Activities. In *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*. Springer International Publishing, 355–377. https://doi.org/10.1007/978-3-031-11686-5_14
- [45] Jabier Martinez, Tewfik Ziadi, Mike Papadakis, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2018. Feature location benchmark for extractive software product line adoption research using realistic and synthetic Eclipse variants. *Inf. Softw. Technol.* 104 (2018), 46–59. <https://doi.org/10.1016/j.infsof.2018.07.005>
- [46] Raul Medeiros, Jabier Martinez, Oscar Diaz, and Jean-Rémy Falleri. 2023. Visualizations for the evolution of Variant-Rich Systems: A systematic mapping study. *Inf. Softw. Technol.* 154 (2023), 107084. <https://doi.org/10.1016/j.infsof.2022.107084>
- [47] Alexandr Murashkin, Michal Antkiewicz, Derek Rayside, and Krzysztof Czarnecki. 2013. Visualization and exploration of optimal variants in product line engineering. In *17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013*, Tomojo Kishi, Stan Jarzabek, and Stefania Gnesi (Eds.). ACM, 111–115. <https://doi.org/10.1145/2491627.2491647>
- [48] I Made Murwantara, Behzad Bordbar, and Leandro L. Minku. 2014. Measuring Energy Consumption for Web Service Product Configuration. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services (Hanoi, Viet Nam) (iiWAS)*. ACM, New York, NY, USA, 224–228.
- [49] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using Bad Learners to Find Good Configurations. In *Proceedings of the European Software Engineering Conference/Foundations of Software Engineering (ESEC/FSE)*. 257–267.
- [50] Nan Niu and Steve M. Easterbrook. 2009. Concept analysis for product line requirements. In *AOSD '09*, Kevin J. Sullivan, Ana Moreira, Christa Schwanninger, and Jeff Gray (Eds.). ACM, 137–148.
- [51] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
- [52] Iris Reinhartz-Berger. 2014. Can domain modeling be automated? Levels of automation in domain modeling. In *SPLC '14*. 359.
- [53] Iris Reinhartz-Berger, Arnon Sturm, and Yair Wand. 2013. Comparing functionality of software systems: An ontological approach. *Data Knowl. Eng.* 87 (2013), 320–338. <https://doi.org/10.1016/j.datak.2012.09.005>
- [54] Julia Rubin. 2015. *Cloned Product Variants: From Ad-hoc to Well-managed Software Reuse*. Ph. D. Dissertation. University of Toronto, Canada. <http://hdl.handle.net/1807/68443>
- [55] Julia Rubin and Marsha Chechik. 2013. N-way model merging. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE '13, Saint Petersburg, Russian Federation, August 18-26, 2013*. ACM, 301–311.
- [56] Norbert Siegmund, Alexander Grebhahn, Christian Kästner, and Sven Apel. 2015. Performance-Influence Models for Highly Configurable Systems. In *ESEC/FSE '15*.
- [57] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615* (2022).
- [58] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the truth: benchmarking the techniques for the evolution of variant-rich systems. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*. ACM, 26:1–26:12. <https://doi.org/10.1145/3336294.3336302>
- [59] Shaghayegh Tavassoli, Carlos Diego Nascimento Damasceno, Ramtin Khosravi, and Mohammad Reza Mousavi. 2022. Adaptive behavioral model learning for software product lines. In *SPLC '22: 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume A, Alexander Felfernig, Lidia Fuentes, Jane Cleland-Huang, Wesley K. G. Assunção, Andreas A. Falkner, Maider Azaña, Miguel Á. Rodríguez Luaces, Megha Bhushan, Laura Semini, Xavier Devroey, Cláudia Maria Lima Werner, Christoph Seidl, Viet-Man Le, and José Miguel Horcas (Eds.)*. ACM, 142–153. <https://doi.org/10.1145/3546932.3546991>
- [60] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, and Olivier Barais. 2017. Learning-Contextual Variability Models. *IEEE Software* 34, 6 (Nov. 2017), 64–70. <https://hal.inria.fr/hal-01659137>
- [61] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 39–50.
- [62] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. 2012. Automated Inference of Goal-Oriented Performance Prediction Functions. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 190–199.
- [63] Cemal Yilmaz, Myra B Cohen, and Adam A Porter. 2006. Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces. *IEEE Transactions on Software Engineering* 32, 1 (2006), 20–34.
- [64] Yi Zhang, Jianmei Guo, Eric Blais, Krzysztof Czarnecki, and Huiqun Yu. 2016. A Mathematical Model of Performance-Relevant Feature Interactions. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 25–34.
- [65] Tewfik Ziadi, Luz Frias, Marcos Aurélio Almeida da Silva, and Mikal Ziane. 2012. Feature identification from the source code of product variants. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 417–422.
- [66] Tewfik Ziadi and Jean-Marc Jézéquel. 2006. Software Product Line Engineering with the UML: Deriving Products. In *Software Product Lines*, Timo Käkölä and Juan C. Dueñas (Eds.). Springer, 557–588.