# On the Variability Secrets of an Online Video Generator

Guillaume Bécan, Mathieu Acher, Jean-Marc Jézéquel, Thomas Menguy

# On the Variability Secrets of an Online Video Generator

Guillaume Bécan, Mathieu Acher, Jean-Marc Jézéquel
Inria - IRISA Université de Rennes 1
Campus de Beaulieu
35000 Rennes, France
{gbecan, mathieu.acher, jean-marc.jezequel}@irisa.fr

Thomas Menguy
Wildmoka
Sophia Antipolis, France
thomas@wildmoka.com

## ABSTRACT

We relate an original experience concerning a popular online video service that offers to generate variants of an humorous video. To further the understanding of the generator, we have reverse engineered its general behavior, architecture, as well as its variation points and its configuration space. The reverse engineering also allows us to create a new generator and online configurator that proposes 18 variation points – instead of only 3 as in the original generator. We explain why and how we have collaborated and are collaborating with the original creators of the video generator. We also highlight how our reverse engineering work represents a threat to the original service and call for further investigating variability-aware security mechanisms.

## Categories and Subject Descriptors

D.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*; D.2.9 [**Software Engineering**]: Management—*Software configuration management*

## General Terms

Experimentation

## Keywords

Reverse engineering, Software product line, Configurator, Video generator, Security

## 1. INTRODUCTION

The concepts of software product lines and variability are more and more observed in numerous domains and contexts [8, 16, 34, 35]. Abstractions, theories, techniques, languages, and tools have been developed and are widely used for managing and building *variants* of user interfaces, requirements, source code, architectures, 3D models, or design models, to name a few [6, 10, 11, 17, 26, 27, 32, 36, 39]. The applicability of variability techniques seems infinite – any artefact is virtually subject to customisation, configuration, or extension.

We report on an experience related to an online video service. The service offers to generate variants of an humorous video. Internet users simply have to type their name, select 3 options (or click on a button for a random selection), and a particular video is launched and visualised in the browser (see Figure 1, page ). The service is quite popular and successful: more than 1.7M of video variants have been generated in 1 week.

We had an interest in the Web site, since the service claims that a particular video is resulting from a combination among billions. Moreover the term *generator* is explicitly employed. Our original intuition was that the video service resembles to a software product line, i.e., generative techniques and variability are likely to be present.

We have first noticed a common property of all the video variants: 18 sequences of videos are assembled; for each sequence of a video, numerous alternatives are possible. These variants represent a product line of videos. We have then contacted the creators of the video generator with two research questions and goals in mind: (1) can we *automate* the extraction of variation points and configurations (video variants)? (2) can we re-engineer another generator and configurator?

The creators accepted to collaborate and provided us with an access to an offline version of the generator. However we did not have access to the source code neither in the server side nor in the client side. Like this, we can study the generator as a black box system – as it would be the case in reality – but without disturbing the deployed Web generator (e.g., with hundreds of HTTP requests). The interest for the creators was to audit their systems w.r.t. variability.

To address the two research questions and further the understanding of the generator, we have *reverse engineered* its general behavior, architecture, as well as its variation points and its configuration space. Overall, we have collected 350,000+ configurations and 1620 video sequences. We have automatically determined to which part of the video the 1620 sequences correspond. We have also computed some statistics to understand the configuration process. The paper describes the reverse engineering process, how we gather variability information and insights about the configurations, and points out the importance of barriers for complicating or avoiding a large-scale extraction of configurations.

The reverse engineering work have also allowed us to re-engineer a new generator and configurator. This time, users can configure in a fine-grained way the 18 variations points – instead of only 3 in the original version. 15 new variation points are now apparent, out of which 13 are configurable (i.e., exhibiting at least 2 alternatives). For the 3 variation points also present in the original configurator, users can now explicitly choose the alternatives. Thumbnails representative of the video sequences are depicted and help the users

during the selection. The paper describes the re-engineering process, highlights the large degree of automation of devising a new configurator, and discusses the differences with the original configurator.

Finally, we explain why the re-engineering of a configurator and the exhibit of new variation points could be a threat for the creators of the original video generator. The main reason is that the new configurator lets users control, choose and visualize any alternative – thus dramatically limiting the *surprise* effect. Our experience shows that the scoping of variability (i.e., what is visible to users) is a strategic solution and, as such, mechanisms to prevent the reverse engineering of variability have to be considered. We report on our preliminary exchanges with the creators of the video generator.

This paper is both an *experience report* and a *research-in-progress*. The intended audience are (1) researchers working on reverse engineering of variability; (2) developers of generators, configurators, and variability-intensive systems. The former shall benefit from an extraction and analysis of a large configuration set – in an original setting. The latter can learn from our experience of (re-)engineering a configurator and scoping variability. We also highlight the need to develop defensive mechanisms for increasing the difficulty of reverse engineering variability. Besides, we believe that the originality of the considered artefacts (videos, Web applications), the popularity of the online service, and the intuitiveness of the application domain can be of interest to a broad audience – not necessarily experts of product lines or variability (e.g., students).

The remainder of the paper is organized as follows. Section 2 introduces the online video generator *"Bref, 30 ans"*. Section 3 presents the reverse engineering process which allows us to understand in a fine-grained way the variability and the general behavior of the generator. Section 4 describes the re-engineering of the configurator and generator. Section 5 discusses related work. Section 6 summarizes the contributions of the paper and concludes.

# 2. "BREF, 30 ANS": AN ONLINE VIDEO GENERATOR

## 2.1 Bref, 30 ans: A Family of Video Variants

*Canal+* (a French premium cable television channel) launched several initiatives for celebrating its 30th anniversary. Among others, the Web site `http://bref30ans.canalplus.fr` has been developed and provides an online service to generate variants of an humorous video. A video variant follows a story line in which the historic perception of *Canal+* by the main character of *Bref*[1] is presented in an humoristic way. A video variant typically includes sequences related to famous programs of *Canal+* while several guests (e.g., actors, presenters) are part of the story line.

Internet users simply have to type their name (step 1) and select 3 options (step 2, see Figure 1). A particular video is launched and visualised in the browser. The information of step 1 is used during the first sequence (the name appears in a blank screen). The three options of step 2 correspond to programs ("souvenirs" in Figure 1) of *Canal+* that a user wants to include in the variant. Users can also choose a randomized selection, corresponding to "Je ne veux pas choisir" (see Figure 1) that can be translated as "I do not want to choose". The service is quite popular and successful. More than 1.7M of video variants have been generated in 1 week.

[1] *Bref* is a popular television show promoted by *Canal+*

## 2.2 Motivation and Research Questions

We had an interest in the Web site since the video service resembles to a software product line. From a research perspective, the online video generator is likely to constitute an interesting case study (i.e., "an empirical inquiry that investigates a contemporary phenomenon within its real-life context" [37]). From a dissemination perspective, the relative accessibility, intuitive nature, and concreteness of the case study are good qualities for a broad audience. For example, the case study could be used for presenting concepts of product lines and variability to students [4]. Interestingly, the main character of *Bref* states, at the very end, that a particular video is resulting from a combination among billions; the term *generator* is even explicitly employed. Hence we suspect that generative techniques and variability are likely to be present.

We sought to verify this assumption and considered the following research questions: What is the variability and commonality of the generator? How is variability modeled and implemented? How many variation points, commonalities, constraints, and configurations are there? How configurations and video variants are generated on-demand?

In short, our original motivation was to *understand* in a fine-grained way the online video generator. Another related motivation was to determine whether we can *re-engineer* the generator and provide an alternative configurator, e.g., with more than 3 options. For both understanding and re-engineering the generator, we adopted a *reverse engineering* approach. In essence, reverse engineering *"consists in deriving information from the available software artifacts and translating it into abstract representations more easily understandable by humans"* [9]. Here, we wanted to extract variability information and to translate it into an understandable form, so that we can extract insights and start the re-engineering of a new configurator.

## 2.3 Research Method

After some preliminary observations, we contacted the creators of the video generator for collaborating on the topic. The creators accepted and provided us with an access to an offline version of the generator. The interest for the creators was to audit their systems w.r.t. variability. Importantly, we did not have access to the original source code neither in the server side nor in the client side (e.g., the JavaScript was minimized and obfuscated, as in the original version). By doing so, we wanted to replicate the exact conditions in which is confronted a person in charge of reverse engineering (e.g., an attacker). Overall, we studied the generator as a *black box* system – as it would be the case in reality – but without disturbing the deployed service. For instance, we performed hundreds of HTTP requests without overwhelming the network with traffic.

# 3. REVERSE ENGINEERING VARIABILITY SECRETS

Reverse engineering scenarios usually assume an access to the source code or any form of documentation, for example, for achieving design recovery of an existing application [9].


**Figure 1: Online video generator, step 2/3 of the configuration process (left-hand side): a user can select 3 options ("souvenirs") or let the system randomly choose. A variant of a video is then played (step 3).**

It is not the case in our context. Not having access to the original source code of the application was a deliberate decision, challenging us to retrieve the variability secrets of the generator.

## 3.1 Preliminary Inspection

We started the investigation of the Web generator through a manual analysis of its behavior. We first sought to understand, at a high level, what are the properties of the generated videos. We manually generated some videos and we reported the following observations:

- the story line seems fixed, with some recurrent sequences, as well as the length of the video (3 minutes);
- the variability not only concerns the 3 souvenirs of the configurator, but also some parts of the video (e.g., some actors are replaced by others)

In a sense, we quickly verified the intuition that the generator exhibits commonality and variability.

We then investigated, at the technical level, how the videos are actually produced. We relied on basic Web tooling support to analyze the JavaScript and the requests with the server; we learned that:

- a video variant is a combination of video sequences;
- each video is encoded in .ts format and accessible via a specific URL, based on the filename;
- there is a specification for executing the video sequences in order (a *playlist*-like file).

At this step, we understood the general principle of the generator but ignored (or had some doubts) about the variation points (e.g., how many variation points are there in the generator?).

## 3.2 Large Scale Extraction of Configurations

Due to the limits of a manual inspection, we thus sought to *automatically* extract as many video variants as possible. We first considered the use of crawlers to emulate the user behavior when selecting options in step 2 (see Figure 1); then we could get videos. However the approach is not really scalable since it requires to download every sequence of video, for each variant.

We thus decided to reverse engineer the way the playlist is retrieved. We analyzed the HTTP and JSON requests of the client with the server. We came to the conclusion that a playlist is documented as an ordered list of 18 string-based values (an example is given below):

```
"sq": ["dwlcjv", "1y60t3z", "1lyfhk", "wqzv0y",
"1xxivi2", "1oxnvtu",  "lolbe9", "wvo06o",
"1u6y5t2", "1eqb8bw", "1j9aij7", "nr7jom",
"1jmv11y", "1qgn9dh", "1bv7rka", "19ykyyw",
"5znrg7", "116hv1k"]
```

Each value of the ordered list actually corresponds a sub-playlist, accessible via an URL. For example, for ml7ila:

```
#EXTINF:05.72,
http://XXX/../ml7ila_med0.ts
#EXTINF:05.96,
http://XXX/../ml7ila_med1.ts
#EXTINF:03.96,
http://XXX/../ml7ila_med2.ts
#EXTINF:02.12,
http://XXX/../ml7ila_med3.ts
```

From a conceptual point of view, the ordered list corresponds to a *configuration* (we will use this term in the remainder of the paper). Overall we automatically generated 363,281 configurations. We also retrieved the sub-playlists corresponding to each value of the configurations. We combined wget and curl to operate over URLs.

We stopped at 350,000+ configurations as we did not observe new values composing a configuration, compared to a 50,000+ configurations sample we first obtained. An attacker is likely to limit the number of requests in order to not draw attention. A defender can also apply defensive strategies to limit the requests. An interesting research direction is thus to determine the number of requests for which the reverse engineering and understanding remains effective (e.g., comprehensive). We discuss this perspective in the conclusion.

## 3.3 Automated Analysis of Configurations

We processed the collected set of configurations to further understand the variability of the generator. We observed

that each configuration is constituted of 18 values, that is, there are 18 *variation points* (we will use this term in the remainder of the paper). We collected some statistics for each of the variation point (see Table 1). 2 variation points, VP2 and VP18, have only 1 alternative. We can consider them as two commonalities or mandatory features of a video variant. 16 variation points exhibit at least 2 alternatives and are thus configurable.

The number of alternatives varies for a given variation point. VP1, VP7, and VP16 correspond to "souvenirs" and the three options originally proposed in the video generator of Bref, 30 ans. These variation points have the highest number of alternatives (resp. 63, 51, and 86). There are only two alternatives for VP9. The first alternative appears in 362,903 configurations; the second alternative only 378 times, i.e., 0.1 % of the configurations contains this video sequence. More generally, we observed that the frequencies of alternatives for a given variation point:

- are almost uniformly distributed (e.g., VP11 has 6 alternatives, and each alternative appears between 60,194 and 60,634 times). It is the case for VP1, VP2, VP4, VP5, VP6, VP7, VP10, VP11, VP15, VP16, VP17, VP18;
- or the distribution support is not equally probable (e.g., see the previous explanations with VP9; for VP3, an alternative only appears 3253 times while the other frequencies of alternatives of VP3 are around 42,000). It is the case for VP3, VP8, VP9, VP12, VP13, VP14.

Besides we observed hard-constraints between the alternatives *within* the sample, e.g., there are some logical, binary exclusions. For example, some alternatives of VP16 are mutually exclusive with the least frequent alternative of VP9. Similarly there are soft constraints [12] *within* the sample, i.e., there is a conditional probability of a configuration to contain one alternative together with another. It is tempting to interpret these hard- or soft-constraints as design decisions of the creators. However, we recall that the configuration set is only a subset of the whole configuration set. Some hard- or soft-constraints may be due to the *incompleteness* of the extracted configuration set. It is thus premature to draw any conclusions.

Finally, it should be noted that an alternative is constituted of a series of videos (e.g., in the example of ml7ila, there are 4 videos); in total, we collect 1620 video sequences. Another form of variability observed in the generator is as follows. The videos constituting an alternative may be partly assembled. For example, in the case of ml7ila, only the 3 last videos (instead of the 4 videos) can be assembled – the goal is to reduce the length of the video or propose yet another subtle variant of a video. A thorough analysis of the obfuscated client side code could reveal such kind of variability. We leave it as future work.

Overall there is a deliberate intention and strategy for *surprising* the users of the video generator. In the same vein, it should be noted that each configuration of our sample is *unique*. The rationale is certainly that users have to obtain a new video variant each time they visit the online service.

# 4. RE-ENGINEERING THE GENERATOR AND CONFIGURATOR

The result of the previous reverse engineering process is as follows:

| Variation Point | # Alternatives |
|---|---|
| VP1 | 63 |
| VP2 | 1 |
| VP3 | 9 |
| VP4 | 34 |
| VP5 | 15 |
| VP6 | 25 |
| VP7 | 51 |
| VP8 | 30 |
| VP9 | 2 |
| VP10 | 6 |
| VP11 | 6 |
| VP12 | 12 |
| VP13 | 21 |
| VP14 | 28 |
| VP15 | 6 |
| VP16 | 86 |
| VP17 | 4 |
| VP18 | 1 |

**Table 1: Number of alternatives per variation point**

- an abstraction of variation points and alternatives. We can infer a variability model (e.g., a feature model);
- an understanding of how video variants are generated. It is simply an ordered combination of video sequences.

Our objective was to demonstrate the feasibility of devising a new generator and configurator – this time exhibiting all variation points. The re-engineering of the same exact version of the generator could have been another objective. In particular we could have tried to replicate the process of generating unique configurations that respect the frequencies of alternatives. We leave it as future work.

## 4.1 New Configurator

By reusing our observations on how the variability is realized in the original service and by analyzing some parts of the Javascript code, we were able to re-engineer a new configurator. The result is depicted in Figure 2. Users can configure in a fine-grained way the 16 variations points – instead of only 3 in the original version – and visualize the result of the two common videos of all variants. For instance, the variation point "Tu m'écoutes", corresponding to VP8 in Table 1, depicts 30 alternatives. For the 9th sequence (VP9, "Jean Jacques") two videos can be selected, etc.

15 new variation points are now apparent, out of which 13 are configurable (i.e., exhibiting at least 2 alternatives). For the 3 variation points also present in the original configurator (the "souvenirs"), users can now explicitly choose the alternatives.

For each variation point, thumbnails representative of the video sequences are depicted and help the users during the selection. We generated the thumbnails from the videos sequences using ffmpeg[2]. Overall the creation of the new configurator was largely automated thanks to the result of the reverse engineering process. The manual task consists in giving a proper name to each variation points (in the form of french sentences) and engineering the generator for playing the video sequences. It should be noted that our generator assembles and reuses videos located in the servers of the original service.

[2]https://www.ffmpeg.org/

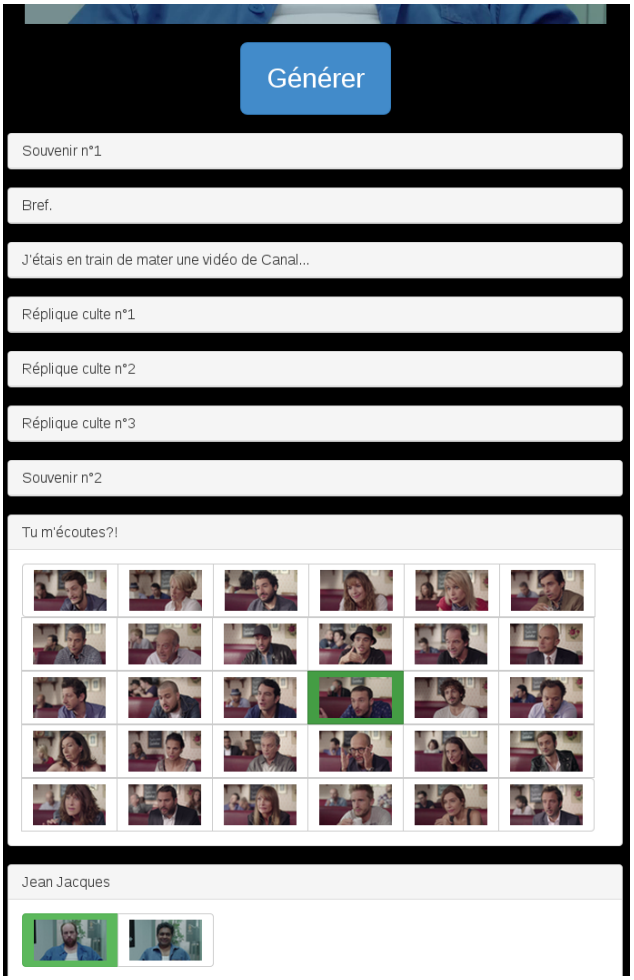


**Figure 2: Re-engineering of the novel configurator (excerpt): users can now select a specific video for the 18 variation points identified during the reverse engineering of the original generator**

## 4.2 The Threats of Variability Secrets

We presented the re-engineered configurator to the original creators of the video generator. The new configurator represents an important threat and could "kill the original idea". That is, with our solution, the surprise effect is limited when getting a new video variant. Instead, users can control, choose and visualize any alternative (video sequence). The creators of the generator did not have this intention. In other words, they have taken a strategic decision for, on-purpose, delimiting the scope of variability and restraining the visibility of some variation points and alternatives.

The main lessons we can learn from this experience is that (1) the large scale extraction of configurations is key and allows us to re-engineer another competing configurator that could have annihilated the original innovation of the generator; (2) whenever the scoping of variability does matter, mechanisms should be used to prevent or limit the mining in the large of non visible alternatives.

With regards to the last point, some defensive techniques have been considered, for instance, the use of block lists for banning IP addresses that perform numerous HTTP requests in a short amount of time. Other security mechanisms have been discussed with the creators of the video generator but are out of the scope of this paper. Our experience so far shows that the problem of protecting the variability requires a careful attention and certainly the use of innovative techniques.

# 5. RELATED WORK

## 5.1 Variability and Video

The use of variability techniques in the video domain is not new and has already been reported [3, 17, 25]. However the underlying motivations and realizations vastly differ. In [3, 25], Moisan *et al.* model variability of video processing *algorithms*. The goal is to systematize the parametrization of highly configurable algorithms composing a processing chain. The algorithms exhibit variability at runtime and are able to adapt to evolving and variable contexts (e.g., a change of luminosity in the video).

In [17], Galindo *et al.* presented an approach for generating synthetic variants of a video. The developed generator relies on a *base* (a video sequence) that is modified for changing the luminosity, adding elements (e.g., vehicles) in the scene, etc. The derivation process is thus quite different. Instead of assembling pre-defined video sequences, the derivation consists in altering an existing video sequence through a series of transformations. The techniques are more advanced and are based on video processing algorithms. Another important difference concerns the objective. The industrial motivation of [17] is that companies providing or consuming algorithms have severe difficulties to evaluate or compare the quality of existing solutions. The use of variability techniques allows practitioners to generate as much as possible videos, with different characteristics, in order to *test or benchmark* algorithms on a rich and diverse dataset.

It is possible to apply the video processing techniques exposed in [17] in the case of the online video generator. We envision to process variants of pre-defined video sequences and then assemble the new variants for deriving a comprehensive video.

## 5.2 Reverse Engineering Variability

Numerous works address the reverse engineering of variability [1,5–7,10,12–14,18–20,23,26,27,31,33,33,36,38,39].

The artefacts considered as input are product descriptions [12, 14, 18, 19], requirements [6, 10, 27, 36], configurators [1], source code [5,26,33,39], or models [23]. The objectives are usually to locate features or configuration options, to formalize logical relations, and possibly to re-engineer a system with a generative, variability-based approach.

The experience report of this paper leans on (1) an intensive and manual observation of a generator (2) the use of automated techniques for a large-scale extraction of configurations and a classification of variations points. The approach has the originality of targeting videos and Web artefacts. Abbasi *et al.* [1] propose generic techniques to reverse engineer Web configurators, but do not develop techniques to extract configurations. Our reverse engineering process leads to a fine-grained understanding of variability (see Section 3) and a re-engineering of the configurator (see Section 4).

## 5.3 Configuration

Techniques for automatic reasoning have been proposed to manage the configuration process, being for guiding users (e.g., [24, 28]) or for piloting an adaptation [32]. Decision or feature-based configuration techniques and environment have been proposed [2, 15, 28–30]. Web configurators are engineered along different practices [21] for representing options, managing constraints, guiding users, scheduling the configuration process, etc. We can reuse and apply part of these techniques when re-engineering the configurator (see Section 4).

# 6. CONCLUSION

We reported an experience in reverse engineering and re-engineering the variability secrets of an online video generator. We described our collaboration with the original creators of the video generator to audit their systems w.r.t. variability. The contributions can be summarized as follows:

- We described how we have inferred the general behavior of the generator (including its variation points) and how we have automatically extracted and classified 1620 video sequences out of 350,000+ configurations;
- We presented the creation of a new configurator exposing 18 variation points instead of only 3 as in the original video generator;
- We explained why letting an attacker re-engineers a new configurator is a threat for the creators, thus motivating the use of defensive mechanisms for complicating his or her task.

**Education material.** Describing the implementation of the video generator (or even relating our experience) can be considered for illustrating the concept of generative techniques and variability. We believe the concreteness and the relative accessibility of the case study are likely to reach a broad audience (e.g., students [4]). At University of Rennes 1 (graduate level), we have already started to illustrate the software product line course with the video generator.

**Lessons learned.** A key aspect of our work is that the ability to reverse engineer a large number of configurations dramatically helps to understand the variability and behavior of the generator. It also helps us to devise a quite comprehensive configurator – at least competing with the original solution. Our conclusion – and the main lesson learned – is that defensive mechanisms should be considered and developed to prevent such large-scale extraction of configurations.

**Future work.** The main limitation of our current and ongoing work is that we do not yet validate our understanding and re-engineering with the creators of the generator. For instance, we ignore if the extraction of alternatives and 1620 video sequences is *complete*. When considering the sample of 350,000+ configurations, we did not observe new alternatives compared to 50,000+ configurations. However we may still have missed very low frequencies of alternatives. Similarly we cannot draw definitive conclusions w.r.t. hard and soft constraints between alternatives since the configuration set is by construction only a sample. Mining a larger number of configurations is a possible strategy and can increase the statistical significance of the results. Generally speaking it is hard to cover the whole configuration space due to the exponential combinations of alternatives. An even more ambitious research direction is to characterize the minimum number of configurations an attacker needs to have an accurate and complete understanding of the generator and configurator. We are currently continuing the discussions with the creators of the video generator along these lines. A comprehensive report of our future exchanges is planned but is out of the scope of this *research-in-progress* paper.

We conjecture that similar security issues, due to the presence of hidden variability, arise in other contexts – beyond the case of the Web video generator. Some proprietary systems or applications hide on purpose some features. Users typically have to pay for activating the features. For example, Windows 7 was subject to a hack that allows a clean install of the operating system using an upgrade disc rather than the full version upgrades [22]. It calls for investigating *variability-aware security* mechanisms.

# 7. REFERENCES


[1] E. K. Abbasi, M. Acher, P. Heymans, and A. Cleve. Reverse engineering web configurators. In *CSMR/WRCE'14*, 2014.

[2] E. K. Abbasi, A. Hubaux, and P. Heymans. A toolset for feature-based configuration workflows. In *Proc. of SPLC'11*, pages 65–69, 2011.

[3] M. Acher, P. Collet, P. Lahire, S. Moisan, and J.-P. Rigault. Modeling variability from requirements to runtime. In *ICECCS'11*, pages 77–86. IEEE, 2011.

[4] M. Acher, R. E. Lopez-Herrejon, and R. Rabiser. A survey on teaching of software product lines. In *Eight International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'14)*, Nice, France, jan 2014. ACM.

[5] R. Al-msie'deen, A.-D. Seriai, M. Huchard, C. Urtado, and S. Vauttier. Mining features from the object-oriented source code of software variants by combining lexical and structural similarity. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 586–593, Aug 2013.

[6] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *SPLC'08*, pages 67–76. IEEE, 2008.

[7] N. Andersen, K. Czarnecki, S. She, and A. Wasowski. Efficient synthesis of feature models. In *Proceedings of SPLC'12*, pages 97–106. ACM, 2012.

[8] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer-Verlag, 2013.

[9] G. Canfora, M. Di Penta, and L. Cerulo. Achievements and challenges in software reverse engineering. *Commun. ACM*, 54(4):142–151, Apr. 2011.

[10] K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. In *RE'05*, pages 31–40, 2005.

[11] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[12] K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. In *SPLC'08*, pages 22–31, 2008.

[13] K. Czarnecki and A. Wasowski. Feature diagrams and logics: There and back again. In *SPLC'07*, pages 23–34. IEEE, 2007.

[14] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans. Feature model extraction from large collections of informal product descriptions. In *ESEC/FSE'13*, 2013.

[15] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. Galindo. Integrating heterogeneous variability modeling approaches with invar. In *Proc. of VAMOS'13*, page 8, 2013.

[16] M. Erwig and E. Walkingshaw. The choice calculus: A representation for software variation. *ACM Trans. Softw. Eng. Methodol.*, 21(1):6, 2011.

[17] J. A. Galindo, M. Alférez, M. Acher, and B. Baudry. A variability-based testing approach for synthesizing video sequences. In *ISSTA'14*, 2014.

[18] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. Reverse engineering feature models from programs' feature sets. In *WCRE'11*, pages 308–312. IEEE, 2011.

[19] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. On extracting feature models from sets of valid feature combinations. In *FASE'13*, volume 7793 of *LNCS*, pages 53–67, 2013.

[20] M. Janota, V. Kuzina, and A. Wasowski. Model construction with external constraints: An interactive journey from semantics to syntax. In *MODELS'08*, volume 5301 of *LNCS*, pages 431–445, 2008.

[21] E. Khalil Abbasi, A. Hubaux, M. Acher, Q. Boucher, and P. Heymans. The anatomy of a sales configurator: An empirical study of 111 cases. In *CAiSE'13*, 2013.

[22] E. Ligman. Regardless of what any hack says, a windows 7 upgrade is an upgrade. what you need to know., oct 2009.

[23] J. Martinez, T. Ziadi, J. Klein, and Y. L. Traon. Identifying and visualising commonality and variability in model variants. In *Modelling Foundations and Applications - 10th European Conference, ECMFA*, pages 117–131, 2014.

[24] R. Mazo, C. Dumitrescu, C. Salinesi, and D. Diaz. Recommendation heuristics for improving product line configuration processes. In *Recommendation Systems in Software Engineering*, pages 511–537. 2014.

[25] S. Moisan, J.-P. Rigault, M. Acher, P. Collet, and P. Lahire. Run Time Adaptation of Video-Surveillance Systems: A Software Modeling Approach. In *8th International Conference on Computer Vision Systems (ICVS'2011)*, LNCS, Sophia Antipolis (France), Sept. 2011. Springer Verlag.

[26] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki. Mining configuration constraints: Static analyses and empirical results. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 6 2014.

[27] N. Niu and S. M. Easterbrook. Concept analysis for product line requirements. In K. J. Sullivan, A. Moreira, C. Schwanninger, and J. Gray, editors, *AOSD'09*, pages 137–148. ACM, 2009.

[28] A. Nöhrer and A. Egyed. Optimizing user guidance during decision-making. In *Proc. of SPLC'11*, pages 25–34, 2011.

[29] A. Pleuss, R. Rabiser, and G. Botterweck. Visualization techniques for application in interactive product configuration. In *Proc. of MAPLE/SCALE'11*, 2011.

[30] R. Rabiser, P. Grünbacher, and M. Lehofer. A qualitative study on user guidance capabilities in product configuration tools. In M. Goedicke, T. Menzies, and M. Saeki, editors, *ASE*, pages 110–119. ACM, 2012.

[31] U. Ryssel, J. Ploennigs, and K. Kabitzsch. Extraction of feature models from formal contexts. In *FOSD'11*, pages 1–8, 2011.

[32] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes. Using constraint programming to manage configurations in self-adaptive systems. *IEEE Computer*, 45(10):56–63, 2012.

[33] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Reverse engineering feature models. In *ICSE'11*, pages 461–470. ACM, 2011.

[34] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35(8):705–754, 2005.

[35] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47(1):6, 2014.

[36] N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In *SPLC'09*, pages 211–220. ACM, 2009.

[37] R. K. Yin. *Case Study Research: Design and Methods*. Sager, 2002.

[38] B. Zhang and M. Becker. Mining complex feature correlations from software product line configurations. In *The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13*, page 19, 2013.

[39] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. L. Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Symposium on Applied Computing, SAC 2014*, pages 1064–1071, 2014.