# Variability Modelling and Analysis During 30 Years

David Benavides(✉)

Department of Computer Languages and Systems, University of Seville, Seville, Spain
benavides@us.es

**Abstract.** Variability modelling and analysis are among the most important activities in software engineering in general and in software product line engineering in particular. In 1990, the FODA report supposed a revolution in the importance of modelling and analysing of variability. In 2020, 30 years of variability modelling and analysis will be celebrated. In this paper, a short overview of the history and the importance of variability modelling and analysis is given, in concordance to that anniversary and on the occasion of Stefania Gnesi's retirement. She was part of this amazing history.

**Keywords:** Software product lines · Feature models · Variability modelling

## 1 Variability Modelling as a Key Activity

Software systems are one of the engineering creations where variability is most important, to such an extent that variability is an intrinsic element of any software system. Variability can be defined as the ability of a software system to be adapted to different situations such as different users, operational environments or quality requirements. For example, a piece of code that calculates the shortest path for graphs can be prepared to calculate it for different types of graphs such as directed or undirected. We can even envision a graph library that can include this and other graph operations that could be assembled to support different kinds of graphs [12].

Variability management becomes very important to any kind of software system and more important when managing what is called a *software product line* [15]. A software product line can be defined as a set of software systems that share more commonalities than variabilities and that are able to operate in a given domain. Examples of software product lines can be found in many different domains such as operating systems (e.g. Android operating systems in different platforms such an mobile phones, TV sets or smart watches); car control systems (e.g. adaptations of a system for different car models); or web commerce solutions (e.g. different adapted solutions for different specific online stores). Variability management is defined as the set of activities that have to


M. H. ter Beek et al. (Eds.): Gnesi Festschrift, LNCS 11865, pp. 365–373, 2019.
https://doi.org/10.1007/978-3-030-30985-5_21

be performed to correctly deal with variability in this kind of scenarios to better profit from the commonalities of the systems while reducing cost and increasing quality in software production.

One of the crucial activities when managing variability is its modelling. Variability modelling can be done at different levels of abstraction. From top level activities such as requirements engineering to more concrete activities such as coding, delivery or deployment. For example, a requirements engineer may want to express the different elements that can be of interest to a given user (such as the supported graph types in the example above) and how these elements could depend on each other (for instance, that an algorithm of cycle detection is only available if a directed graph is selected). Likewise, during deployment, developers may want to express the different configurations available for deployment with different storage capabilities, different memory consumption or different number of processors.

Variability modelling has been present in the general software engineering literature since its infancy. As an example, in 1968 McIlroy's text in the NATO conference [13] one can already divine that variability modelling had to be one of the key activities when mass producing software products. Nevertheless, it is in 1990 that Kang *et al.* presented the FODA report and explicitly gave a way of modelling variability in the form of feature models.

## 2 The FODA Report in 1990

The FODA report can be considered as the kickoff for variability modelling and analysis [10]. It had and still has an immense impact in the software engineering literature in general and in the software product line engineering research and practice in particular. However, as shown in Fig. 1 it is remarkable that the recognition of the importance of the FODA report started only around 2005, that is, 15 years after its publication. There was still another peak in 2010 that was basically due to the consolidation of a new line of research, as we will explain further in Sect. 3.

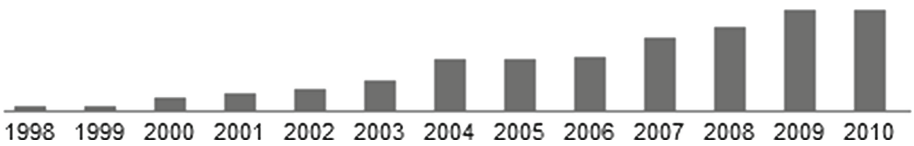


**Fig. 1.** Citations evolution of the FODA report (taken from Google scholar)

In 2020, 30 years of the FODA report will be celebrated, making this a good moment to remember some of the novelties that were presented at that time. It is not easy to find scientific contributions in software engineering that pass the test of time, but the FODA report is one such contribution. It is remarkable that a

scientific text with more than 4.500 citations in Google scholar was produced as a technical report instead of being a journal paper or a conference proceedings contribution. The report was produced in the Software Engineering Institute (SEI) where leading projects on software product line engineering were run. Not in vain, in 2000 one of the most well-known books on software product line engineering was produced [5].

One of the main outputs of the FODA report was what they called "feature models" as part of the domain modelling activity. A feature model is a compact representation of all possible products of a software product line and it is represented in a feature diagram as a tree-like structure composed of features and relationships among them [3]. A feature is defined as an increment in product functionality [1] and is a widely used concept in software product line engineering. Since the appearance of the FODA report there have been a lot of different notations for feature modelling, most of them well surveyed by Schobbens *et al.* [17]. There are different concepts and elements in the different notations, but most of them have the basic elements of what we often call "basic feature models".

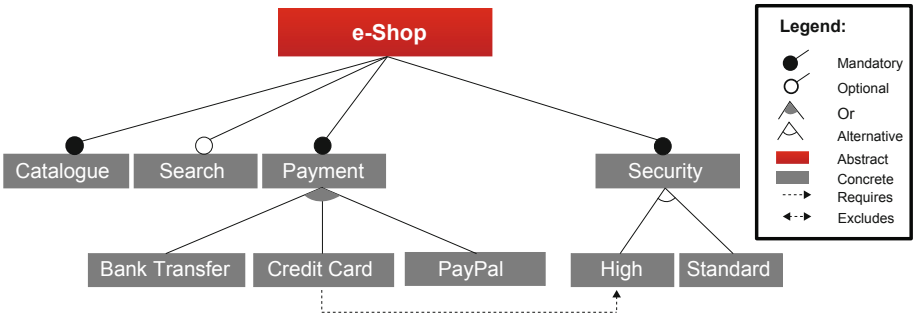


**Fig. 2.** Feature model of an e-shop software product line (from [16])

In basic feature models, features are depicted as boxes and relationships among features are depicted as arrows. Figure 2 shows a feature model of an e-shop software product line where the root is an abstract feature (i.e. it's used to model the concept, but it has no concrete implementation) and the other features are so-called concrete features (i.e. features that have concrete implementations). Relationships among features are divided in hierarchical relationships and cross-tree constraints. The hierarchical relationships are:

- **Mandatory**: A parent feature X has a mandatory relationship with a child feature Y when Y has to be present whenever X is present in a given product. For instance, any product of the e-shop product line of Fig. 2 has to have a catalogue of products.
- **Optional**: A parent feature X has an optional relationship with a child feature Y when Y can be present or not whenever X is present in a given product.

For instance, a product of the e-shop product line can optionally have a search feature.
- **Or**: A parent feature X has an or relationship with a set of child features $Y_1, \ldots, Y_n$ when any of the children can be present or not whenever X is present in a given product. For instance, a product of the e-shop product line can optionally have different payment methods such as bank transfer, credit card, paypal or any combination of the three.
- **Alternative**: A parent feature X has an alternative relationship with a set of child features $Y_1, \ldots, Y_n$ when one and only one of the children can be present whenever X is present in a given product. For instance, a product of the e-shop product line can only have high or standard security, but not both.

Furthermore, cross-tree constraints can be used to model restrictions among features. These constraints can be complex constraints on the form of propositional formulas [1]. However, very often we see two kinds of cross-tree constraints:

- **Requires**: A feature X has a requires relationship with a feature Y when Y has to be present whenever X is present in a given product. For instance, any product of the e-shop product line of Fig. 2 has to have high security whenever a credit card is used for payment purposes.
- **Excludes**: A feature X has an excludes relationship with a feature Y when X and Y cannot be present together in a given product.

There is an important aspect that has to be underlined in the history of feature models and it is about one of the areas in which Stefania Gnesi has done a lot of research: formal methods. Feature models were first introduced in a rather informal form. Only around 10 years after their introduction, formal definitions of feature models were introduced. Operational semantics using constraint satisfaction problems [2] or propositional formulas were introduced in 2005 [1]. Later, formal syntax and semantics of feature models were conscientiously developed [6,17]. Feature models were then extended to work with more complex models, analysis and domains. It is remarkable that Stefania Gnesi's team worked in depth to use feature models together with modal transition systems with clean and formal semantics to perform complex analysis in highly (re)configurable systems [7,18,19].

With the formalisation achievement, a new line of research –that was already marginally mentioned in the original FODA report– took the scene: the automated analysis of variability models.

## 3 Automated Analysis of Variability Models

The automated analysis of variability models is about the computer-aided extraction of information from variability models [3]. Figure 3 shows a simplification of the analysis process. First, a variability model is taken as input. Second, the variability model is translated to a given logical representation following several

rules. Third, an off-the-shelf solver is used to perform different analysis operations over the logical representation. And finally, an analysis result is constructed as a final output of the process.

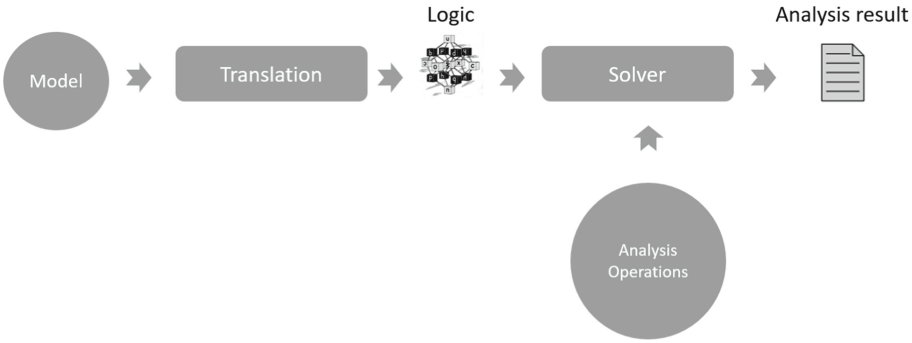


**Fig. 3.** Process of the automated analysis of variability models

An example of an analysis operation can be to count the number of products represented by the feature models. In the model of Fig. 2, the total number of potential products is 20. This operation could be done manually for small feature model examples as the one in Fig. 2 but when dealing with large-scale feature models, this operation become complex and even infeasible in some cases due to the computational complexity required. For instance, it is known that the feature model representing the Linux kernel can have more configurations than there are atoms in the universe!

Another often used operation is "valid configuration" that takes a set of features representing a configuration after which the analysis process determines whether the configuration is valid or not. For instance, the configuration A below is a valid configuration for the feature model of Fig. 2, but the configuration B is not valid because it has a credit card payment but a standard security system.

$$A = \{eShop, Catalogue, Payment, BankTransfer, Security, High\}$$
$$B = \{eShop, Catalogue, Search, Payment, CreditCard, Security, Standard\}$$

Imagine that we include a new cross—tree constraint that states that a catalogue feature excludes any payment method. In such a case, the feature model becomes "void" in the sense that the feature model would represent no product.

Also, imagine that we add a new cross-tree constraint that states that the Security feature excludes Bank Transfer. In such a case, we can still derive products from the feature model. For instance, the product B in the example above. However, the feature Bank Transfer becomes what is known as a "dead feature". A dead feature is a feature that is present in the feature model, but can never be present in any product derived from the feature model.

In 2005, there were two papers [1,2] that emphasised the importance of the automated analysis of feature models and gave a new impetus to the study of modelling and analysis of variability. Independently but complementary, the teams of Batory (University of Texas at Austin) and Benavides (University of Sevilla) worked in proposing automated mechanisms to support analysis operations[1].

Batory proposed the usage of SAT solvers [1], Benavides proposed the usage of CSP solvers [2]. The idea was quite similar, but the solution was a bit different. Batory proposed to translate feature models into propositional formulas and then use SAT solvers to operate over the formulas. SAT solvers are well known to perform fast in most of the cases and have been proved to also work fast in many cases of feature model analysis [14].

Benavides *et al.* proposed to translate feature models into Constraint Satisfaction Problems (CSP) and then use CSP solvers to perform the analysis operations. The novelty with respect to SAT-based analysis was that in CSPs you can also have numerical values and not only Boolean features. Feature models could then be extended with attributes. For example, imagine that we add an integer attribute to the payment feature stating the maximum amount for a given transaction.

When adding attributes to feature models the kind of analysis that can be performed becomes even more complex [11]. For instance, you can try to optimise some attributes so that a line of research was born dealing with optimisation in feature model analysis.

In 2010, an extensive review of the analysis proposed up-to-date was presented [3]. A total of 30 different analysis operations were reported. As well as SAT and CSP solvers, also description logic solvers, BDD solvers and ad-hoc algorithms were reported. In 2010, the conceptual underpinnings of the discipline were settled and the automated analysis of feature models was added to software product line tools in commercial and open-source formats. For instance, Pure::Variants and FeatureIDE are examples of commercial and open-source tools that incorporate feature model analysis among their capabilities.

After that, the decade of the applications was started and is at the point of finishing with this anniversary. More and more applications were found specially in the following areas, as reported in [9]:

- product configuration and derivation: the automated analysis of feature models is used to configure and derive products, for instance, using consistency checking capabilities to avoid compiling or linking incompatible configurations.
- testing and evolution: there is a still active part of the research in software product lines that deals with the problem of testing these kind of systems, which is another area where Stefania Gnesi contributed early on [4,8]. Testing software product lines adds an extra level of difficulty and complexity from

[1] These two works were recently recognised with the Test-of-Time Award and the Most Influential Paper Award by the software product line community.

the traditional testing of a single product. The automated analysis of feature models has been used, for instance, to select representative combinations of features when the testing of all the configurations becomes infeasible.
- reverse engineering: another still active area of research is how to build variability models from existing artefacts. It is quite common that software product lines are built from existing products and artefacts rather than building them from the beginning. One of the ideas is to extract variability models from these assets. The automated analysis of feature models can be used in these scenarios to check the consistency among the assets and the produced models.
- multi-model variability-analysis: it is common that variability is not expressed in a single model or by a single stakeholder and in the same variability modelling language. In this cases, we have to deal with heterogeneous scenarios where the automated analysis has to be adapted
- variability modelling: variability modelling is still being studied because for different scenarios, different variability constructs might be used. In different domains, different adaptations of the variability language constructs have to be defined. The automated analysis has to be adapted as well to these different scenarios.
- variability-intensive systems: the automated analysis of variability models is going beyond software product lines to a wider scope that can be named as "variability-intensive systems". These are systems that are not built following a software product line philosophy specially from a process engineering perspective, but that have to deal with variability. The Linux kernel, the Android ecosystem, and the Eclipse IDE framework are examples of these systems. The automated analysis of variability models is also being used and extended in this kind of new environments.

## 4 Conclusions

Variability modelling and analysis has progressed in the last three decades. One of the points were the discipline progressed faster and better was when formal approaches were considered by the researchers. One of those researchers was Stefania Gnesi and she was an important part of the health of the community. As well as technically, she contributed with community service chairing the Systems and Software Product Line Conference (SPLC) in 2014 in Florence and being active part of the Steering Committee of SPLC in the recent past. The community will lose a very important active researcher with her retirement, but we will be lucky to have both her already established contributions and their fellows that are still active in the field. This book is only a small piece of the immense gratitude that the community owes Stefania. Thank you very much for sharing your time and talent with us and all the best for the future.

**Acknowledgements.** This work has been partially funded by the EU FEDER program, the MINECO project OPHELIA (RTI2018-101204-B-C22); the TASOVA

network (MCIU-AEI TIN2017-90644-REDT); and the Junta de Andalucia METAMORFOSIS project. I would like to give special thanks to Maurice ter Beek for taking care of the book, the ceremony and the gratitude to Stefania. This acknowledgement is extended to all her team.

## References

1. Batory, D.: Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005). https://doi.org/10.1007/11554844_3
2. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Pastor, O., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005). https://doi.org/10.1007/11431855_34
3. Benavidges, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010)
4. Bertolino, A., Gnesi, S.: PLUTO: a test methodology for product families. In: van der Linden, F.J. (ed.) PFE 2003. LNCS, vol. 3014, pp. 181–197. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24667-1_14
5. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. The SEI Series in Software Engineering. Addison-Wesley, Boston and London (2001)
6. Durán, A., Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortés, A.: FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing. Softw. Syst. Model. **16**(4), 1049–1082 (2017)
7. Fantechi, A., Gnesi, S.: Formal modeling for product families engineering. In: Proceedings of the 12th International Software Product Line Conference (SPLC 2008), pp. 193–202. IEEE (2008)
8. Fantechi, A., Gnesi, S., Lami, G., Nesti, E.: A methodology for the derivation and verification of use cases for product lines. In: Nord, R.L. (ed.) SPLC 2004. LNCS, vol. 3154, pp. 255–265. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28630-1_16
9. Galindo, J.A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A.-M., Ruiz-Cortés, A.: Automated analysis of feature models: quo vadis? Computing (2018)
10. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Spencer Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document (1990)
11. Lettner, M., Rodas, J., Galindo, J.A., Benavides, D.: Automated analysis of two-layered feature models with feature attributes. J. Comput. Lang. **51**, 154–172 (2019)
12. Lopez-Herrejon, R.E., Batory, D.: A standard problem for evaluating product-line methodologies. In: Bosch, J. (ed.) GCSE 2001. LNCS, vol. 2186, pp. 10–24. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44800-4_2
13. Douglas McIlroy, M., Buxton, J., Naur, P., Randell, B.: Mass-produced software components. In: Proceedings of the 1st International Conference on Software Engineering, Garmisch Partenkirchen, Germany, pp. 88–98 (1968)
14. Mendonca, M., Wąsowski, A., Czarnecki, K.: SAT-based analysis of feature models is easy. In: Proceedings of the 13th International Software Product Line Conference, pp. 231–240. Carnegie Mellon University (2009)

15. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-28901-1
16. Rodas-Silva, J., Galindo, J.A., García-Gutiérrez, J., Benavides, D.: Selection of software product line implementation components using recommender systems: an application to wordpress. IEEE Access **7**, 69226–69245 (2019)
17. Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2), 456–479 (2007)
18. ter Beek, M.H., Damiani, F., Gnesi, S., Mazzanti, F., Paolini, L.: On the expressiveness of modal transition systems with variability constraints. Sci. Comput. Program. **169**, 1–17 (2019)
19. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: Modelling and analysing variability in product families: model checking of modal transition systems with variability constraints. J. Log. Algebraic Methods Program. **85**(2), 287–315 (2016)