

KEY CONCEPTS – AD HOC

This document is an integral part of the training and presents a theoretical synthesis of the main information for understanding the Ad hoc: Software Product Line, SMarty approach and software inspection.

1. Software Product Line (SPL)

An SPL is defined by Northrop (2002) LPS as set of software systems that share common and variable characteristics, which satisfy the needs of a particular market segment developed from common artifacts.

SPL Engineering establishes the steps, processes and methodologies for using SPL in software development. It has two essential activities: (i) Domain Engineering, a process in which there is the development and evolution of the core of artifacts, (ii) Application Engineering, in which the configuration of a specific product takes place.

Variable features define the variable software, that is the capacity or ability of a system or device be efficiently extended, changed, or custom configured for use in a particular context (Van Gurp et al., 2001).

Variability is described by variation points and variants:

- **Variation Point** A specific location in an artifact where a design decision has not yet been made, that is, it has been postponed;
- **Variant:** Corresponds to a design alternative to solve a given variability.
- **Constraints between Variants:** Defines the relationships between two or more variants so that you can resolve a point of variation or variability.

Variability management is one of the most important activities in controlling an SPL. It includes activities to identify variability, explicitly represent it in software artifacts throughout the lifecycle, and trace dependencies between variability (Pohl et al., 2005). Therefore, the success of an SPL is related to the organization's ability to manage variability and produce customized systems that stand out in the market.

a. Features Diagram

The feature diagram allows you to hierarchically model the common and variable characteristics of all products that can be derived from an SPL, and can thus be understood by all stakeholders in the development process, including the customer.

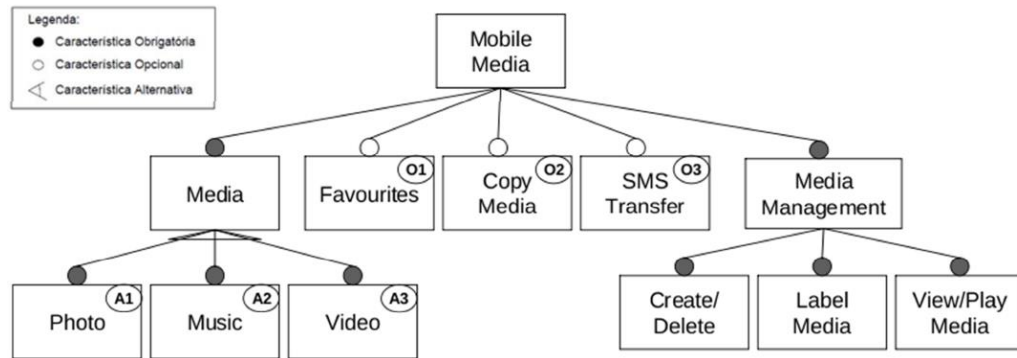
Ad hoc

A feature can be (Benbassat, 2017):

- Mandatory: the feature must be present in any SPL configuration (filled circle);
- Optional: it may or may not be present in an SPL application (empty circle);
- Alternative: only one of the features must be chosen for the SPL configuration (empty arc); and
- OR: it is possible to select all the variant features for the product configuration (filled arc).

In Figure 1, the SPL Mobile Media feature diagram is presented, which is composed of applications that manipulate music, videos and photos for mobile devices. (Geraldi and OliveiraJr, 2017).

Figure 1: SPL Mobile Media feature diagram



Font: (Contieri Junior, 2010)

b. SMarty Approach

The SMarty (Stereotype-based Management of Variability) approach was proposed for the management of SPL variability based on UML models (OliveiraJr et al., 2010). It is in version 5.2 with support for diagrams of: use case, class, activity, component and sequence.

SMarty allows the management of the variability of an SPL in a systematic way through the profile (SMartyProfile), which graphically and visually represents the variability through the UML and a process (SMartyProcess), which defines guidelines to guide the user in identification, representation and tracing variability (OliveiraJr et al., 2010).

The SMartyProfile is a set of stereotypes and meta-attributes (Table 1) that represent the main concepts of variability management: variability, point of variation, variant and variant constraints in UML models for SPL.

Table 1: Stereotypes of the SMarty Approach

Ad hoc

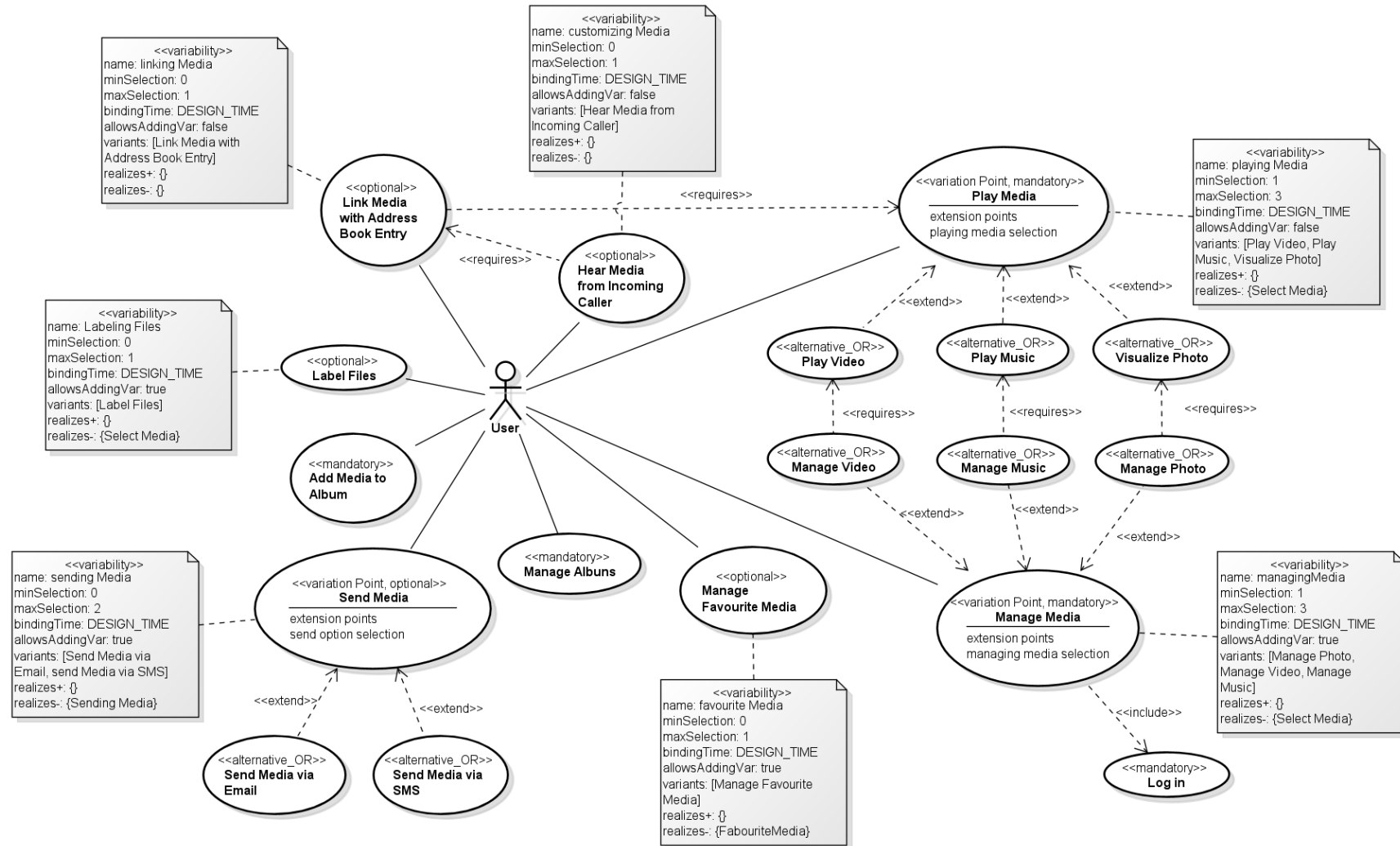
Stereotype	Summary
<<variationPoint>>	Represents the place where variability occurs. A variation point is always associated with one or more variants.
<<mandatory>>	The variant must be present in the configuration of any SPL model.
<<optional>>	The variant may or may not be present in the SPL configuration. Optional variants may or may not be associated with a variation point as well.
<<alternative_or>>	They are always associated with the variation points. At least one of the variants must be chosen to resolve the variation point.
<<alternative_xor>>	They are always associated with the variation points. Only one of the variants must be chosen to solve the variation point.
<<variability>>	Indicates existing variability in a UML model and is only applied to UML grades. Associated with this stereotype has the meta-attributes: name (name of variability), maxSelection and minSelection (maximum and minimum number of variants to be selected to resolve a variation point), bindingTime (defines the moment at which a variability will be resolved) , allowsAddingVar (displays whether there is the possibility of adding new variants after a variability is resolved), variants (set of instances associated with the variability); realize- and realize+ (set of lower and higher-level model variability, respectively, which performs the variability)
<<requires>>	it is a relationship between variants, where the chosen variant requires another related variant;
<<mutex>>	represents the concept of restriction between variants. If a variant is selected, then the dependent variant must not be in the SPL configuration;

Font: adapted de Geraldi and OliveiraJr (2017)

An example of the use of stereotypes of the SMarty approach can be seen in Figure 2. In Figure 3, a part of the SPL Mobile Media use case diagram in Figure 2 is presented. stereotype of the SMarty <<mandatory>> approach. This means that **Manage Albums** is a mandatory feature in any valid configuration for this SPL.

The **Send Media** use case is an optional feature (<<optional>>) for any SPL configuration, so it is up to the user to define whether or not it will be part of the configuration of their specific application. In addition to being an optional feature, **Send Media** is also a variation point (<<variationPoint>>). To solve it, two inclusive variants were defined: **Send Media via SMS** and **Send Media via Email**.

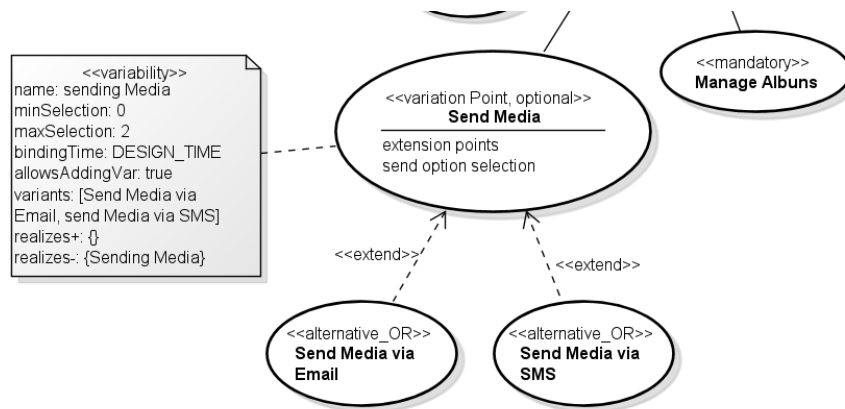
Figure 2: SPLMobile Media use case diagram based on Smarty approach



SMartyCheck

These variants were defined with the `<<alternative_OR>>` stereotype, so at least one of the variants must be chosen to resolve this variation point. Both variants can even be chosen for a specific application. If, instead of `<<alternative_OR>>`, the stereotype was `<<alternative_XOR>>`, only one of the variants could be chosen: sending media by SMS or by email, but not both.

Figure 3: Example of using SMarty approach stereotypes



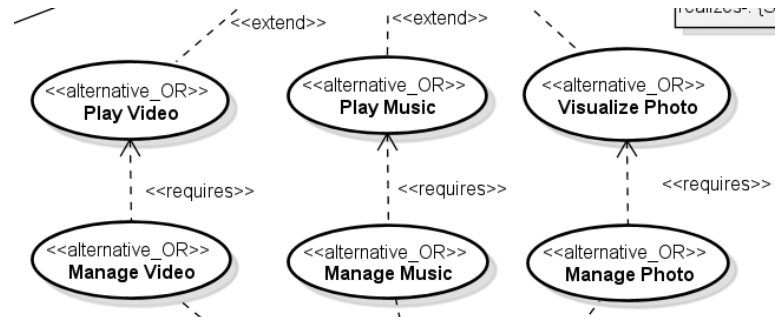
Still in Figure 3, it is possible to observe the UML note contains the variability information in the SMarty approach with the stereotype `<<variability>>`, which represents a variability, and its meta-attributes associated with the Send Media variation point. From the meta-attributes it is possible to extract the following information:

- The name of the variability is sending media. The name is important for tracing variability, therefore, it is how we identify traceabilities that are traceable among the artifacts.
- The minimum selection number of variants is zero, because although variants are of type `<<alternative_OR>>`, which require selection of at least one of the variants, the Send Media element is optional. If this functionality is not present in the product configuration, its variants are also not present. Therefore, **minSelection** must be 0, not 1. **MaxSelection**, on the other hand, has a value of 2, because, if the function is in the configuration, the user can choose one or two variants for a specific application.
- In the meta-attribute variants, the variants related to the use case **Send Media**, **Send Media via SMS** and **Send Media via Email** were defined. This information must be consistent with the number of variants shown in the diagram.
- The realizes- set, for this example, contains the variability realization for the SPL MM class diagram, as the class diagram is less abstract than the use case one. Therefore, the sending media variability must be named in the class diagram in this way to have traceability between the variability.

In Figure 4, the relationship between **Manage Video** and **Play Video** is of type `<<requires>>`, which means that, if the configuration has **Manage Video** functionality, then it must also have **Play Video**. The same goes for **Manage Music** and **Play Music**, **View**

Photo and Manage Photo. If the relationship were of type `<<mutex>>`, then a product that has **Manage Video**, could not also have **Play Video**.

Figure 4: Example of using the stereotype `<<requires>>`



2. Software Inspection

Software quality assurance is a structured approach that involves defect containment, prevention, detection, and removal activities. To this end, verification and validation (V&V) activities help to quantify the quality assurance process by considering that the process was not conducted correctly and that there is a possibility that the product contains defects that must be corrected (Koscianski and Soares, 2007).

Verification is a technique that checks that the software is being developed according to the user's specifications. To this end, software review can be applied to verify the quality of a product, acting as a “filter” to purify the artifacts generated in the software process by discovering errors that can be removed. Its need is allied to two facts: (i) artifacts are generated by people and so, there can be errors; and (ii) errors are hardly detected by those who create the artifact, thus requiring reading by someone else, the reviewer (Pressman and Maxim, 2016).

Software inspection is considered a type of review, which guides the reviewer in reading the artifact through a set of instructions. It is essential “to understand a given representation of the software artifact and compare it to a set of expectations regarding structure, content and quality” (Biffl and Halling, 2003).

It can be applied at the end of all software development steps as soon as an artifact is created, which reduces rework costs, as defects are usually found close to the point where they were inserted (Höhn, 2003) and if they pass. for the next activity they can cost up to ten times more (Zhu ,2016).

To support the reviewer when reading a document in practice, there are reading techniques, which provide a series of instructions for the reader on how to read and what to look for in the artifact (Basili et al., 1996). They provide a general understanding of the inspection activity through the process characteristic of the chosen technique and the defect classes covered by it for the verification of quality attributes. The main ones are: Ad hoc and Checklist-Based Reading and Scenario-Based Reading.

SMartyCheck

a. Defect Detection Ad hoc

The Ad hoc Technique does not provide any kind of instructions, does not have clear procedures or mechanisms for inspectors to correctly read software artifacts individually. In this way, the inspection depends on the inspector's point of view, his knowledge and experience in detecting defects in software artifacts.

However, this technique is one of the most widespread techniques because it is "informal", that is, without bureaucracy or systematic criteria to carry out inspections on specific software artifacts, unlike the techniques for detecting defects based on checklists and perspectives.

Thus, studies indicate that in fact the ad hoc technique does not approach the effectiveness, in most cases, of the defect detection techniques based on checklist and perspectives.

3. References

- Benbassat, F. Evolução Segura de Linhas de Produtos de Software: cenários de extração de features. 67 f. Dissertação (Mestrado) - Curso de Ciência da Computação, UFPE, Recife, 2017.
- Basili, V. R.; Green, S.; Laitenberger, O.; Lanubile, F.; Shull, F.; Sørumgård, S.; Zelkowitz, M. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, v. 1, n. 2, p. 133–164, 1996
- Biffl, Halling. Investigating the defect detection effectiveness and cost benefit of nominal inspection teams. *IEEE Transactions on Software Engineering*, v. 29, n. 5, p. 385–397, 2003.
- Ciolkowski, M.; Differding, C.; Laitenberger, O.; Münch, J. Empirical investigation of perspective-based reading: A replicated experiment. Fraunhofer Institute for Experimental Software Engineering, Germany, 1997.
- Contieri Junior, A.; Aplicação de Métricas em Arquiteturas de Linhas de Produto de Software. 2010. 73 f. Monografia de TCC. Universidade Estadual de Maringá, 2010.
- Geraldi, R. T.; Oliveira Jr, E. Towards Initial Evidence of SMartyCheck for Defect Detection on Product-Line Use Case and Class Diagrams. *Journal of Software*, v. 12, p. 379–392, 2017
- Höhn, E. Técnicas de leitura de especificação de requisitos de software: estudos empíricos e gerência de conhecimento em ambientes acadêmico e industrial. Tese de Doutorado, USP, 2003.
- Lanubile et al. Assessing the impact of active guidance for defect detection: a replicated experiment. *International Symposium on Software Metrics*, 2004, pp. 269–278.
- de Mello, R. M.; Teixeira, E. N.; Schots, M.; Werner, C. M. L.; Travassos, G. H. Verification of Software Product Line Artifacts: a Checklist to Support Feature Model Inspections. *Journal of Universal Computer Science*, v. 20, n. 5, p. 720–745, 2014.
- Oliveira Jr, E.; Gimenes, I. ; Maldonado, J. C. Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science*, v. 16, n. 17, p. 2374–2393, 2010.
- Pressman, R.; Maxim, B. *Software engineering: a practitioner's approach*. 8 ed. Palgrave macmillan, 2016.
- Pohl, K.; Böckle, G.; van Der Linden, F. J. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer Science & Business Media, 2005.
- Northrop, L. M. Sei's software product line tenets. *IEEE Software*, v. 19, n. 4, p. 32–40, 2002.
- Shull, F. J. Developing techniques for using software documents: A series of empirical studies. Tese de Doutorado, University of Maryland at College Park, USA, 1998.
- Travassos, G.; Shull, F.; Fredericks, M.; Basili, V. Detecting defects in object-oriented designs: using reading techniques to increase software quality. *ACM Sigplan Notices*, v. 34, n. 10, p. 47–56, 1999.
- Van Gurp, J.; Bosch, J.; Svahnberg, M. On the notion of variability in software product lines. In: *Proceedings Working IEEE / IFIP Conference on Software Architecture*, 2001, p. 45–54.
- Van der Linden, F. J.; Schmid, K.; Rommes, E. *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media, 2007.
- Zhu, Y. *Software Reading Techniques: Twenty Techniques for More Effective Software Review and Inspection*. Apress, 2016.