On the relationship of concern metrics and requirements maintainability.

ARTICLE *in* INFORMATION AND SOFTWARE TECHNOLOGY · FEBRUARY 2012

Impact Factor: 1.33 · DOI: 10.1016/j.infsof.2011.09.003 · Source: DBLP

5 AUTHORS, INCLUDING:

José María Conejero Manzano

Universidad de Extremadura

42 PUBLICATIONS **184** CITATIONS

SEE PROFILE

Eduardo Figueiredo

Federal University of Minas Gerais

58 PUBLICATIONS **1,296** CITATIONS

SEE PROFILE

Juan Hernández

Universidad de Extremadura

93 PUBLICATIONS **511** CITATIONS

SEE PROFILE

Elena Jurado

Universidad de Extremadura

15 PUBLICATIONS **46** CITATIONS

SEE PROFILE

Information and Software Technology 54 (2012) 212–238

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

On the relationship of concern metrics and requirements maintainability ☆

José M. Conejero [a,*], Eduardo Figueiredo [b], Alessandro Garcia [c], Juan Hernández [a], Elena Jurado [a]

[a] Quercus Software Engineering Group, University of Extremadura. Avda. de la Universidad, s/n, 10071, Spain
[b] Computing Department, Federal University of Minas Gerais, Brazil
[c] Informatics Department, Pontifical Catholic University of Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received 9 July 2010
Received in revised form 25 July 2011
Accepted 12 September 2011
Available online 21 September 2011

Keywords:
Requirements engineering
Product lines
Crosscutting
Concern metrics
Maintainability
Stability

ABSTRACT

Context: Maintainability has become one of the most essential attributes of software quality, as software maintenance has shown to be one of the most costly and time-consuming tasks of software development. Many studies reveal that maintainability is not often a major consideration in requirements and design stages, and software maintenance costs may be reduced by a more controlled design early in the software life cycle. Several problem factors have been identified as harmful for software maintainability, such as lack of upfront consideration of proper modularity choices. In that sense, the presence of crosscutting concerns is one of such modularity anomalies that possibly exert negative effects on software maintainability. However, to the date there is little or no knowledge about how characteristics of crosscutting concerns, observable in early artefacts, are correlated with maintainability.
Objective: In this setting, this paper introduces an empirical analysis where the correlation between crosscutting properties and two ISO/IEC 9126 maintainability attributes, namely changeability and stability, is presented.
Method: This correlation is based on the utilization of a set of concern metrics that allows the quantification of crosscutting, scattering and tangling.
Results: Our study confirms that a change in a crosscutting concern is more difficult to be accomplished and that artefacts addressing crosscutting concerns are found to be less stable later as the system evolves. Moreover, our empirical analysis reveals that crosscutting properties introduce non-syntactic dependencies between software artefacts, thereby decreasing the quality of software in terms of changeability and stability as well. These subtle dependencies cannot be easily detected without the use of concern metrics.
Conclusion: The correlation provides evidence that the presence of certain crosscutting properties negatively affects to changeability and stability. The whole analysis is performed using as target cases three software product lines, where maintainability properties are of upmost importance not only for individual products but also for the core architecture of the product line.

1. Introduction

Maintainability is one of the most essential quality attributes as maintenance tasks consume a high proportion of the total effort spent in the software life cycle [1]. Many studies in the literature reveal that software maintenance is the most costly phase in the software life cycle, currently draining 60–90% of the total cost of software [2–9]. The ISO/IEC 9126 quality model defines maintainability as the capability of the software product to be modified [10]. With software maintenance increasingly becoming a time consuming activity, predicting the system maintainability may serve as an indicator of the cost and endeavour needed during its evolution [1,2,11]. This prediction may also serve to detect opportunities to improve the design modularity upfront.

ISO/IEC 9126 divides maintainability into a number of sub-attributes which may be measured by software metrics, more notably changeability and stability. Changeability is defined as the capability of the software product to enable a specified modification to be implemented [10]. On the other hand, stability is defined as the capability of software products to avoid unexpected ripple effects when modifications are realized [10]. While changeability indicates the ability of a software artifact to be prepared to deal with changes, stability is quantified/observed after a change has been completed.

Research shows evidence that changeability and stability are two of the most critical maintainability factors: poorly managed changes often generate software instabilities and a significant

☆ This paper is a significant extension of a paper published at the TOOLS.09 conference: José Conejero, Eduardo Figueiredo, Alessandro Garcia, Juan Hernández, Elena Jurado, Early crosscutting metrics as predictors of software instability, TOOLS 47 (2009) 136–156.

* Corresponding author. Tel.: +34 927 25 7195; fax: +34 927 25 7202.
E-mail addresses: chemacm@unex.es (J.M. Conejero), figueiredo@dcc.ufmg.br (E. Figueiredo), afgarcia@inf.puc-rio.br (A. Garcia), juanher@unex.es (J. Hernández), elenajur@unex.es (E. Jurado).

0950-5849/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2011.09.003

increase in the cost of software maintenance – up to 75% of the software total ownership costs [1,2,11]. The recent study conducted by Chen and Huang [2] highlights that software maintenance problems are caused by software maintainability not being often a major consideration at an early design stage. A reduction in software maintenance costs could be achieved by a more controlled design early in the software life cycle [1,2,12]. The early identification of potential unstable modules that require high priority attention allows high-quality products to be delivered before the maintenance phase. When this task is postponed to implementation or later, the cost of correction would be much higher [11]. Therefore, changeability and stability analysis should be carried out at early development stages.

A significant impediment to maintenance is the level of interconnection between modules [13,14]. In that sense, crosscutting is a special kind of relationship that generates undesired non-syntactical interactions between modules [15]. In fact, there is growing empirical evidence that the key attributes of software maintainability are often inversely proportional to the presence of crosscutting concerns [16–19]. As an example, it has been observed that crosscutting often leads to increased modularity anomalies [17,19,20] and higher number of introduced faults [16,21,22]. The same observation has been made regarding changeability and stability attributes, often negatively affected by the presence of crosscutting [17,23]. However, most of the systematic studies about crosscutting (e.g. Refs. [16–20,22,24]) concentrate on the analysis of source code, when architectural decisions have already been made. Crosscutting concerns manifest in early development artifacts, such as requirements descriptions [25] and architectural models [26,27], due to their widely-scoped influence in software decompositions. They can be observed in every kind of requirements and design representations, such as use cases and component models [25–28]. Over the last years, aspect-oriented software development (AOSD) [15] has investigated the impact of crosscutting concerns throughout the software lifecycle.

Recent empirical studies of AOSD revealed that, whilst not all types of crosscutting concerns are harmful to design maintainability [17,29], there are certain measurable characteristics of crosscutting concerns that seem to recurrently lead to design instabilities [16,17,19,30]. However, these studies are based on source-code analysis when inferring maintainability attributes after investing efforts in implementations can be expensive and impractical. In addition, concern metrics [31] defined for early design representations are very specific to certain models, such as component-and-connector models [27]. These metrics are overly limited as many crosscutting concerns are visible in certain system representations, but not in others [31]. Last, but not least, the correlation analysis of early crosscutting properties with changeability and stability has been neglected in the literature. In other words, there is little or no knowledge about how characteristics of crosscutting concerns, observable in early artifacts (e.g. use cases), are correlated with changeability and stability.

1.1. Research method

The goal of this paper is to understand how the presence of crosscutting concerns affects changeability and stability of software artifacts at requirements level. More specifically we aim at analysing whether or not certain crosscutting characteristics affect software maintainability. As the problem of crosscutting concerns is usually described in terms of scattering and tangling [25], the following main research question (MRQ) drove our research method:

MRQ: How do scattering, tangling and crosscutting affect the requirements maintainability?

Maintainability can be predicted through empirical analysis [14]. However, as it is observed in Ref. [31], early design anomalies cannot be straightforwardly detected with traditional module-oriented metrics [18,19] because these metrics are restricted to quantify module properties and their direct dependencies. Many modularity anomalies are caused by the way a concern is realized across different modules. We then use a concern metric suite [33] to investigate the above research question. Concern metrics allow the quantification of properties of one or multiple concerns with respect to the underlying modular structure instead of measuring properties of a particular module [31]. To address the main research question, we selected three different Software Product Lines (SPL), called MobileMedia [17], HealthWatcher [19] and SmartHome [32] as our target cases. In this context, the investigated concerns are the features[1] of the SPLs. These systems were selected because the effectiveness of a SPL approach highly depends on how well features are managed throughout both development and maintenance stages [34]: the more independent the assets are, the easier the products are likely to be built [35]. Furthermore, previous research has shown clear evidences that features may crosscut each other, making them rigidly dependent on each other and thus reducing the changeability and stability of SPL assets [20,34–36]. This evidence makes us to raise the following sub-questions (RQ1–RQ3) to address MRQ:

RQ1: *Are the features with the highest degrees of scattering, tangling and crosscutting more difficult to change?* That is, we are interested in investigating the possible correlation between crosscutting properties and changeability of features of a SPL.

RQ2: *Are the features with the highest degrees of scattering, tangling and crosscutting more unstable?* That is, we aim at revealing the possible correlation between crosscutting properties and stability of a SPL.

RQ3: *Is there any relationship between the number of feature dependencies of a SPL and their degree of scattering, tangling and crosscutting? If so, how do feature dependencies affect changeability and stability?* That is, we are interested in observing if the number of feature dependencies is correlated with the degree of feature scattering, tangling and crosscutting, and with the changeability and stability of the SPL.

1.2. Contributions

This paper makes a number of research and practical contributions and significantly extends our previous work [33], where a limited set of concern metrics was defined to assess software modularity. We analyse how the presence of crosscutting concerns affects software maintainability attributes at the requirements level. The analysis is performed through an empirical study where new concern metrics are defined to relate modularity with these attributes, namely changeability, stability and feature dependencies. This relation is tested by a threefold empirical analysis. First, we analyse the ability of the metrics to predict the impact set of a feature change, thus addressing the research question RQ1. The analysis provides an estimation of the impact of a feature change before carrying out the change, showing evidences that there exists a correlation between the degree of scattering and crosscutting, and the changeability of SPL features. This is particular useful not only at early stages of development when the system has not been completely developed but also for deployed systems that require maintenance changes. Second, to address the research question RQ2, we present an analysis of the concern metrics in terms of

[1] Note that the term of concern is used as a synonym of feature.

their predictability power of software stability. The analysis provides evidence that more unstable features are those with higher degree of scattering, tangling and crosscutting. Third, an evaluation of the metrics capability to assess feature dependencies [17] and how these dependencies affect changeability and stability is presented, thus addressing the research question RQ3. This third analysis provides insights that the more dependencies among features of a SPL, the higher the degree of crosscutting of these features and, consequently, affecting negatively changeability and stability of SPLs. Note that our previous work just explored the relation between scattering and crosscutting with stability but not with changeability and feature dependencies. Likewise it did not explore the relation between tangling and stability.

In addition, the above three-dimensional evaluation was performed using three real product lines. We observed that changeability and stability correlate with early concern measures. This result provides evidence that scattering, tangling and crosscutting negatively affect software maintainability at early development phases (the main research question MRQ). The used concern metrics have shown to be able to measure the aforementioned maintainability attributes so that developers do not need to grasp too many concern metrics to predict maintainability effort. The simultaneous analyses of changeability and stability have shown that these attributes: (i) are complementary and very central to support a broader maintainability analysis; (ii) can be potentially predicted with concern metrics. Maintainability analysis typically considers changeability and stability attributes independently [17,23,37]. Based on the experimental study, this research argues that they should be considered together through early concern analysis in order to effectively control maintenance effort. Note that in Ref. [33], the concern metrics defined were only used to assess modularity in a single system (MobileMedia). The new scenario tested in this paper allows, in addition, checking the consistency of the results obtained.

The remainder of this paper is organized as follows. Section 2 summarizes our previous work where a conceptual framework and a crosscutting mining process are presented in order to identify crosscutting situations independently of specific requirements and architecture models. Section 3 presents a motivating example for the application of the modularity analysis at early stages of development and the target product-line cases used throughout the paper. The settings of our empirical study based on concern metrics are presented in Section 3.3. These metrics are used in Section 5 to evaluate modularity in the target product lines. Then, the hypotheses of our study are empirically validated by comparing and correlating modularity of concerns with changeability, stability and feature dependencies in Section 6. Finally, Sections 7 and 8 discuss related work and conclude this paper.

2. Background: characterizing and identifying crosscutting concerns

A concern is anything with interest for a stakeholder in a software system [15]. The term concern is closely related to the term feature (used in the SPL context) in the sense of being a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [38]. In this paper, we consider these two terms as synonyms and concerns are also used referring to features of a software product line (e.g. in our running examples). Concern properties, such as crosscutting and scattering, are often not formally defined. However, their precise definitions are required, for instance, to automate the identification of crosscutting concerns, e.g. in the context of aspect mining, or to investigate the actual impact of crosscutting concerns on maintainability attributes. Analogously, the operational definitions of concern metrics need to be conceived in an unambiguous manner. In that sense, the concern metrics used in this paper (Section 4.2) are based on a previously-defined conceptual framework [25]. The goal of this framework is to support the characterization and identification of crosscutting. Section 2.1 describes the key definitions of this conceptual framework whilst Section 2.2 illustrates how traceability matrices can be used for the identification of crosscutting concerns.

2.1. A conceptual framework for analysing crosscutting dependencies

Our previous work [25] presented a conceptual framework where formal definitions of concern properties, such as scattering, tangling, and crosscutting, were provided. This framework is based on the study of trace dependencies that exist between two different domains. These domains, which are generically called Source and Target, could be, for example, concerns and requirements descriptions, respectively. Alternatively, they could be features and use cases in a different situation. These two domains could even be design modules and programming artifacts when we consider low level elements. We use the term Crosscutting Pattern (Fig. 1) to denote the situation where source and target are related to each other through trace dependencies.

From a mathematical point of view, the Crosscutting Pattern indicates that the Source and Target domains are related to each other through a mapping. This mapping is the trace relationship that exists between the Source and Target domains, and it can be formalized as follows.

According to Fig. 1, there exists a multivalued function f' from the Source to Target domains such that if $f'(s) = t$, then there exists a trace relation between $s \in Source$ and $t \in Target$. Analogously, we can define another multivalued function g' from Target to Source that can be considered as a *special* inverse of f'. If f' is not a surjection, we consider that Target is the range of f'. Obviously, f' and g' can be represented as single-valued functions considering that the codomains are the set of non-empty subsets of Target and Source, respectively.

Let $f:\ \text{Source} \to \mathcal{P}(Target)]$ and $g : Target \to \mathcal{P}\ (Source)$ be two *functions defined by*:

$$\forall s \in Source,\ f(s) = \{t \in Target : f'(s) = t\}$$
$$\forall t \in Target,\ g(t) = \{s \in Source : g'(t) = s\}$$

The concepts of scattering, tangling and crosscutting are defined as specific cases of these functions.

Definition 1. (Scattering) We say that an element $s \in Source$ is scattered if $card(f(s)) > 1$, where *card (f (s))* refers to cardinality of *f (s)*. In other words, *scattering occurs when, in a mapping between source and target, a source element is related to multiple target elements.*

Definition 2. (Tangling) We say that an element $t \in Target$ is tangled if $card(g(t)) > 1$. Hence, *tangling occurs when, in a mapping between source and target, a target element is related to multiple source elements.*

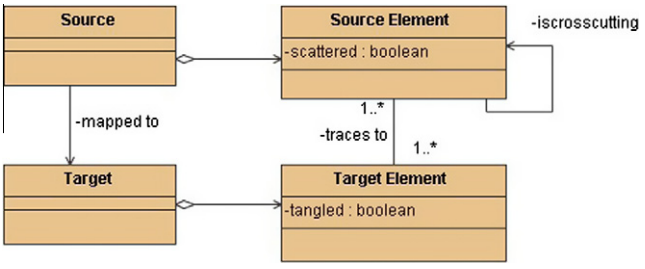

Fig. 1. The crosscutting pattern.

There is a specific combination of scattering and tangling which we call crosscutting.

Definition 3. (Crosscutting) Let $s1, s2 \in Source, s1 \neq s2$, we say that *s1* crosscuts *s2* if $card(f(s1)) > 1$ and $\exists t \in f(s1): s2 \in g(t)$. In other words, *crosscutting occurs when, in a mapping between source and target, a source element is scattered over target elements and, in at least one of these target elements, source elements are tangled.*

According to the previous definitions, the following result is a direct consequence.

Lemma 1. *Let* $s1,s2 \in Source, s1 \neq s2$, *then s1 crosscuts s2 iff* $card(f(s1)) > 1$ *and* $f(s1) \cap f(s2) \neq \emptyset$.

2.2. Identification of crosscutting relations

We defined in previous work [25] a special kind of traceability matrix that we called *dependency matrix* to represent the function *f*. An example of dependency matrix with five source and six target elements is shown in Table 1. In the rows, we have the source elements, and in the columns, we have the target elements. A '1' in a cell denotes that the target element of the corresponding column contributes to or addresses the source element of the corresponding row (in Table 1, t[1] and t[4] contribute to the functionality of s[1]).

Two different matrices called scattering matrix (Table 2) and tangling matrix (Table 3) are derived from the dependency matrix. These matrices show the scattered and tangled elements in a system, respectively:

- In the scattering matrix, a row contains only dependency relations from source to target elements if the source element in a row is scattered (mapped onto multiple target elements); otherwise the row contains just zero values (no scattering).
- In the tangling matrix, a row contains only dependency relations from target to source elements if the target element in a row is tangled (mapped onto multiple source elements); otherwise the row contains just zero values (no tangling).

Table 3
Tangling matrix for Table 1.

tangling matrix

		Source				
		s[1]	s[2]	s[3]	s[4]	s[5]
Target	t[1]	1	1	1	0	0
	t[2]	0	0	0	0	0
	t[3]	0	1	0	1	0
	t[4]	1	0	0	0	1
	t[5]	0	1	0	0	1
	t[6]	0	0	0	0	0

The crosscutting product matrix (Table 4) is obtained through the multiplication of scattering and tangling matrices. The crosscutting product matrix shows the quantity of crosscutting relations and is used to derive the final crosscutting matrix. Tables 4 and 5 show, respectively, the crosscutting product and crosscutting matrices derived from Tables 2 and 3. In the crosscutting matrix, each cell denotes the occurrence of crosscutting; it abstracts from the quantity of crosscutting. A crosscutting matrix *ccm* can be derived from a crosscutting product matrix *ccpm* using a simple conversion: $ccm[i][k] = if\ (ccpm[i][k] > 0) \wedge (i \neq k)\ then\ 1\ else\ 0$. More details about the conceptual framework and the matrix operations can be found in Ref. [25].

The conceptual framework has been also extended in order to be automatically applied to software requirements. In particular, it was extended in Ref. [39] with syntactical and dependency-based analyses to automatically obtain the mappings between source and target elements. In other words, the extension described in Ref. [39] allows the automation of the process of building the dependency matrix which represents the starting point to perform crosscutting concern identification. The reader may obtain further details of this extension in Ref. [39].

3. The motivating cases

The conceptual framework summarized in Section 2 allows developers to identify and quantify properties of crosscutting features. For instance, the quantification of scattering, tangling or

Table 1
Example dependency matrix.

dependency matrix

		Target					
		t[1]	t[2]	t[3]	t[4]	t[5]	t[6]
Source	s[1]	1	0	0	1	0	0
	s[2]	1	0	1	0	1	1
	s[3]	1	0	0	0	0	0
	s[4]	0	1	1	0	0	0
	s[5]	0	0	0	1	1	0

Table 2
Scattering matrix for Table 1.

scattering matrix

		Target					
		t[1]	t[2]	t[3]	t[4]	t[5]	t[6]
Source	s[1]	1	0	0	1	0	0
	s[2]	1	0	1	0	1	1
	s[3]	0	0	0	0	0	0
	s[4]	0	1	1	0	0	0
	s[5]	0	0	0	1	1	0

Table 4
Crosscutting product matrix for dependency matrix in Table 1.

crosscutting product matrix

		Source				
		s[1]	s[2]	s[3]	s[4]	s[5]
Source	s 1]	2	1	1	0	1
	s[2]	1	3	1	1	1
	s[3]	0	0	0	0	0
	s[4]	0	1	0	1	0
	s[5]	1	1	0	0	2

Table 5
Crosscutting matrix for dependency matrix in Table 1.

crosscutting matrix

		Source				
		s[1]	s[2]	s[3]	s[4]	s[5]
Source	s[1]	0	1	1	0	1
	s[2]	1	0	1	1	1
	s[3]	0	0	0	0	0
	s[4]	0	1	0	0	0
	s[5]	1	1	0	0	0

crosscutting may help to decide whether a certain feature should be refactored to improve its modularity. Nevertheless, choosing the right feature to refactor is usually not a trivial task since it may have important implications for other quality attributes of the software system. In this setting, the empirical study presented in this paper comes to provide information regarding to the relation between modularity properties and maintainability quality attributes. This information may drive the engineer in taking such decisions. This section shows an example that motivates the need for this empirical analysis and presents the three Software Product Lines (SPL) used in our study, namely MobileMedia [17], HealthWatcher [19] and SmartHome [32]. We selected these three SPLs to test our hypothesis for several reasons: (i) they are publicly available and were not designed in an ad hoc manner just for the sake of our empirical analysis; (ii) these systems have been used in several studies with different analysis purposes [17,19,29,37,43,40,41]; (iii) they have different sizes ranging from 3 KLOC (MobileMedia) to 17 KLOC (SmartHome) and thus providing first evidences about the scalability of our approach; (iv) they are representative application examples for different domains, namely mobile devices, web applications and desktop applications. For simplicity, we choose MobileMedia to be systematically described and used throughout the paper. The analyses of HealthWatcher and SmartHome are summarized and contrasted to check whether and when the results obtained for all the systems are consistent or not.

3.1. The MobileMedia system

MobileMedia is a product line system built to allow the user of a mobile device to perform different operations, such as visualizing photos, playing music or videos and sending photos via SMS. It has about 3 KLOC. MobileMedia encompasses eight designed and implemented subsequent releases (from 0 to 7) that support the analysis of different maintainability facets, such as changeability and stability. For instance, release 0 implements the original system with just the functionality of viewing photos and organizing them into albums. Table 6 summarizes the changes made in each MobileMedia release (see Ref. [17] for more detail). The scenarios cover heterogeneous concerns raging from mandatory, optional, and alternative features, as well as non-functional concerns. Table 6 also presents which types of change each release encompassed. The purpose of these changes is to exercise the implementation of the feature boundaries and, so, assess the changeability and stability of the product line requirements.

Note that some non-functional concerns (NFC) are also explicitly considered as concerns of the system (e.g. Persistence and Error Handling). Table 7 shows the concerns used in the analysis and the releases which include these concerns. The reason for choosing this application for our analyses is threefold. First, as MobileMedia is a software product line, maintainability is of upmost importance; instabilities and changes affect negatively not only the software product line architecture but also all the instantiated products. Second, the software architecture and the requirements had all-encompassing documentation; e.g. the description of all the use cases were made available as well as a complete specification of all component interfaces. Third, the architectural components were independently defined and provided by experienced developers, rather than ourselves.

In this section, a particular release (release 3) of the MobileMedia is used to motivate the empirical analysis presented in subsequent sections. This release includes the functionality to manage albums and photos and some other optional features like sort photos by frequency and set favourite photos. This release was selected because it includes some variable features and non-functional concerns. Nevertheless, the release is simple enough to not complicate the explanation of the process; Section 5 shows the whole modularity analysis using all MobileMedia releases. Fig. 2 shows a simplified feature model for release 3 of the MobileMedia product line. Since its design has been used in previous analyses, we utilize the same feature model used by the original authors [17]. Note that variability between products is mainly concentrated in the possibility of sorting and setting the favourite photos. Moreover some non-functional concerns involved in the system were identified, namely Persistence and Error Handling. Based on the requirements of the system for

Table 7
MobileMedia concerns and releases where are included.

Concern	Releases	Concern	Releases
Album	r0 - r7,	Copy	r4 - r7
Photo	r0 - r7,	SMS	r5 - r7
Label	r0 - r7,	Music	r6, r7
Persistence	r0 - r7,	Media	r6, r7
Error Handling	r1 - r7	Video	r7
Sorting	r2 - r7	Capture	r7
Favourites	r3 - r7		

Table 6
Different releases of MobileMedia.

Release	Description	Type of Changes
r0	MobileMedia basic functionality.	none
r1	Exception handling included.	Inclusion of non-functional requirement
r2	New feature added to count the number of times a photo has been viewed and sorting photos by highest viewing frequency. New feature added to edit the photo's label	Inclusion of optional and mandatory features
r3	New feature added to allow users to specify and view their favorite photos	Inclusion of an optional feature
r4	New feature added to allow users to keep multiple copies of photos in different albums	Inclusion of an optional feature
r5	New feature added to send and receive photos via SMS	Inclusion of an optional feature
r6	New feature added to store, play, and organize music. The management of photo (e.g. create, delete and labeling) was turned into an alternative feature. All alternative features (e.g. sorting, favorites, and copy) were also provided for music.	Changing one mandatory feature into two alternatives
r7	New feature added to manage videos.	Inclusion of an alternative feature

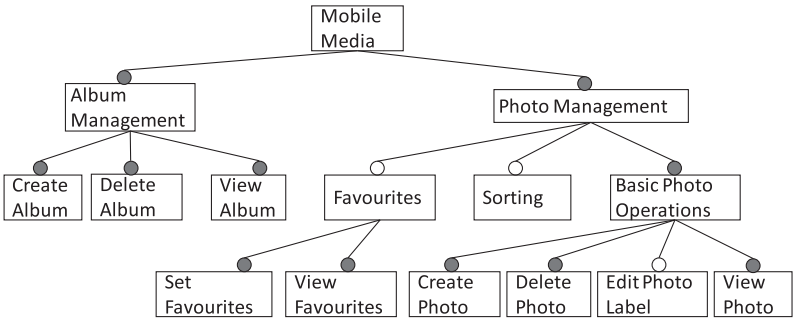

Fig. 2. Features model for MobileMedia.

release 3, a use case diagram was built where these requirements were modelled (see Fig. 3). Whilst both features and non-functional concerns are considered as source elements, the use cases are considered as the target elements of the crosscutting pattern introduced in Section 2.1.

Based on the selected source and target elements and on the process of identifying trace relations between these elements described in Ref. [39], the dependency matrix shown in Table 8 is obtained. As it may be observed, the dependency matrix shows the features and NFCs addressed by each use case. Then, using the matrix operations summarized in Section 2.2, the crosscutting product and the crosscutting matrices are automatically derived (shown in Tables 9 and 10, respectively). Zeros are omitted in these tables for simplification purpose.

Table 10 confirms the common intuition: the NFCs *Persistence* and *Error Handling* are the source elements that crosscut more features. This table also shows how mandatory features (Album and Photo) may be crosscut by variable features (e.g. Label or Sorting) and vice versa, even at the requirements level. Thus, these tables provide important information regarding modularity. However, the tables may be also used to infer information regarding other software quality attributes. As an example, a change scenario may be proposed for the MobileMedia system. For instance, consider that the way Persistence is implemented is changed. Photos selected by the user are updated to a remote site in order to share them with other users whilst non-selected ones are stored in a memory card. That change implies also modifications in artifacts implementing other features. Observe in Table 8 that Persistence is addressed by almost all use cases of this release. This is mainly due to the include relations between some of these use cases and

Table 8
Dependency matrix for the MobileMedia.

		Use cases															
		Add Album	Delete Album	Add Photo	Delete Photo	View Photo	View Album	Provide Label	Store Data	Remove Data	Retrieve Data	Edit Label	Count Photo	View Sorted Photos	Set Favourite	View Favourites
Features	Album	1	1				1									
	Photo			1	1	1										
	Label	1		1				1				1				
	Sorting					1							1	1		
	Favourites														1	1
NFCs	Persistence	1	1	1	1	1	1	1	1	1	1		1		1	
	Error Handling	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

the *Store Data* or *Retrieve Data* use cases (which are entirely dedicated to the Persistence functionality). Then, all use cases that include *Store Data* or *Retrieve Data* must be aware of the change and must be modified accordingly. These changes are due to the existence of dependency relations between the different features. In that sense, crosscutting is a special kind of relation [35] that introduces new dependencies between features. In particular, it may be observed (Table 10) how the Persistence non-functional concern is crosscutting the rest of the features of this release. Obviously, a change in a feature with a lower degree of crosscutting could be easier to accomplish since the set of impacted elements

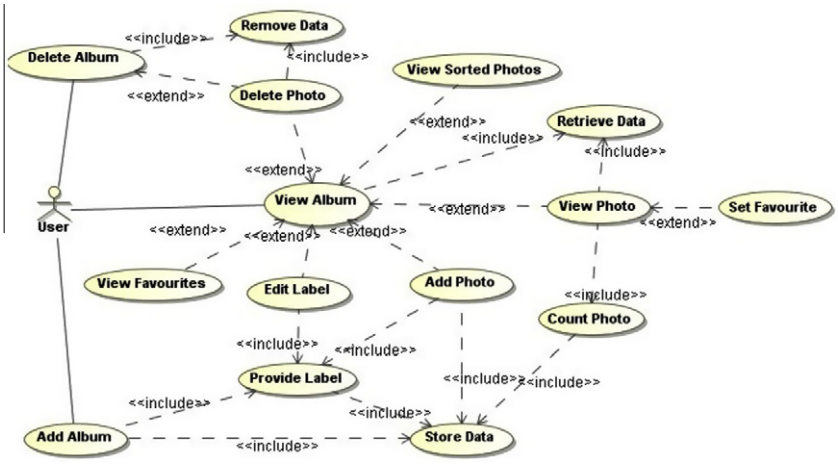

Fig. 3. Use case diagram for release 3.

Table 9
Crosscutting product matrix for MobileMedia.

		Features					NFCs		
		Album	Photo	Label	Sorting	Favourites	Persistence	Error Handling	
Features	Album	3		1			3	3	M
	Photo		3	1	1		3	3	M
	Label	1	1	4			3	4	V
	Sorting		1		3		2	3	V
	Favourites					2	1	2	V
NFCs	Persistence	3	3	3	2	1	12	12	
	Error Handling	3	3	4	3	2	12	15	

Table 10
Crosscutting matrix for MobileMedia.

		Features					NFCs		
		Album	Photo	Label	Sorting	Favourites	Persistence	Error Handling	
Features	Album			1			1	1	M
	Photo			1	1		1	1	M
	Label	1	1				1	1	V
	Sorting		1				1	1	V
	Favourites						1	1	V
NFCs	Persistence	1	1	1	1	1		1	
	Error Handling	1	1	1	1	1	1		

by the change would be smaller. As an example, a change in the Favourites feature will be addressed just by changing Set Favourite and View Favourites use cases (see Fig. 3).

The empirical analysis presented in this paper shows evidence of how our concern metrics (defined in Ref. [33]) may be used to provide such information and assist the developer in taking according decisions. In particular, we show how the information provided by our metrics may be used to select the most unstable features or those with the biggest impact on the system due to dependencies and, thus, more difficult to change.

3.2. The HealthWatcher system

The second system used in our analysis is called HealthWatcher. HealthWatcher is a typical Web-based program family that allows citizen to register complaints regarding health issues [19]. The system has around 4 KLOC and it has been developed as a product line in different releases. The first Health Watcher release of the Java implementation was deployed in March 2001. Since then, a number of incremental and perfective changes have been addressed in posterior Health Watcher releases. These releases allow us to observe typical types of changes in such an application domain. In particular, for the purpose of our analysis we have considered the requirements of five different releases of the product line. These releases are summarized in Table 11. As an example release 0 contains the core system whilst release 1 represents the core system with the functionality of sorting complaints by most popular or most frequent. The different features of the system and the releases where they were included are described in Table 12. Note, again, that we have used the concerns used in previous analyses at later stages, e.g. at architectural level [19]. The reasons for selecting this system in our study are the same described for the MobileMedia, namely: (i) the system is a product line, (ii) the software artifacts of the system were all made available with whole documentation from the requirements stage to the final implementation one; and (iii) the software artifacts were independently defined and provided by experienced developers, allowing future analyses at different development abstraction levels.

Table 11
Different releases of HealthWatcher.

Release	Description	Type of changes
r0	HealthWatcher core	none
r1	Feature added to count the number of times a complaint has been viewed and sorting them by frequency.	Inclusion of optional feature
r2	Allow citizens to geolocalize complaints origin when they create them	Inclusion of optional feature
r3	Allow citizens to login by using digital signature	Inclusion of optional feature
r4	Allow citizens to store and manage their complaints	Inclusion of mandatory feature

Table 12
HealthWatcher concerns and releases where are included.

Concern	Releases	Concern	Releases
QueryInformation	r0 – r4	Standards	r0 – r4
RegisterComplaint	r0 – r4	Hardware&Software	r0 – r4
RegisterTables	r0 – r4	Distribution	r0 – r4
UpdateComplaint	r0 – r4	UserInterface	r0 – r4
RegisterNewEmployee	r0 – r4	OperationalEnvironments	r0 – r4
UpdateEmployee	r0 – r4	Persistence	r0 – r4
UpdateHealthUnit	r0 – r4	Concurrency	r0 – r4
ChangeLoggedEmployee	r0 – r4	Performance	r0 – r4
ResponseTime	r0 – r4	ErrorHandling	r0 – r4
Encryption	r0 – r4	ViewComplaints	r0 – r4
Compatibility	r0 – r4	Popularcomplaints	r1 – r4
Access-Control	r0 – r4	Geolocalization	r2 – r4
Usability	r0 – r4	DigitalSignature	r3, r4
Availability	r0 – r4	ClientComplaints	r4

3.3. The SmartHome system

The last product line analyzed in our study was developed by industry partners of the AMPLE European project [42]. This product line is taken from the domain of the Building Technologies (BT) division at Siemens [32] and allows simulating the control of different devices of a smart home, including windows, heating, air conditioning, blinds, alarms, doors, and so forth. We selected this system as a wide range of the system artifacts are publicly available at the AMPLE website [42], e.g. system descriptions, feature models, and architecture design. Moreover, the system is from a different domain and bigger than MobileMedia and HealthWatcher, thereby allowing us to evaluate the generality and scalability of concern-driven analyses. The SmartHome system has around 17 KLOC [32]. The feature model of the product line has been built by using the SPLOT tool and it is stored and publicly available at its repository.[2] This feature model would allow the generation of around 382.205 K different products. From this huge amount of possible products we selected three releases (product instantiations), detailed in Table 13. We have used an additive strategy to select

[2] Software Product Line Online Tool: http://www.splot-research.org/.

Table 13
Different releases of SmartHome.

Release	Description	Type of changes
r0	SmartHome core (Heating Management, Windows Management, Lights Management, Presence Simulator, Fire Control, Authentication, User Notifications) + Door Lock + Security	none
r1	r0 + Blinds Management + Gas Detection + Water detection + Air conditioning control	Inclusion of optional features
r2	r1 + Audio Management + Dimming Lights + Phone Call notifications + Intruse Detection + CardReader as Authentication method	Inclusion of optional features

the three releases so that the first release contains a set of core features and the other ones just add features to the former. This strategy allows us to analyze the stability of the product line to accomplish changes. Table 14 details the concerns (features) and the releases in which they are involved. Finally, although the system has been previously used in existing analyses [40,41], to the best of our knowledge, an analysis of modularity at the requirements level for this system has been neglected in the literature. In that sense, requirement models for the system have been built by using use cases notation and the use case diagrams for the three selected releases are considered in our analyses.

4. Empirical study settings

This section shows the settings of our experimental study based on the framework presented in Ref. [45] which consists of three main steps: (i) the definition of a set of experimental goals and hypotheses; (ii) the definition of appropriate measures; and (iii) the experimental validation of these measures testing the established hypotheses. Nevertheless, observe that the main purpose of this paper is the empirical analysis of the hypotheses also showing the utility of the metrics presented in Ref. [33].

4.1. Experimental goals and hypotheses

The goal of our experiment is to analyse the relation between modularity properties and maintainability attributes at the requirements level. To address this goal, we establish the following research question previously described in the introduction:

- **MRQ:** *How do scattering, tangling and crosscutting affect requirements maintainability?*

This question is refined into three different research questions:

- **RQ1**: *Are the features with the higher degree of scattering, tangling and crosscutting more difficult to change?*
- **RQ2**: *Are the features with the higher degree of scattering, tangling and crosscutting more unstable?*
- **RQ3**: *Is there any relationship between the number of feature dependencies of a SPL and their degree of scattering, tangling and crosscutting? If so, how do feature dependencies affect changeability and stability?*

Based on the experimental goals and the research questions, the following hypotheses are derived that will be tested in order to confirm or refute our assumptions:

Hypothesis 1. The higher the degree of scattering, tangling and crosscutting for a feature, the more difficult to change the feature is.

Hypothesis 2. The higher the degree of scattering, tangling and crosscutting for a feature, the more unstable the feature is.

Hypothesis 3.1. The higher the degree of scattering, tangling and crosscutting for a feature, the more dependencies the feature has.

Hypothesis 3.2. The more dependencies a feature has, the more difficult to change and unstable the feature is.

4.2. Definition of measures

The evaluation of the hypotheses requires the definition of appropriate measures to assess the attributes that are relevant to such hypotheses, namely scattering, tangling and crosscutting. As explained in Ref. [44], these attributes must be formally defined and characterized in order to be measured. The formal characterization of these attributes was summarized in Section 2.1. On the other hand, the hypotheses relate the aforementioned attributes with: (i) number of links or dependencies (for changeability); (ii) number of changes (for stability); (iii) software artifacts shared (for feature dependencies). As suggested in Ref. [44] these attributes are software characteristics directly measurable so that they do not need to be formally characterized.

Software metrics need to be defined to quantify these attributes. In this study, a set of modularity metrics previously defined [33] has been used in combination with some new metrics defined in this paper. All the metrics used in our analysis are presented in Table 15. The upper part of this table shows the metrics defined in our previous framework whilst the new metrics defined in this paper are presented in the lower part. These new metrics are deeply explained in the empirical analyses where they are used. The

Table 14
SmartHome concerns and releases where are included.

Concern	Releases	Concern	Releases
Temperature Control	r0 – r2	User Notifications	r0 – r2
Windows Management	r0 – r2	Access to Phyisical KNX Devices	r0 – r2
Lights Management	r0 – r2	Blinds Management	r1, r2
Presence Simulation	r0 – r2	Floods Detection	r1, r2
Fire Control	r0 – r2	Gas Detection	r1, r2
Door Lock	r0 – r2	Air Conditioning	r1, r2
Authenticaton	r0 – r2	Audio Management	r2
Security	r0 – r2	Intruse Detection	r2

Table 15
Summary of the concern metrics used to perform our experiment.

	Metric	Definition	Relation with matrices	Calculation	Hypotheses
Defined in Conceptual Framework [33]	*Nscattering* (s_k)	Number of target elements addressing source element s_k	Addition of the values of cells in row *k* in dependency matrix (dm)	$= \sum_{j=1}^{\lvert T\rvert} dm_{kj}$	1, 2, 3
	Degree of scattering (s_k)	Normalization of Nscattering (s_k) between 0 and 1		$= \begin{cases} \frac{Nscattering\ (s_k)}{\lvert T\rvert} & if\ \ Nscattering\ (s_k) > 1 \\ 0 & if\ \ Nscattering\ (s_k) = 1 \end{cases}$	1, 2, 3
	Ntangling (t_k)	Number of source elements addressed by target element t_k	Addition of the values of cells in column *k* in dependency matrix (dm)	$= \sum_{i=1}^{\lvert S\rvert} dm_{ik}$	1, 2, 3
	Degree of tangling (t_k)	Normalization of Ntangling (t_k) between 0 and 1		$= \begin{cases} \frac{Ntangling\ (t_k)}{\lvert S\rvert} & if\ \ Ntangling\ (t_k) > 1 \\ 0 & if\ \ Ntangling\ (t_k) = 1 \end{cases}$	1, 2, 3
	Crosscutpoints (s_k)	Number of target elements where the source element s_k crosscuts to other source elements	Diagonal cell of row *k* in the crosscutting product matrix (ccpm)	$= ccpm_{kk}$	1, 2, 3
	Ncrosscut (s_k)	Number of source elements crosscut by the source element s_k	Addition of the values of cells in row *k* in the crosscutting matrix (ccm)	$= \sum_{i=1}^{\lvert S\rvert} ccm_{ki}$	1, 2, 3
	Degree of crosscutting (s_k)	Addition of the two last metrics normalized between 0 and 1		$= \frac{Crosscutpoints(s_k) + Ncrosscut(s_k)}{\lvert S\rvert + \lvert T\rvert}$	1, 2, 3
New metrics	*Impact Set* (s_k)	Addition of the number of trace links that must be changed and the number of trace links that must be preserved due to a change in s_k	Number of 1 values of row *k* and the number of 1 values in the columns where there are 1's in row *k* of the dependency matrix (dm)	$= \sum_{i=1}^{\lvert T\rvert} \left(\mathrm{dm_{ki}} \left(\sum_{j=1}^{\lvert S\rvert} \mathrm{dm_{ij}} \right) \right)$	1, 4
	Concern degree of tangling (s_k)	Addition of the Degree of tangling metric of each use case that addresses the concern s_k		$= \sum_{i=1}^{\lvert T\rvert} \left(\text{Degree of tangling } (t_i) \right) / f'(s_k) = t_i$	1, 4
	Use Case-level Interlacing Between Concerns (s_k, s_t)	Number of use cases shared by the implementation of two concerns s_k *and* s_t	Cell (*k*,*t*) of the Concern Interlacing Matrix (cim), which is calculated as the product of dependency matrix by its transpose	$= cim_{kt}$	3, 4
	Concern Interlacing (s_k)	Total number of feature dependencies of a particular feature	Addition of the cells in the row *k* of the Concern Interlacing Matrix (cim) minus the diagonal cell of row *k*.	$= \left(\sum_{i=1}^{\lvert T\rvert} cim_{ki} \right) - cim_{kk}$	3, 4

information shown in the table for each metric is: (i) its name, (ii) a brief description, (iii) its relation with the conceptual framework and traceability matrices summarized in Section 2, (iv) the formula used to compute it, and (v) the hypotheses that the metric is defined for testing. All metrics presented in Table 15 are based on the relations between source and target domains represented by the crosscutting pattern and, thus, we should point out that they may be automatically calculated by the values obtained in our dependency matrix (see Section 2.2) or the other matrices derived from it. Although these metrics are used here at early stages of development, they are not tied to any specific development artifact. As the set of metrics for quantifying the modularity properties was previously defined in Ref. [33] and the main purpose of this paper is the empirical validation described in next step, the theoretical validation of the metrics is not repeated here (see Ref. [33]).

4.3. Empirical validation of the measurement

The empirical validation of measures aims at validating the experimental hypotheses involving the attributes quantified by the measures. Moreover, this validation shows the usefulness of a measure in terms of other external quality attributes of practical interest [44]. Thus, the main purpose of this step is twofold: testing the hypotheses presented in Section 4.1 and showing the utility of our metrics for predicting changeability and stability. To perform this evaluation, first an empirical modularity analysis is presented that allows the quantification of the degree of crosscutting at the requirements level (Section 5). Second, modularity attributes are empirically compared with other software quality attributes showing the utility of the modularity analysis (Section 5). The latter is illustrated by correlating the degree of crosscutting of a feature with different maintainability attributes. This correlation allows anticipating important decisions regarding maintainability, e.g. what the impact of a change is or what the more unstable parts of the system are.

5. Analysis of modularity

In order to investigate their predictive power, we first use the measures described in Section 4.2 to assess the modularity of the three product lines used as our target cases. We focus on quantifying the crosscutting properties of the different features of these systems. In that sense, we compute the *Degree of scattering* and *Degree of crosscutting* metrics for the features and the *Degree of tangling* for the use cases. As a result of the analysis: (i) the features with poor modularity, i.e. those with higher degree of scattering and crosscutting, are obtained; and (ii) use cases addressing more features, i.e. with more tangled concerns, are identified. Moreover, the analysis is used to extract important conclusions, such as the need for a specific metric for crosscutting aiming to help developers to identify some crosscutting situations that scattering and tangling metrics do not capture. Second, once modularity has been assessed, it is compared with different maintainability attributes in Section 6. Then, this section computes all the values needed for the empirical comparison between modularity and maintainability.

5.1. Metrics computation process

As discussed in previous sections, MobileMedia is used to deeply explain the empirical process followed to perform our analysis. MobileMedia has evolved to 8 successive releases by adding different concerns to the product line. In order to compute the different metrics used to assess modularity in the system, the dependency matrix (see Section 2.2) for each release is built using concerns and use cases as source and target domains, respectively. Thus, each matrix shows the use cases contributing to the different concerns. Then, all metrics were automatically calculated using as input the dependency matrix. Based on the dependency matrix, we derived the rest of matrices as presented in Section 2.2 (Scattering, Tangling, Crosscutting Product, and Crosscutting Matrices). The dependency matrix for the MobileMedia system in release 7 is shown in Table 16. This release includes all features of the system.

Although our original dependency matrix is binary, in this case a non-binary matrix has been used (Table 16) in order to allow the calculation of metrics which utilize a granularity level different from use cases. This means that a cell presents the number of control flows or steps of the use cases addressing a particular concern. For instance, we can see that the View Album use case has 3 and 1 control flows addressing the Album and Label concerns, respectively. Thus, in order to be able to identify the control flows related to each concern, we have complemented the process described in Section 2.2 with a mapping technique [45]. That is, the control flows or steps in the use case descriptions are mapped when they contribute to the functionality of a particular concern. The concern mapping technique was also applied in Ref. [17], where the authors used it at the source code level. The same process explained for the MobileMedia system is applied to HealthWatcher and SmartHome by building a dependency matrix for each of its five and three releases, respectively. Based on these dependency matrices, we automatically computed the metrics described in Section 4.2 for each release. We have followed existing recommendations in order to avoid and mitigate the introduction of mistakes in the concern mapping process [45–47].

5.2. Metric results

Fig. 4 shows three charts where the average of *Degree of scattering*, *Degree of crosscutting and Degree of tangling* metrics for the 8 releases of the MobileMedia system are represented. By using these charts, we can intuitively see what the features with a higher degree of scattering and crosscutting are. As an example, the concerns with the higher values for both metrics are Persistence and Error Handling. Likewise, the use cases with more tangled concerns can be also easily identified in Fig. 4b. The results of all the metrics used in our analysis for the three product lines (including all their releases) are available in our website [48]. We have also performed a pairwise systematic comparison of the metrics and the key results of our analysis are discussed in the following subsection. We focus on the most interesting findings in this paper. For further details and discussion, the reader can refer to the report in the study's website [48].

The main goal of the measures shown in this section is to analyse the modularity of the system in order to compare it with maintainability attributes, shown in next section. However, by means of this evaluation, an important conclusion about the used metrics can also be extracted. This conclusion was the stringent need for using the *Degree of crosscutting* metric while quantifying concern attributes. This metric is an original contribution of our metrics suite. It is a special combination of scattering and tangling metrics calculated using the *Crosscutpoints* and *Ncrosscut* metrics (see Section 4.2). Fig. 5 shows the *Degree of scattering* and *Degree of tangling* metrics for releases 0 and 1. Note that in these releases, the Album concern presents the same value for *Degree of scattering*. However, the *Degree of crosscutting* metric for this concern is higher in release 1 than in release 0 (see Fig. 6). This is due to the tangling of the use cases where the Album concern is addressed (see in Fig. 5b). Accordingly, we observed that the Album concern is worse modularized in release 1 than in release 0. There are other examples, such as Persistence or Photo. Note that this situation could not be discovered using only the *Degree of scattering* metric. Although the combination of the *Degree of scattering* and *Degree of tangling* metrics could help to reveal the problem, it would be a tedious task since the metrics do not provide information about which target elements are addressing each source element. Thus, the utilization of *Degree of crosscutting* allows the detection of this modularity problem just observing the values of one metric.

Finally, the conclusions extracted in this section for the MobileMedia system were also confirmed by the metric results obtained for the HealthWatcher and SmartHome systems – reported in our website [48]. As a summary of the data obtained for these systems, Figs. 7 and 8 show the *Degree of crosscutting* metric calculated for the five and three releases of these systems, respectively. Observe

Table 16
Dependency matrix for the MobileMedia system in release 7.

		Use cases																					
		Add Album	Delete Album	Add Media	Delete Media	View Photo	View Album	Provide Label	Store Data	Remove Data	Retrieve Data	Edit Label	Count Media	View Sorted Media	Set Favourite	View Favourites	Copy Media	Send Media	Receive Media	Music Control	Access Media	Play Video	Capture Media
Concerns	Album	2	2	1			3																
	Photo					1																	
	Label	2		2			1	1				1					2		1				
	Persistence	2	2	2	2	2	2	2	2	2	2		2		2		3	1			2		
	Error Handling	2	2	3	2	2	1	2	2	2	2	1	1	1	1	1	2	2	2		2		1
	Sorting					1	1						1	2							1		
	Favourites						2								1	1							
	Copy					1											1				1		
	SMS					1												1	1		1		
	Music																			3			
	Media		1	2	1		3												2		3		1
	Video																					1	
	Capture																						1

in the figure that crosscutting is mainly concentrated in the non-functional concerns involved in both systems, such as: (i) Error Handling, Persistence, and Access Control in HealthWatcher or (ii) Access to Physical KNX Devices, Authentication or User Notifications in SmartHome. There are also some functional concerns where crosscutting appears (e.g. RegisterComplaint and PopularComplaints in HealthWatcher). Similarly, we also observed that our conclusion about the need of a metric specific for crosscutting was supported by the data obtained for these systems. In particular, we observed situations where degree of crosscutting provides extra information, which are not highlighted by degree of scattering; this occurs due to an increase in degree of tangling. More conclusions about the utilization of the metrics can be found in Ref. [33], where we evaluated our metrics by comparing them with similar ones introduced by other authors.

6. Using feature modularity to assess maintainability attributes

This section complements our empirical modularity analysis in previous section by showing the usefulness of the concern metrics (Section 4.2). It evaluates to what extent the metrics support the anticipation of further problems associated with two quality attributes, namely changeability and stability. The overall goal of this analysis is to address our main research question MRQ (Section 1.1) and to investigate how our concern metrics may help in predicting these maintainability attributes based on the analysis of early software artifacts. In that sense, we analyse here whether the metrics provide indications that crosscutting negatively affects software maintainability. This hypothesis is tested by using a three-dimensional evaluation: (1) an evaluation showing the ability of the metrics for predicting changeability (Sections 6.1 and 6.2); (2) an analysis of the concern metrics in terms of their predictive power for measurement of software stability [50] (Sections 6.3 and 6.4); and (3) an evaluation of the metrics capability to assess feature dependencies [17] and how these dependencies affect to changeability and stability (Sections 6.5 and 6.6).

The first evaluation is carried out by observing the relations between source and target domains. It introduces a new metric for anticipating the set of impacted elements for a change in a particular source element (*"a priori"*, without the need of performing the change). As the analysis provides an estimation of the impact of a

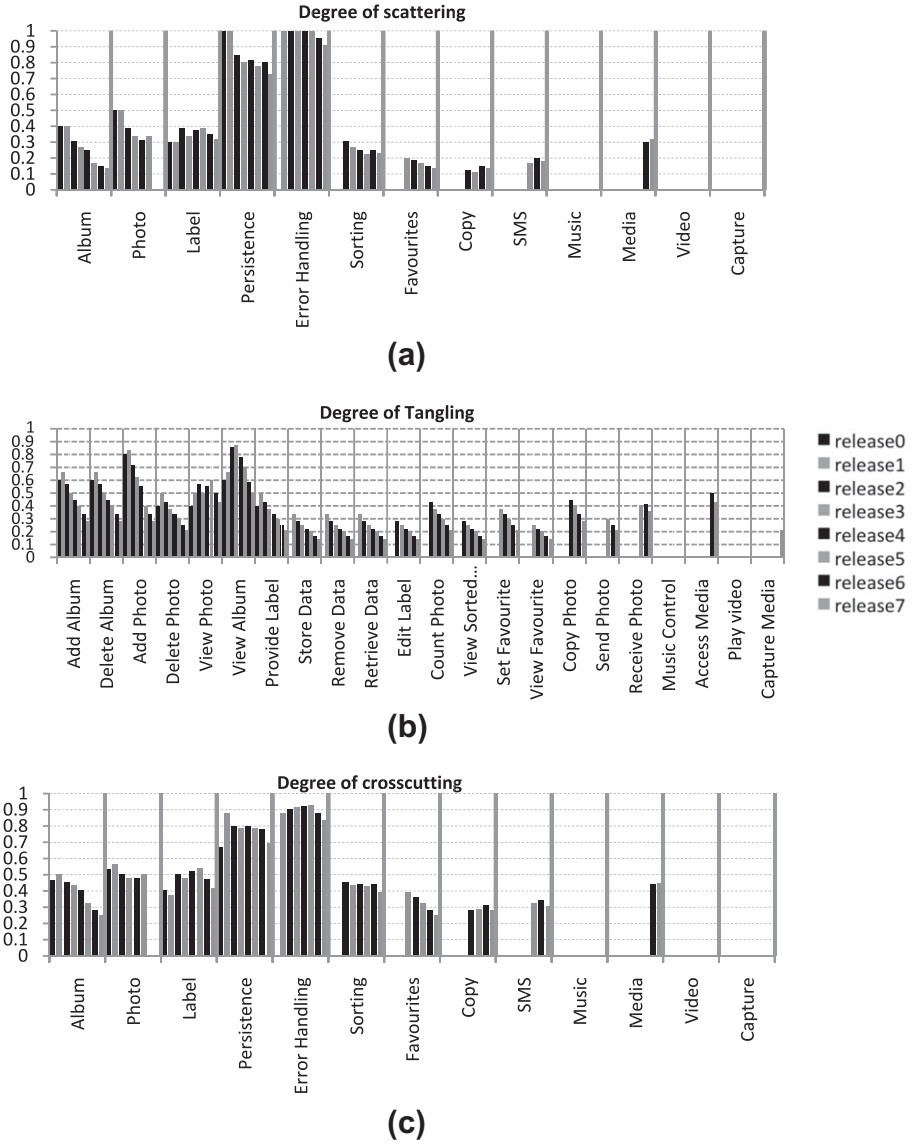

Fig. 4. Charts showing *Degree of scattering*, *Degree of tangling* and *Degree of crosscutting* metrics for the MobileMedia.

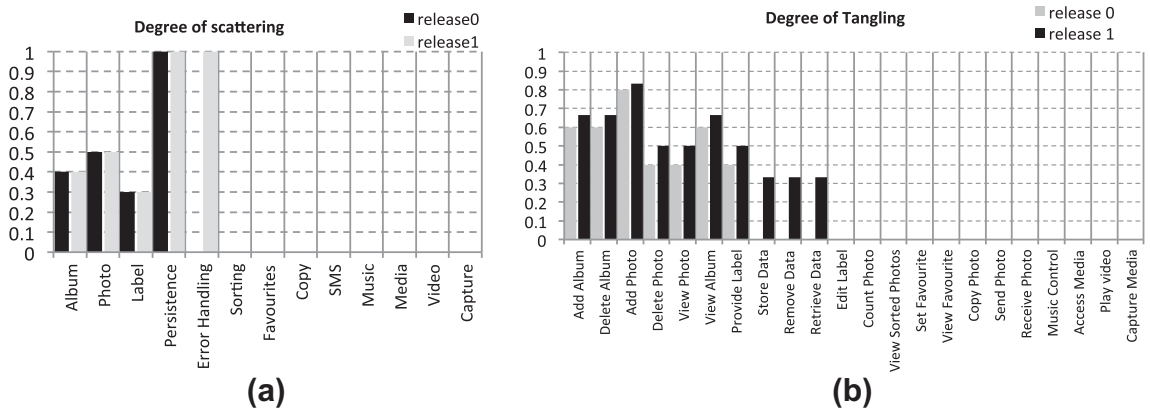

Fig. 5. *Degree of scattering* and *Degree of tangling* for releases 0 and 1.

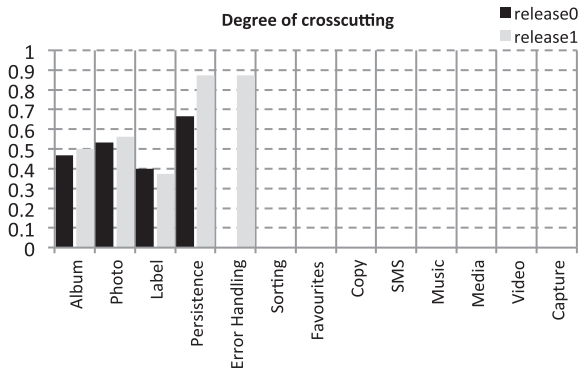

Fig. 6. *Degree of crosscutting* for releases 0 and 1.

change before carrying out the change, it also offers important traceability information since the elements affected by a change in a source element may be anticipated. This is even more important in the SPL context where family products may be built just changing features.

The second analysis is performed by empirically observing the actual changes produced in a system and relating these changes to the degree of scattering and crosscutting (the analysis is performed once the changes have produced, *"a posteriori"*). In this case, since the analysis is performed once the changes have occurred, the estimation provided in the first analysis is tested by observing the changes produced in different modifications. This analysis is also used to check the interplay between the two first analyses, observing whether the features with a higher impact set are also those implemented by more unstable use cases.

Finally, a third empirical analysis is performed where feature dependencies are compared with our *Degree of crosscutting* metric. The goal of this analysis is to observe whether crosscutting introduces feature dependencies, making changes to the features more difficult. In particular, the analysis compares crosscutting with concern interlacing [17], an important characteristic of features in software product lines. Interlacing refers to the dependency existing between two features when their implementations share, at least, one component. Then, we check whether the reduction of dependencies between features (including crosscutting dependencies) enhances the two aforementioned attributes, namely changeability and stability. This hypothesis is evaluated by empirically comparing the presence of feature dependencies with the attributes.

As previously mentioned, the different analyses are systematically explained by using the MobileMedia as running example. The HealthWatcher and SmartHome systems are described to enable us to check whether and when the results converge or not; then, we will emphasize their data whenever we observe different trends in these systems. All the raw data for the three systems is available at Ref. [48].

6.1. Changeability analysis

This section addresses RQ1 testing whether the features with higher impact of changes (no matter whether they are variable or mandatory) are those with higher *Degree of crosscutting* (*Hypothesis 1*). In order to have a way to assess the change impact of a feature, we use the idea presented in Ref. [23]. This work is also based on the crosscutting pattern introduced in Section 2 and it defines change impact in terms of the elements involved in the change of a source element. Assuming there are two related

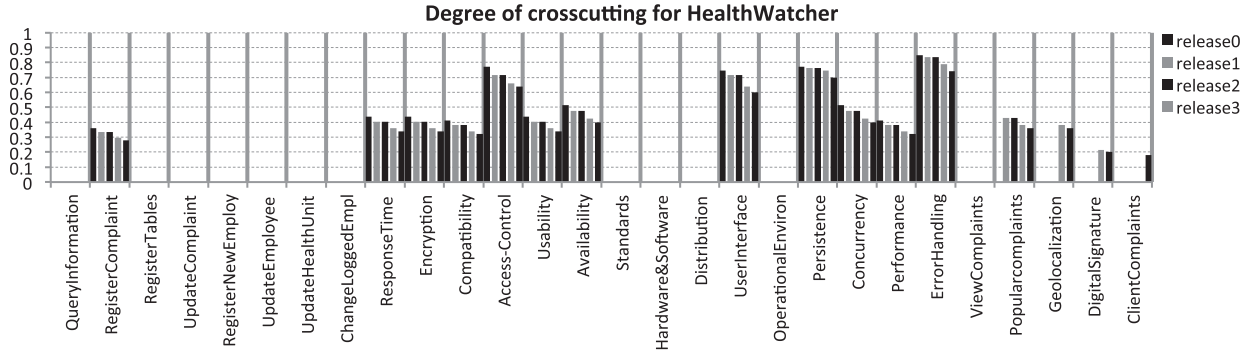

Fig. 7. *Degree of crosscutting* metric for the five releases of the HealthWatcher system.

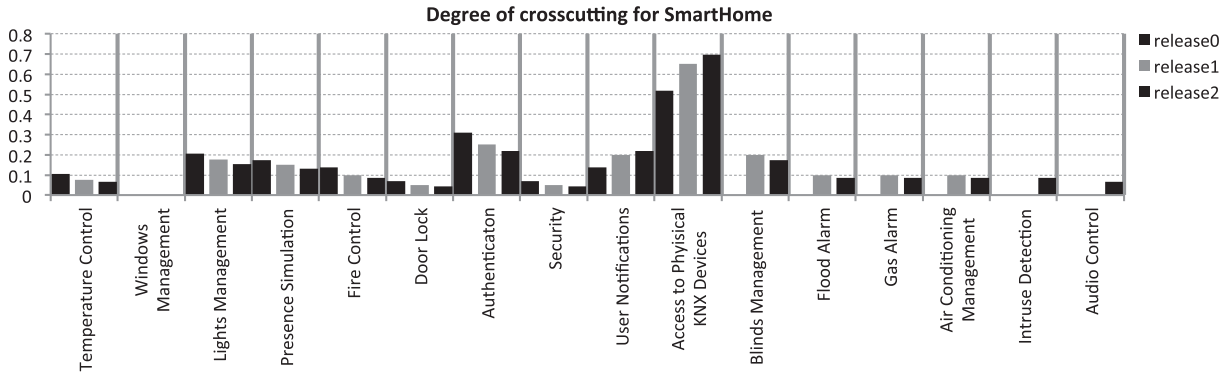

Fig. 8. *Degree of crosscutting* metric for the three releases of the SmartHome system.

domains (source and target), the change impact of a source element is defined as the addition of the number of trace links that must be changed and the number of trace links that must be preserved. In Fig. 9 an example of several source and target elements related is shown. A change in the source element s2 implies to change target elements t1, t2 and t3. There are three trace links from s2 to these target elements. However, since s1 and s3 are also addressed by two of the target elements changed, the trace links from s1 to t1 and from s3 to t3 must be also preserved, so that s1 and s3 are also affected by the change of s2: *impact* $(s2)$ = *change* $((s2, t1), (s2, t2), (s2, t3))$ + *preserve* $((s1, t1), (s3, t3))$.

Using this assumption, the change impact of a source element can be quantified using our dependency matrix (dm). In particular, the number of trace links affected by a change of a source element s_k is calculated counting the number of 1 values of row k and the number of 1 values in the columns where there are 1's in row k. This may be formally defined as:

$$\textit{Impact Set}\ (s_k) = \sum_{i=1}^{|T|}\left(\mathrm{dm}_{ki}\left(\sum_{j=1}^{|S|}\mathrm{dm}_{ij}\right)\right) \tag{11}$$

Applying formula (11), the impact set of a change in each feature in the different releases of MobileMedia has been calculated. Table 17 shows these results. The table shows, for each feature, the number of trace links affected by a change in the feature. These trace links include both the links to be changed and the links to be preserved. Since we are focusing on changes in features, in this analysis we consider modifications to accomplish changes in functional specifications or, in general, requirements of the system. An M, V, or O has been added to the feature's name indicating whether the feature is mandatory, variable or optional, respectively. Note also that the *Degree of scattering* and *Degree of crosscutting* for each feature are also shown in the table in order to be compared with the *Impact Set* metric. Finally, taking the results obtained by the change impact analysis, the *Impact Set* metric has been correlated with both *Degree of scattering* and *Degree of crosscutting* metrics. The results are shown in Figs. 10 and 11 where we can see the correlations for the eight different releases. We have used the least squares criterion to estimate the linear regression between the variables assessed.

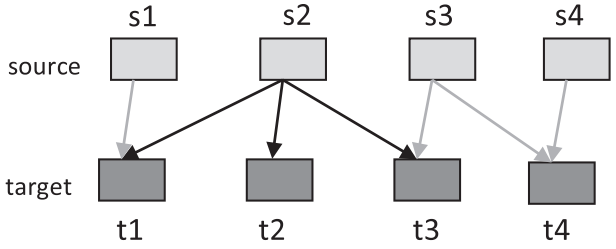

Fig. 9. A dependency graph with three and four source and target elements respectively.

Finally, in order to study how tangling affects changeability, the *Impact set* has been also compared with the *Degree of tangling*. Note that, the former is a metric defined in terms of source elements whilst the latter is defined based on target elements. Then, in order to compare both measurements, the *Concern degree of tangling* metric for each concern has been defined. This new metric is calculated by the addition of the *Degree of tangling* of each use case that addresses the corresponding concern. Then, this metric was computed by using the dependency matrix for each release. This matrix was modified to show the *Degree of tangling* of the different use cases instead of just representing the mappings existing between source and target elements. For the sake of simplicity, the dependency matrix for release 7 is the only one shown. This matrix is depicted in Table 18. As an example, observe that the cells of the first column have been changed by values of 0.28 which is the *Degree of tangling* of the Add Album use case. The *Concern degree of tangling* for each concern is shown in the last column of the table.

Then, once the *Concern degree of tangling* have been calculated for each release, these values are compared with the impact set metric previously calculated. Table 19 shows the values obtained for both metrics and for each release. Based on these values, both metrics have been also correlated. The correlation between Impact set and *Concern degree of tangling* for each release is presented in Fig. 12, from a to h.

Regarding to our second analyzed system, the HealthWatcher, Fig. 13 shows the correlation of the *Impact Set* metric with *Degree of Crosscutting* in the five releases of the system. Similarly, Fig. 14 shows the correlations between the metrics *Impact Set* and *Concern Degree of Tangling* for the same system. The same correlations are also shown for the SmartHome system in Figs. 15 and 16, respectively.

6.2. Discussion on changeability analysis

The main conclusions extracted from the changeability analysis are explained next:

- From Figs. 10 and 11 we observed that those features with higher *Degree of scattering* or *Degree of crosscutting* have a bigger set of elements impacted than those with lower values for the metrics. Accordingly, these metrics are linearly correlated as show the values obtained for Pearson's r [49] presented in the last two correlations in Figs. 10 and 11. Concretely, the values for correlation (h) in Figs. 10 and 11 are 0.993 and 0,981,

Table 17
Size of the impact set for features and NFCs and their degree of scattering and crosscutting.

	Release0			Release1			Release2			Release3			Release4			Release5			Release6			Release7		
Features	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting	Impact Set	Degree of scattering	Degree of crosscutting
Album (M)	13	0,4	0,46	17	0,4	0,5	19	0,3	0,45	20	0,267	0,43	20	0,25	0,40	15	0,16	0,32	15	0,15	0,28	15	0,13	0,25
Photo (M)	14	0,5	0,53	19	0,5	0,56	22	0,38	0,50	23	0,333	0,47	25	0,31	0,48	28	0,33	0,50	1	0	0	1	0	0
Label (V)	9	0,3	0,40	12	0,3	0,37	20	0,38	0,50	21	0,333	0,47	25	0,37	0,52	28	0,38	0,53	29	0,35	0,46	29	0,31	0,41
Persistence	22	1	0,66	32	1	0,87	38	0,84	0,80	42	0,8	0,78	47	0,81	0,80	50	0,77	0,78	61	0,8	0,78	61	0,72	0,69
Error Handling				32	1	0,87	42	1	0,90	48	1	0,91	53	1	0,92	60	1	0,92	67	0,95	0,87	70	0,90	0,83
Sorting (V)							15	0,3	0,45	16	0,267	0,43	17	0,25	0,44	18	0,22	0,42	24	0,25	0,43	24	0,22	0,38
Favourites (V)										12	0,2	0,39	12	0,18	0,36	12	0,16	0,32	12	0,15	0,28	12	0,13	0,25
Copy (V)													9	0,12	0,28	10	0,11	0,28	16	0,15	0,31	16	0,13	0,27
SMS (V)																13	0,16	0,32	20	0,2	0,34	20	0,18	0,30
Music (O)																			1	0	0	1	0	0
Media (M)																			29	0,3	0,43	32	0,31	0,44
Video (O)																						1	0	0
Capture (V)																						1	0	0

respectively. These values indicate that *Degree of scattering and Degree of crosscutting* metrics are highly correlated with the *Impact Set*. Note that the probability that these measures are correlated with *Impact Set* is higher than 99.95% in both cases (based on the critical values table for r [49]). These conclusions were also supported by the data obtained for HealthWatcher and SmartHome systems that are summarized from Figs. 13–16, where the reader may observe that the measures are also highly correlated in the five releases of this system.

- As it may be observed in the results, the correlation obtained for *Degree of scattering* and *Degree of crosscutting* are very similar. Interestingly, we observed that the correlations obtained for *Degree of scattering* were barely higher than those obtained for the *Degree of crosscutting* in the MobleMedia. However, in HealthWatcher and Smarthome, *Degree of crosscutting* presents a higher correlation with *ImpactSet*. These results suggest that, although being quite similar, as complexity of the system – in terms of system size and number of concerns – increases, the *Degree of Crosscutting* exhibits higher correlation with *Impact Set* than *Degree of scattering*. Note that while the *Degree of scattering* metric only considers scattering, the *Degree of crosscutting* metric includes information regarding scattering and tangling. As it has been shown in Section 5.2, in certain situations, the latter may provide information that the former does not do. This information is useful for discerning the features to refactor when there are some features with the same *Degree of scattering*.
- Fig. 12 shows how the *Impact set* metric is also highly correlated (with a value for r close to 1) with the *Concern degree of tangling* in the MobileMedia system so that the more tangled the use cases that address a concern, the more difficult to accomplish a change in this concern is. These results are also confirmed by the data obtained for HeatlhWatcher and SmartHome summarized in Figs. 14 and 16, respectively. This finding also reveals that it is likely the case that the

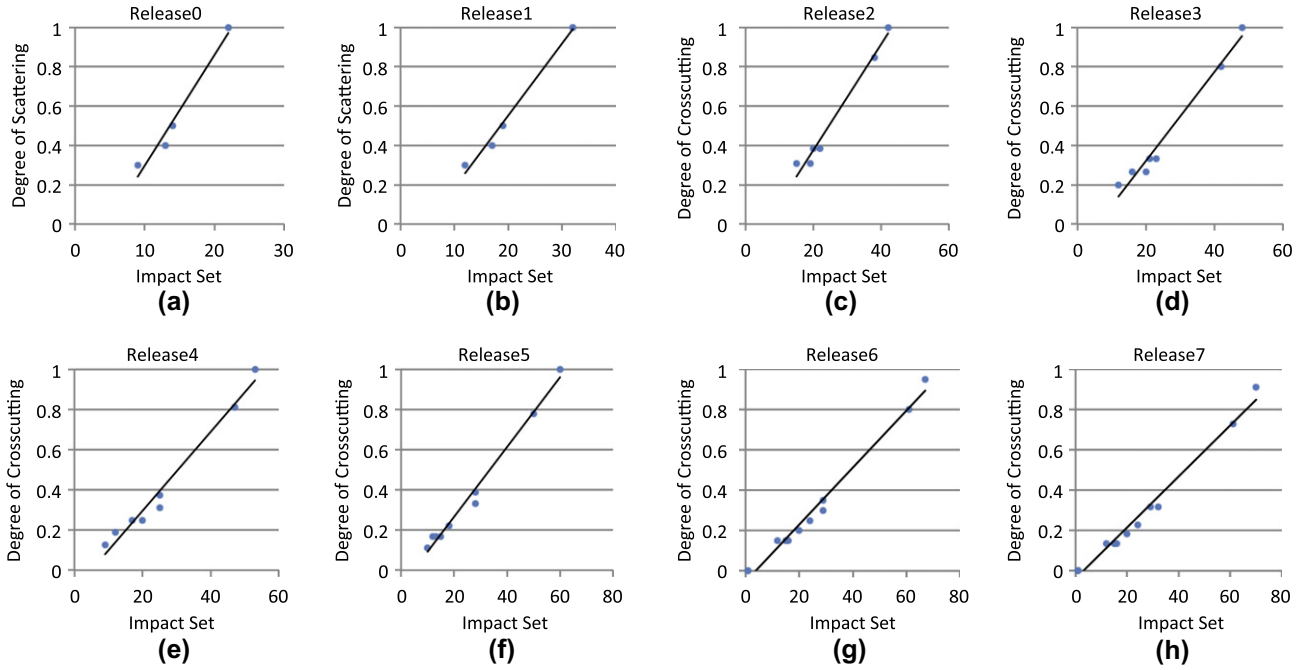

Fig. 10. Correlations between *Impact set* and *Degree of scattering* for the eight releases in MobileMedia.

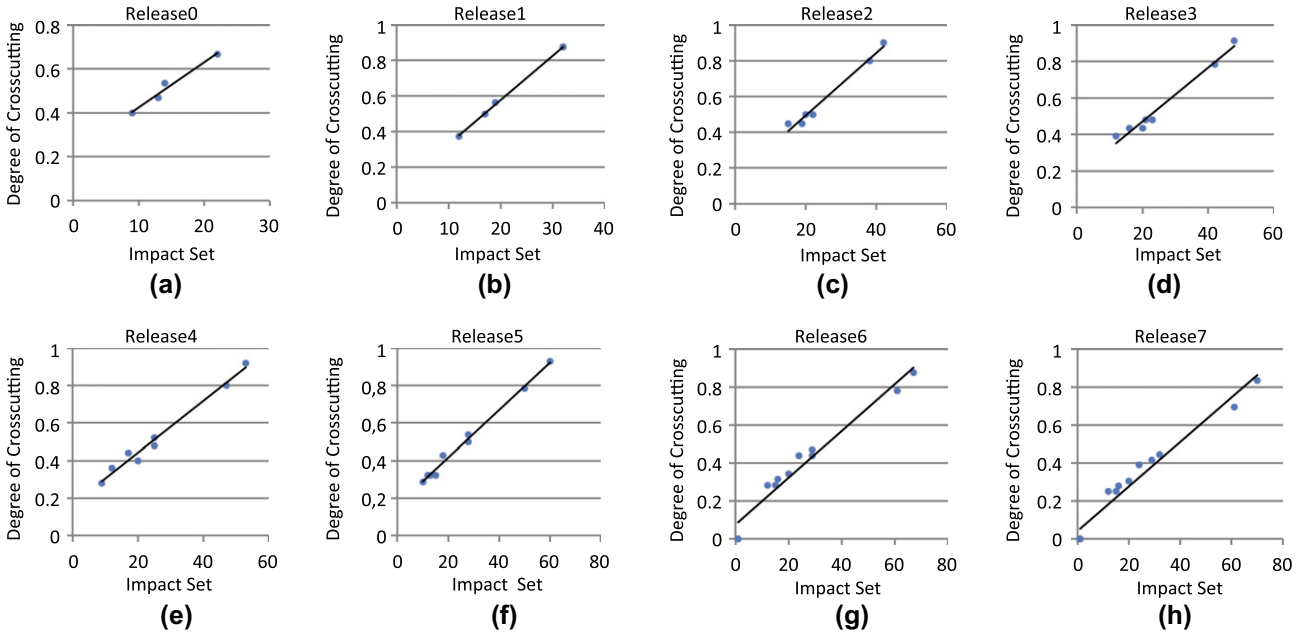

Fig. 11. Correlations between *Impact set* and *Degree of crosscutting* for the eight releases in MobileMedia.

decomposition of early artifacts, such as use cases, exerts major influence on the architectural and detailed design decomposition. As a consequence, the early modular decomposition tends to be carried through other development stages, thereby extending the crosscutting effects to artifacts and respective changes generated later. In fact, similar phenomena can be observed when concern measures are applied and analysed in code artifacts. This is one of the reasons that possibly explain why early concern metrics can be useful, probably even more in such cases where there is a natural traceability of features through all the SPL artifacts.

- We observed that the fact of being mandatory or variable does not influence on the impact set of elements of a feature. As an example, it can be observed in Table 17 how, in release 7 of the MobileMedia, the mandatory feature Album has a bigger impact set than the variable feature Favourites. However, Label (variability) has a bigger impact set than Album. In that sense, there are several works that propose aspect-oriented techniques for the modelling of variable features (e.g. Refs. [20,34–36,51,52]). However, the changeability analysis provides empirical evidence showing that mandatory features may be crosscutting features as well, supporting the need for performing an analysis to identify these crosscutting features (consistently with Ref. [20], where it is stated that common features could be also modelled using aspect-oriented techniques). Analogously, variable features do not need to be always defined as crosscutting concerns if they do not crosscut other features.
- Interestingly, we observed some cases where features with the same *Degree of crosscutting* have different impact sets. This is due to the fact that *Degree of crosscutting* metric is calculated based on the number of source and target elements involved in the crosscutting relation, whereas the *Impact Set* is quantified in terms of trace links instead of source or target elements. As an example, in Fig. 9 we can see how the *Ncrosscut* and *Crosscutpoints* metrics for s2 and s3 have the

Table 18
Degree of tangling of the use cases that address the different concerns and *Concern degree of tangling* metric for release 7.

		Use cases																						
		Add Album	Delete Album	Add Media	Delete Media	View Photo	View Album	Provide Label	Store Data	Remove Data	Retrieve Data	Edit Label	Count Media	View Sorted Media	Set Favourite	View Favourites	Copy Media	Send Media	Receive Media	Music Control	Access Media	Play Video	Capture Media	Concern Degree of tangling
Concerns	Album	0,28	0,28				0,5																	**1,071**
	Photo					0,42																		**0,429**
	Label	0,28		0,28			0,5	0,21				0,14					0,28		0,35					**2,071**
	Persistence	0,28	0,28	0,28	0,21	0,42	0,5	0,21	0,14	0,14	0,14		0,21		0,21		0,28	0,21	0,35		0,42			**4,357**
	Error Handling	0,28	0,28	0,28	0,21	0,42	0,5	0,21	0,14	0,14	0,14	0,14	0,21	0,14	0,21	0,14	0,28	0,21	0,35		0,42		0,21	**5**
	Sorting					0,42	0,5						0,21	0,14							0,42			**1,714**
	Favourites						0,5								0,21	0,14								**0,857**
	Copy					0,42											0,28				0,42			**1,143**
	SMS					0,42												0,21	0,35		0,42			**1,429**
	Music																			0				**0**
	Media		0,28	0,28	0,21		0,5												0,35		0,42		0,21	**2,286**
	Video																					0		**0**
	Capture																						0,23	**0,227**

Table 19
Impact set and *Concern degree of tangling* for the 8 releases.

	release0		release1		release2		release3		release4		release5		release6		release7	
Features	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling	Impact Set	Concern Degree of tangling
Album (M)	13	2,6	17	2,833	19	2,714	20	2,5	20	2,222	15	1,5	15	1,25	15	1,071
Photo (M)	14	2,8	19	3,167	22	3,143	23	2,875	25	2,667	28	2,8	1	0,5	1	0,429
Label (V)	9	1,8	12	2	20	2,857	21	2,625	25	2,778	28	2,8	29	2,417	29	2,071
Persistence	22	3,8	32	5,333	38	5,429	42	5,25	47	5,222	50	5	61	5,083	61	4,357
Error Handling			32	5,333	42	6	48	6	53	5,889	60	6	67	5,583	70	5
Sorting (V)					15	2,143	16	2	17	1,889	18	1,8	24	2	24	1,714
Favourites (V)							12	1,5	12	1,333	12	1,2	12	1	12	0,857
Copy (V)									9	1	10	1	16	1,333	16	1,143
SMS (V)											13	1,3	20	1,667	20	1,429
Music (O)													1	0	1	0
Media (M)													29	2,417	32	2,286
Video (O)																0
Capture (V)																0,227

same values: 2 in both cases (s2 is crosscutting to s1 and s3 in two target elements whilst s3 is crosscutting to s2 and s4 in two target elements). That implies that the degree of crosscutting for s2 and s3 would be the same, however, the impact sets of s2 and s3 are not the same. While the impact set of s2 includes 5 different trace links, those from s2 to t1, t2, t3 and those from s1 to t3 and s3 to t4; the impact set of s3 includes 4 trace links, those from s3 to t2, t3 and those from s2 to t3 and s4 to t4.

The final conclusions that summarize this section and verify Hypothesis 1 (answering question RQ1) are the next:

- **C1**: *The higher the Degree of crosscutting or Scattering for a feature, the wider the impact set of a change in this feature.*
- **C2**: *The higher the Degree of tangling of the use cases that address a feature, the wider the impact set of a change in this feature.*

6.3. Stability analysis

To date, there is no empirical study that investigates whether scattering and crosscutting negatively affect software stability at requirements level. This section addresses RQ2 by analysing whether the concerns with a higher degree of scattering and crosscutting are addressed by more unstable use cases than concerns with lower degree of scattering and crosscutting (Hypothesis 2). It also compares the degree of tangling of the use cases in order to study whether those with higher values are more unstable. Stability is highly related to change management so that the more unstable a system is, the more complicated the change

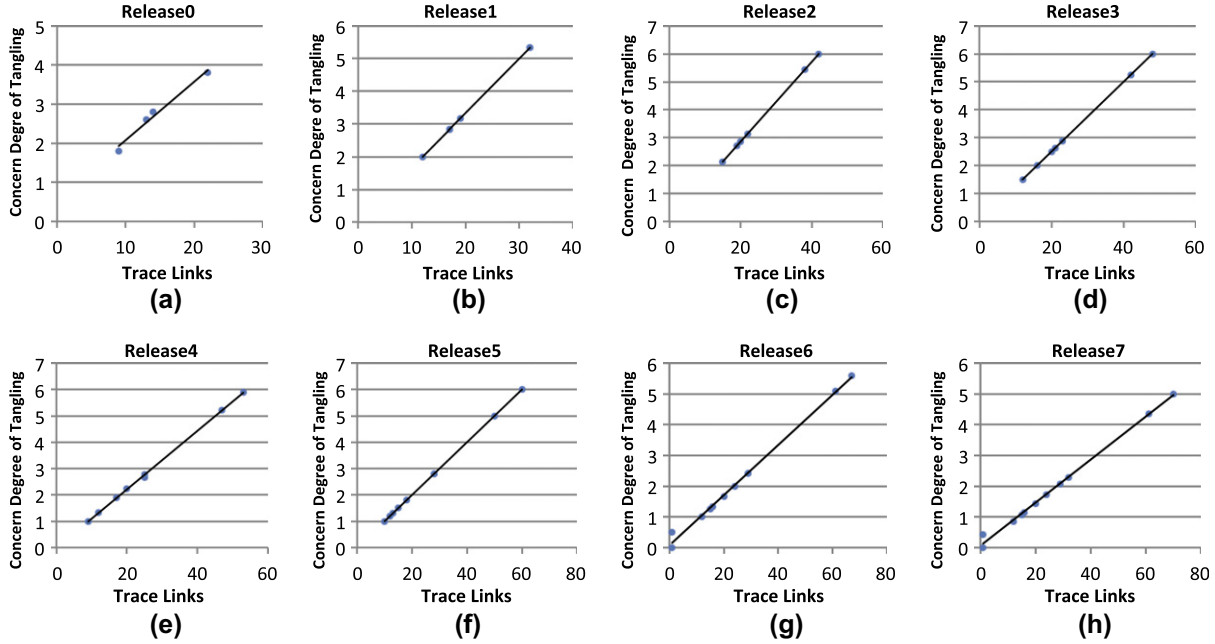

Fig. 12. Correlation between *Impact set* and *Concern degree of tangling* for MobileMedia.

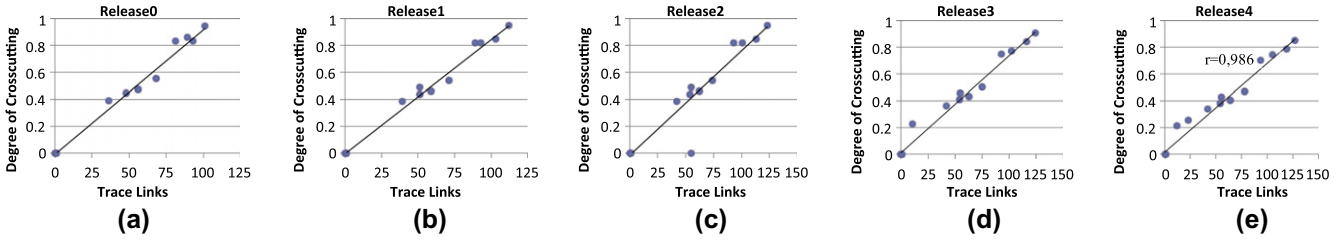

Fig. 13. Correlation between *Impact Set* and *Degree of Crosscutting* in the HealthWatcher system.

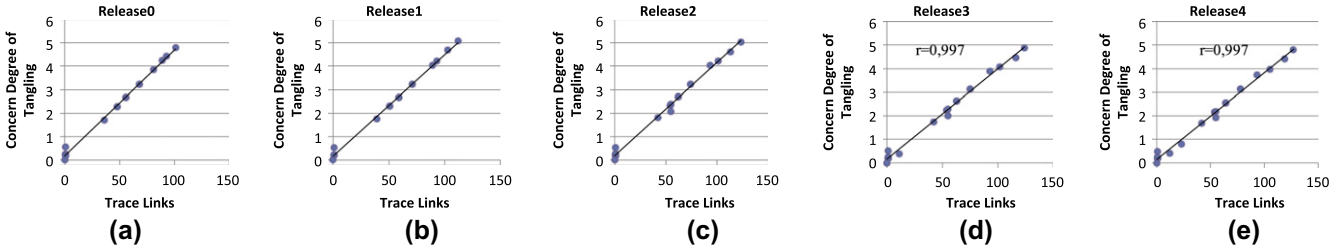

Fig. 14. Correlation between *Impact Set* and *Concern Degree of Tangling* in the HealthWatcher system.

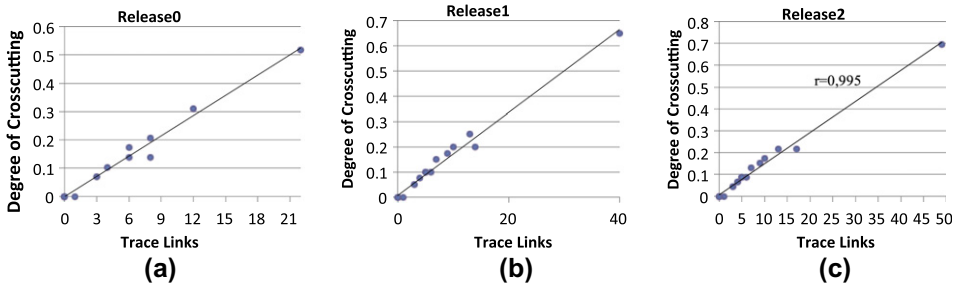

Fig. 15. Correlation between *Impact Set* and *Degree of Crosscutting* in the SmartHome system.

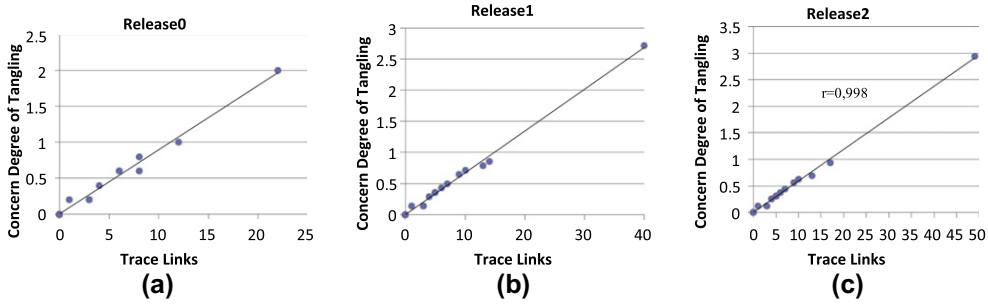

Fig. 16. Correlation between *Impact Set* and *Concern Degree of Tangling* in the SmartHome system.

management becomes [23]. Unstable models gradually lead to the degeneration of the design maintainability and its quality in general [23]. Then, we can infer that early crosscutting is likely to have a negative effect on software quality. In this analysis we mainly focus on changes in the functionality of the system, i.e. modifications to accomplish changes in requirements or functional specifications [10]. It is important to highlight that our focus is not on corrective changes performed to remove bugs or on perfective modifications. However, we do not rule out this kind of changes in future analyses and they should be the target of further studies.

In order to perform our empirical study, Table 20 shows the use cases (rows) which are modified in the different releases (columns) of the MobileMedia. A modification in a use case is mainly due to: either (i) the concerns, which it addresses, have evolved or (ii) it has been affected by the addition, modification or removal of a concern to the system. In this table, a '1' in a cell represents that the corresponding use case has been modified in that release. As an example, in release 1 (r1) all the cells in the column present the value 1. This is due to the fact that Error Handling is added in this release, and this concern affects all use cases. An "a" in a cell represents that the use case is added in that release. There are also some use cases whose names are modified in a release. These use cases are marked in the "Renaming" column, where the release which introduces the modification in the name is shown (e.g. Add Photo use case modifies its name to Add Media in release 6). Finally, use cases with a number of modifications higher than a

Table 20
Changes in use cases in the different releases.

		Releases									
Renaming	Requirements Element	r0	r1	r2	r3	r4	r5	r6	r7	#Changes	Unstable?
	Add Album	a	1	0	0	0	0	0	0	1	no
	Delete Album	a	1	0	0	0	0	0	0	1	no
r6	Add Photo [Media]	a	1	0	0	0	0	1	0	2	yes
r6	Delete Photo [Media]	a	1	0	0	0	0	1	0	2	yes
	View Photo	a	1	1	0	1	1	1	0	5	yes
	View Album	a	1	1	1	0	0	1	0	4	yes
	Provide Label	a	1	0	0	0	0	0	0	1	no
	Store Data	a	1	0	0	0	0	0	0	1	no
	Remove Data	a	1	0	0	0	0	0	0	1	no
	Retrieve Data	a	1	0	0	0	0	0	0	1	no
	Edit Label			a	0	0	0	0	0	0	no
r6	Count Photo [Media]			a	0	0	0	1	0	1	no
r6	View Sorted Photo			a	0	0	0	1	0	1	no
	Set Favourites				a	0	0	0	0	0	no
	View Favourites				a	0	0	0	0	0	no
r6	Copy Photo [Media]					a	0	1	0	1	no
r6	Send Photo [Media]						a	1	0	1	no
r6	Receive Photo [Media]						a	1	0	1	no
	Play Music							a	0	0	no
	Access Media							a	0	0	no
	Play Video								a	0	no
	Capture Media								a	0	no

threshold value (higher than 1 in our analysis) are marked as unstable.

To study how crosscutting properties affect to stability, once the changes affecting each use case are known, these values are used to compare with our metrics. First, we observe the *Degree of tangling* for each use case in order to analyse if those use cases with a higher number of changes (unstable) are also more tangled (whole data in Ref. [48]). Then, in Table 21 the *Degree of tangling* for each use case is compared with the number of changes. In order to relate stability with metrics for source elements, the dependency matrix for release 7 of the MobileMedia was observed to calculate the number of unstable use cases which realise each concern. The results obtained are presented in Table 22. In this table, the columns show the unstable use cases, in particular, those with two modifications or more. The concerns are shown in the rows. A cell with 1 represents that the use case addresses the corresponding concern. The last column of the table shows the total number of unstable use cases contributing to each concern.

Then, the number of unstable use cases for each concern is related with the degree of scattering and crosscutting for such concerns. In particular, Fig. 17 shows the linear regression between the number of unstable use cases and the *Degree of scattering* and *Degree of crosscutting* metrics, respectively. Again, the least squares criteria to estimate the linear regression between the variables assessed was used. At a first glance, we may anticipate that use cases addressing scattered or crosscutting concerns are more prone to be unstable. This first intuition is later confirmed by the computation of Pearson's coefficient, as it is explained in next section. The selection of scattering and tangling metrics is influenced by our purpose of comparing modularity with stability. Then, this is why we select the metrics quantifying crosscutting properties (closely related to modularity).

The process to compare modularity and stability was replicated for the HealthWatcher and SmartHome systems. We do not show here all the data calculated in the process but just show the correlations between instability and both *Degree of scattering* and *Degree of crosscutting* metrics. These correlations may be observed in Figs. 18 and 19 for both systems, respectively.

6.4. Discussion on stability analysis

In this section, we present some conclusions extracted from the analysis performed in the previous section.

Table 21
Degree of tangling and number of changes for each use case.

		Use cases																				
		Add Album	Delete Album	Add Photo	Delete Photo	View Photo	View Content	Provide Label	Store Data	Remove Data	Retrieve Data	Edit Label	Count Photo	View Sorted Photos	Set Favourite	View Favourite	Copy Photo	Send Photo	Receive Photo	Music Control	Access Media	Play video	Capture Media
Metrics	Degree of tangling	0,48	0,48	0,57	0,35	0,51	0,69	0,35	0,2	0,2	0,2	0,21	0,32	0,21	0,29	0,2	0,37	0,25	0,39	0	0,46	0	0,21
	#Changes	1	1	2	2	5	4	1	1	1	1	0	1	1	0	0	1	1	1	0	0	0	0

Table 22
Number of unstable use cases addressing each concern.

		Use cases				
		Add Media	Delete Media	View Photo	View Album	Unstable use cases
Concerns	Album	1			1	2
	Photo	1	1	1	1	4
	Label	1			1	2
	Persistence	1	1	1	1	4
	Error Handling	1	1	1	1	4
	Sorting			1	1	2
	Favourites				1	1
	Copy			1		1
	SMS			1		1
	Music					0
	Media	1	1		1	3
	Video					0
	Capture					0

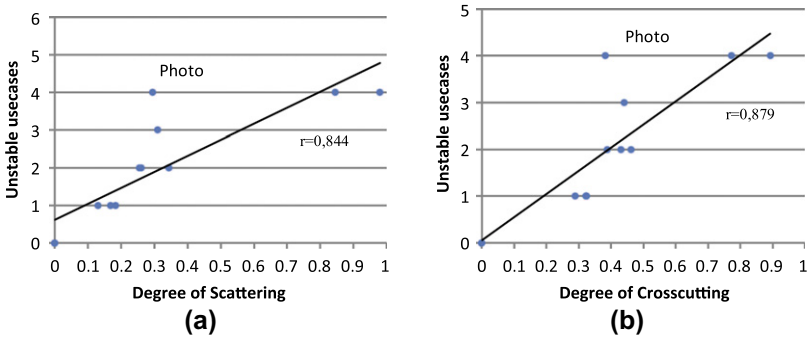

Fig. 17. Correlations of *Degree of Scattering* and *Degree of Crosscutting* with instability in the MobileMedia.

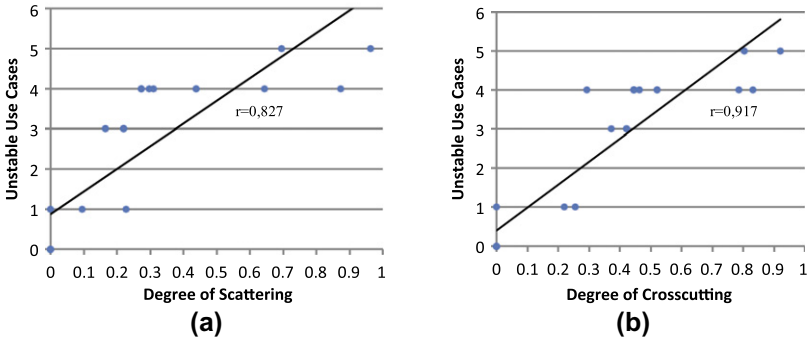

Fig. 18. Correlations of *Degree of Scattering* and *Degree of Crosscutting* with instability in the HealthWatcher.

- Correlations follow a linear tendency so that the higher the degree of scattering or crosscutting for a concern in the MobileMedia is, the more unstable use cases addressing this concern are (see Fig. 17). The values for Pearson's *r* [49] shown in the graphics come to confirm this correlation between the measures. Concretely, the probability that the variables measured in Fig. 17a and b are linearly correlated is 99.9% [49], which indicates a high correlation of *Degree of scattering* and *Degree of crosscutting* with the number of unstable use cases. This conclusion is also supported by the results obtained for the HealthWatcher and SmartHome systems, summarized in Figs. 18 and 19.
- We observed how the unstable use cases present, in general, higher *Degree of tangling* than those use cases with a lower number of changes. Then, the assessment of tangling properties may also help to anticipate the target elements that are more prone to change. In particular, in the MobileMedia the *View content* or *View photo* use cases are those with a higher *Degree of tangling* and also with a higher number of changes (Table 21).
- Since the analysis is performed at requirements, the developer may anticipate important decisions about stability at this early stage of development, improving the later architecture or detailed design of the system. Moreover, the results

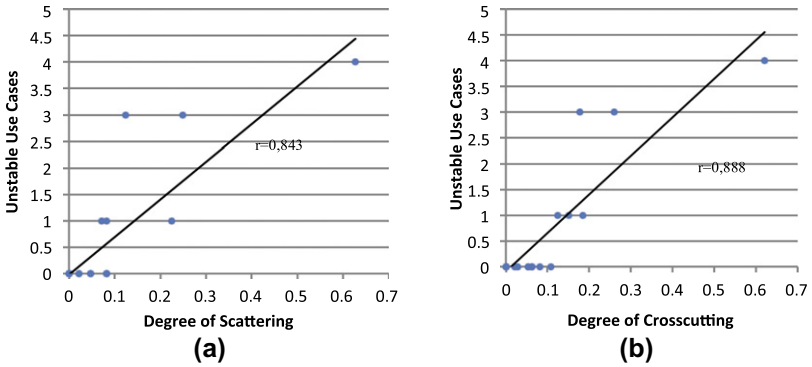

Fig. 19. Correlations of *Degree of Scattering* and *Degree of Crosscutting* with instability in the SmartHome.

obtained by this analysis were consistent with those shown in Section 6.1. In particular, the features with a higher *Degree of crosscutting* are those implemented by more unstable use cases and also those with a bigger *Impact Set* of elements. As an example, Persistence and Error Handling concerns in MobileMedia have the higher *Impact Set* (see Table 17) values and they are also the concerns more unstable.

- It has been observed that, in general, we obtained a higher correlation for the *Degree of crosscutting* with stability than for *Degree of scattering* with stability in the three systems analysed (Figs. 17–19). After analysing the data, we observed that the correlation between *Degree of scattering* metrics and stability was much influenced by those concerns either without scattering or completely scattered. As an example, we can see in Fig. 17a that there is a point with a *Degree of scattering* of almost 1 while most of the points present a *Degree of scattering* lower than 0.4. This fact highly influences the correlation.
- Interestingly, the correlations for the MobileMedia (Fig. 17) have been also annotated with a point called *Photo being the* most digressed from the linear regression in all the figures. We realized that this concern presents a high degree of scattering and crosscutting in the six first releases. After release 5, a new concern is added (Media) which is responsible for addressing the actions common to photo, music and video, and carrying out many actions previously assigned to the Photo concern. This is why *Degree of scattering* and *Degree of crosscutting* for Photo drastically decrease in releases 6 and 7. It highly influences the average of the metrics and this is the reason why, although having non-relatively high values for the metrics, the number of unstable use cases remains high. The same situation was observed for some values obtained for tangling metrics, e.g. the *Delete photo* unstable use case has a *Degree of tangling* lower than other non unstable use cases. The reason is that this use case implements part of the functionality of the Photo concern in the first releases, however, part of its tangled functionality is removed from it after release 5.

In this case, the final conclusions extracted from the analysis to verify Hypothesis 2 and answer RQ2 are:

- **C3**: *The higher the Degree of crosscutting for a feature, the more unstable the software artifacts that address this feature are.*
- **C4**: *The higher the Degree of tangling of a use case, the more unstable the use case is.*

6.5. Dependency analysis

It has been already observed that feature dependencies are harmful for maintainability of software assets [17,20,35,39]. However, to date there is no empirical study analysing whether the presence of crosscutting features at requirements level increments dependencies between features, making thus, their maintainability difficult. This section addresses RQ3 investigating whether the presence of crosscutting introduces dependencies between features (Hypothesis 3.1). This hypothesis is tested by using our concern metrics to infer information about the feature dependencies existing in the product line. Moreover, since crosscutting has been compared with change impact and stability in previous sections, we also compare these two maintainability attributes with feature dependencies showing their relation (Hypothesis 3.2).

In Ref. [17], the authors introduce the concepts of feature interlacing in order to assess feature dependencies in a product line. Interlacing denotes the situation where *the implementations of two features, F1 and F2, have one or more components (or operations) in common* [17]. The authors introduce the metric *Component-level Interlacing Between Concerns* (CIBC) which counts the number of components shared by two different features. This metric, among others defined in Refs. [17,18,27,37], have been previously used to assess dependencies at the programming, detailed design or architectural levels. However, to date, they have not been applied to measure dependencies at an earlier stage, i.e. in requirements documents. In this section, the metrics have been applied to the requirements of the three product lines used in our analyses. In particular we have defined and calculated a new *Use Case-level Interlacing Between Concerns* (UCIBC) metric, which measures the number of use cases shared by the implementation of two features. Moreover, we have defined a new metric, called *Concern Interlacing* (CI), as the total number of feature dependencies of a particular feature.

The *Concern Interlacing* metric is calculated by the addition of the UCIBC values for a feature with respect to the rest of features. As an example, in a system where three features (A, B and C) are considered, if feature A shares 2 use cases with feature B and 4 uses cases with feature C, then the *Concern Interlacing* metric for A is 6. Note that a use case may be computed twice if feature A shares this use case with two different features. Both metrics, UCIBC and CI are calculated using our dependency matrix. In particular, the calculation of UCIBC metric is obtained by counting the number of columns of the dependency matrix where two particular features have a 1. In other words, this metric can be calculated for all the features by performing the product of the dependency matrix and its transpose. The matrix resulting of this product is called the Concern interlacing matrix. CI metric is calculated by counting the values obtained for each row of this matrix (discarding the values obtained in the diagonal). The formalization of Concern Interlacing Matrix and CI metric are as follows:

$$\textit{Concern interlacing matrix } (cim) = DM \cdot DM^{\mathrm{T}} \tag{12}$$

$$\textit{Concern Interlacing } (s_k) = \left(\sum_{i=1}^{|T|} cim_{ki}\right) - \mathrm{cim_{kk}} \tag{13}$$

where *DM* and cim_{ki} represent the Dependency matrix and the cell $[k, i]$ of the Concern interlacing matrix, respectively.

In order to illustrate the calculation of the UCIBC metric, Table 23 shows the dependency matrix obtained for release 3 of the MobileMedia system. By applying the product of this matrix and its transpose, the Concern Interlacing Matrix shown in Table 24 is obtained. In this matrix, a cell denotes the value of the UCIBC metric for the feature represented in the row with respect to the feature of the corresponding column. The values of the CI metric for each feature are also shown in the last column of this matrix. As an example, observe that the UCIBC metric for Photo and Label features is 2 since these features share the implementation of the AddPhoto and ViewAlbum use cases. However, Photo shares the implementation of DeleteAlbum, AddPhoto and ViewAlbum use cases with the Album concern, thus the metric is 3 for these two features. Finally, the resulting CI metric for photo is 18 since the feature has 18 occurrences of use cases shared with other features or NFCs.

Using the dependency matrix for each release, we have calculated the CI metric for the 8 releases of the MobileMedia. The gathered data are shown in Table 25. For each release the degree of crosscutting for each feature has been also shown in order to have a way of comparing both metrics. The last column of the table shows the average of both metrics taking into account the eight releases. In this case, we do not show the *Degree of scattering* metric since, like with the previous analyses, the results obtained are very similar to those obtained for the *Degree of crosscutting* metric. Analogously, *Concern degree of tangling* metric is not shown since the results are also consistent and similar to the shown for the *Degree of crosscutting* metric.

Based on the data presented in Table 25, *Concern Interlacing* and *Degree of crosscutting* metrics have been compared in order to observe the correlation that exists between both metrics. The correlations for each release are shown in Fig. 20, from a to h.

Since the *Degree of crosscutting* metric has been compared with the *Concern Interlacing* and *Change Impact Set* metrics, we compare now these two metrics. Intuitively, since *Degree of crosscutting* is correlated with *Concern Interlacing* and *Change Impact*, these two last metrics should be also correlated. However, in order to check this conclusion, we have empirically compared the two metrics. The results are used to check whether the features with higher concern interlacing (more dependencies) are those with a bigger change impact set. The graphics that show the correlation between *Change Impact* and *Concern Interlacing* for each release are shown in Fig. 21.

Finally, once *Concern Interlacing* has been compared with *Degree of crosscutting* and *Change Impact Set* metrics, we may also compare it with the stability of the different features. The results allow us to

Table 23
Dependency matrix for the MobileMedia.

Features \ Use case	A um	Delete A um	Ad Phot	Delete P oto	View Photo	View Album	Pro ide Label	Store Data	Remove Data	Retrieve Data	Edit Label	Count Photo	View Sorted Photos	Set Favourite	View Favourites
Album	1	1	1			1									
Photo		1	1	1	1	1									
Label	1		1			1	1				1				
Persistence	1	1	1	1	1	1	1	1	1	1		1		1	
Exception Handling	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Sorting					1	1						1	1		
Favourites						1								1	1

Table 24
Concern interlacing matrix for the MobileMedia.

Features	Album	Photo	Label	Persistence	Error Handling	Sorting	Favourites	CI
Album	4	3	3	4	4	1	1	16
Photo	3	5	2	5	5	2	1	18
Label	3	2	5	4	5	1	1	16
Persistence	4	5	4	12	12	3	2	30
Exception Handling	4	5	5	12	15	4	3	33
Sorting	1	2	1	3	4	4	1	12
Favourites	1	1	1	2	3	1	3	9

Table 25
Concern Interlacing and *Degree of crosscutting* for the MobileMedia system.

	Release0		Release1		Release2		Release3		Release4		Release5		Release6		Release7		Average	
Features	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting	Concern Interlaicng	Degree of crosscutting
Album (M)	9	0,46	13	0,50	15	0,45	16	0,43	16	0,40	12	0,32	12	0,28	12	0,25	13,12	0,38
Photo (M)	9	0,53	14	0,56	17	0,50	18	0,47	19	0,48	22	0,50	5	0	5	0	13,62	0,38
Label (V)	6	0,40	9	0,37	15	0,50	16	0,47	19	0,52	21	0,53	22	0,46	22	0,41	16,25	0,46
Persistence	12	0,66	22	0,87	27	0,80	30	0,78	34	0,80	36	0,78	45	0,78	45	0,69	31,37	0,77
Error Handling			22	0,87	29	0,90	33	0,91	37	0,92	42	0,92	48	0,87	50	0,83	37,28	0,89
Sorting (V)					11	0,45	12	0,43	13	0,44	14	0,42	19	0,43	19	0,38	14,66	0,42
Favourites (V)							9	0,39	9	0,36	9	0,32	9	0,28	9	0,25	9	0,32
Copy (V)									7	0,28	8	0,28	13	0,31	13	0,27	10,25	0,28
SMS (V)											10	0,32	16	0,34	16	0,30	14	0,32
Music (O)													0	0	0	0	0	0
Media (M)													23	0,43	25	0,44	24	0,44
Video (O)															0	0	0	0
Capture (V)															2	0	2	0

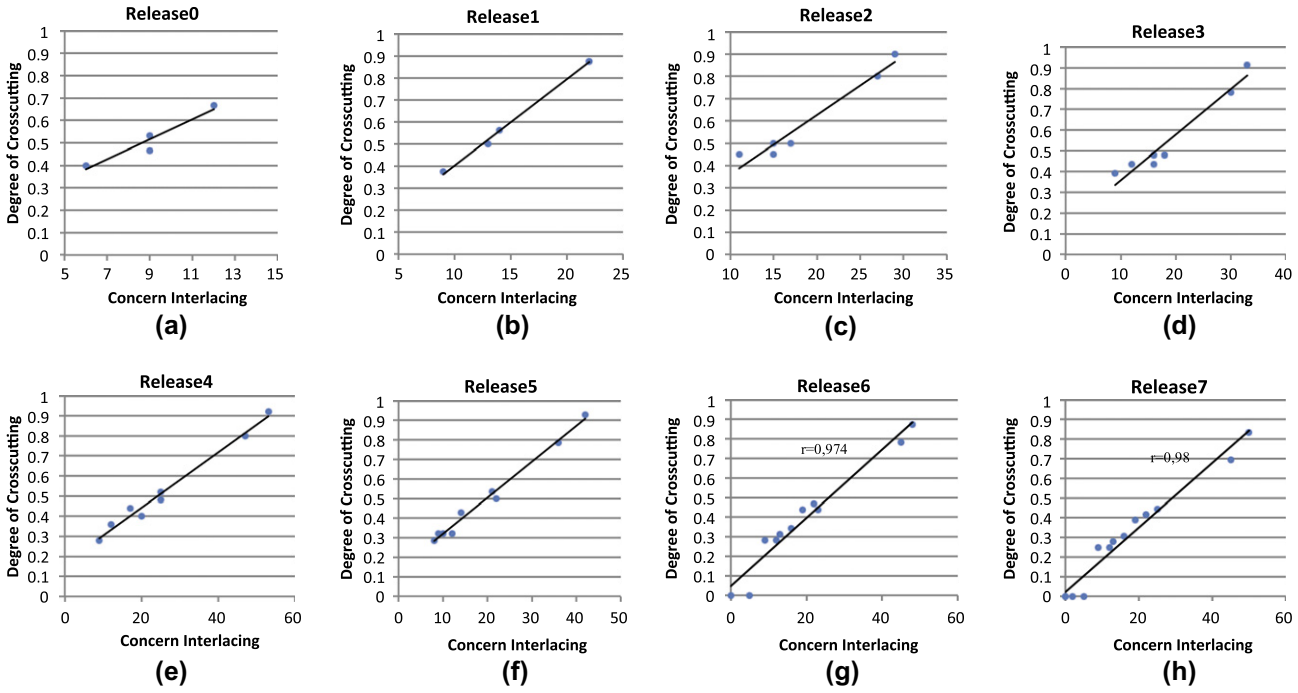

Fig. 20. Correlation between *Concern Interlacing* and *Degree of crosscutting* metrics for the MobileMedia.

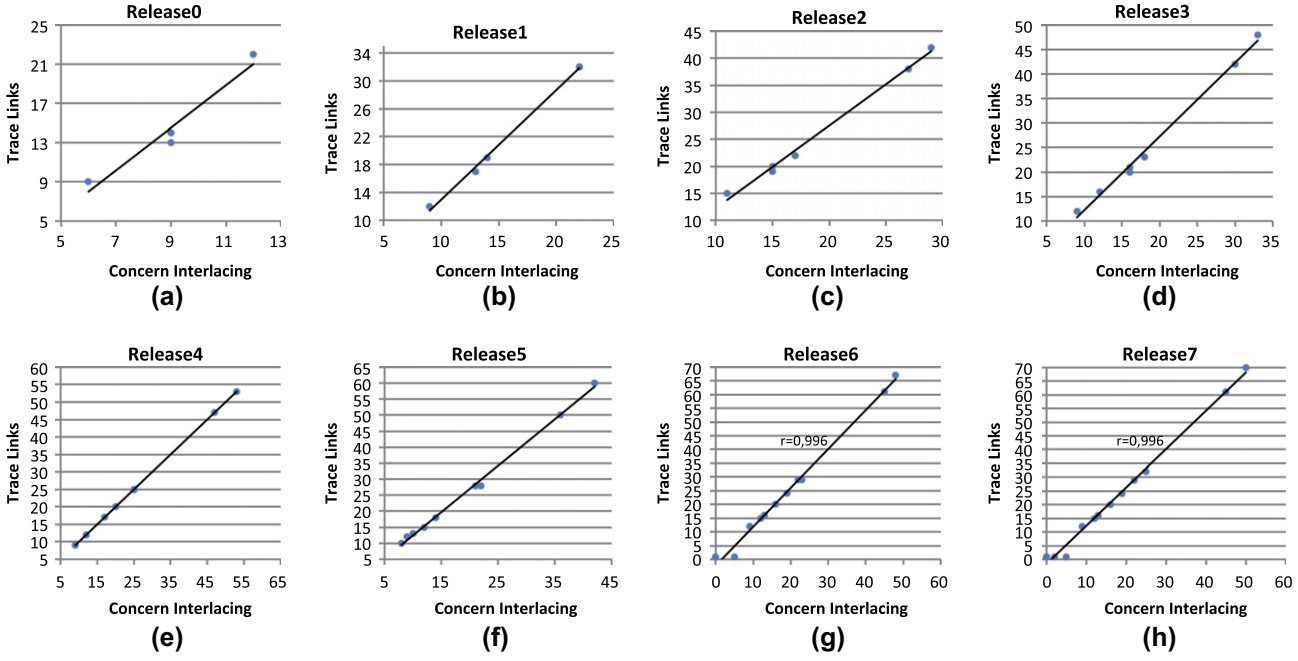

Fig. 21. Correlation between *Concern Interlacing* and *Change Impact Set* metrics for the MobileMedia.

check whether the features with more dependencies are those implemented by more unstable use cases. Fig. 22 shows the correlation existing between these two measurements. The data to obtain this graphic are extracted from Table 22 (unstable use cases) and Table 25 (concern interlacing average).

As we did with the previous studies, we replicated the process for the HealthWatcher and SmartHome systems in order to check whether the results obtained for the product lines are consistent. The results obtained are shown in the next figures: Figs. 23 and 24 show the correlations between *Concern Interlacing* and *Degree of crosscutting* for HealthWatcher and SmartHome, respectively; Fig. 25 does the same for *Concern interlacing* and *Instability*. The correlations between *Concern Interlacing* and *Impact Set* for these systems are not shown as the results are similar to those presented for *Concern Interlacing* and *Degree of Crosscutting*.

6.6. Discussion on feature dependency analysis

The main conclusions extracted while observing the results obtained in previous section are:

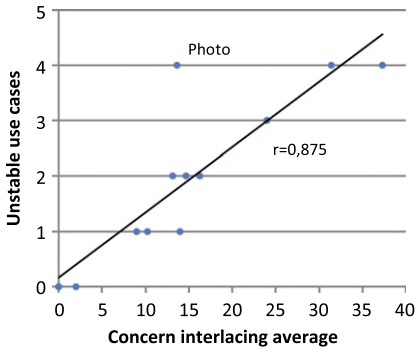

Fig. 22. Correlation between *Concern interlacing* and *Instability* for the MobileMedia.

- Fig. 20 shows that *Degree of crosscutting* and *Concern Interlacing* metrics are linearly correlated in the MobileMedia so that the more dependencies between features, the higher the *Degree of crosscutting* of these features (confirmed by the values obtained for Pearson's *r*). The data obtained for HealthWatcher and SmartHome systems also support this conclusion (Fig. 23). Of course, the inverse conclusion could be also stated, since a higher *Degree of crosscutting* introduces feature dependencies, making, thus, maintainability of the features difficult. This conclusion supports the need for introducing a modularity analysis so that the identification of crosscutting features at early development stages enhances the identification of feature dependencies.
- We observed that there are variable features with values for *Concern Interlacing* similar to the obtained for mandatory features (see Table 25). Even, there are variable features with lower concern interlacing results. Therefore, the conclusion extracted was that feature dependencies are highly determined by the crosscutting property of the features but not by their commonality or variability nature (similarly to the conclusion mentioned for changeability).
- Regarding to the *Concern Interlacing* metric, observe that a new metric to quantify this property for each feature has been introduced. This metric is based on those previously introduced in Refs. [17,18,27,37]. However, previous section has demonstrated how the calculation of this metric may be automated using our matrices, like with the rest of metrics presented here. This fact, together with the process to automatically discover source-to-target mappings (described in Ref. [39]), allows achieving a high degree of automation in the empirical analysis performed.

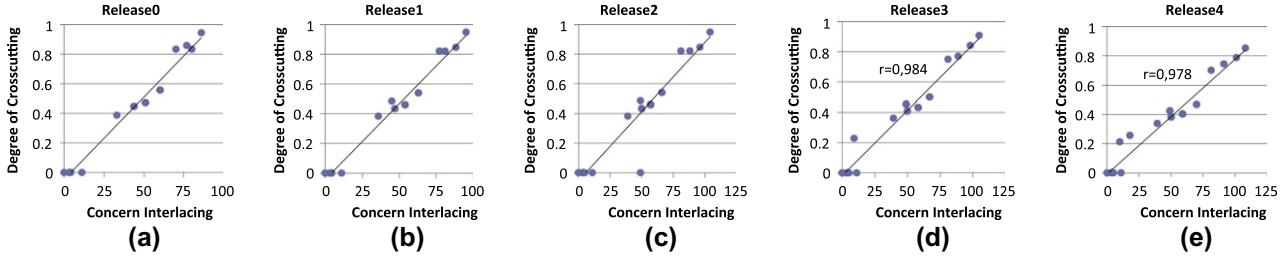

Fig. 23. Correlation between *Concern interlacing* and *Degree of Crosscutting* in the HealthWatcher system.

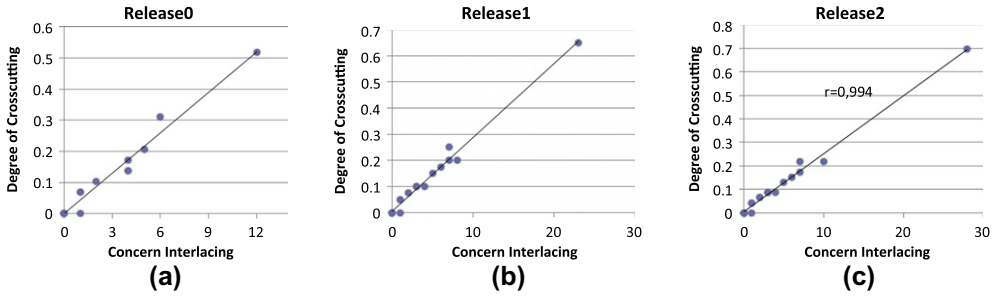

Fig. 24. Correlation between *Concern interlacing* and *Degree of Crosscutting* in the SmartHome system.

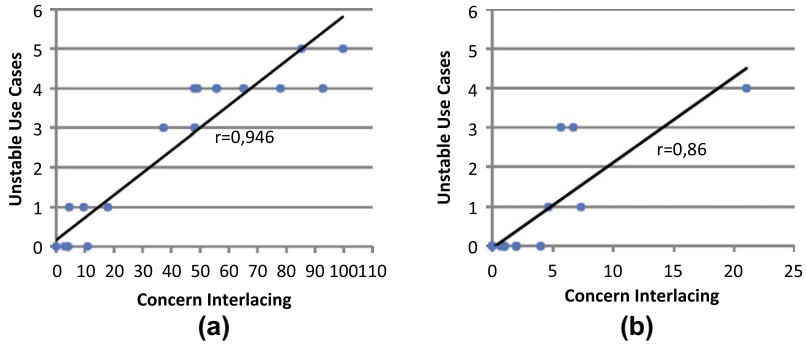

Fig. 25. Correlation *between Concern interlacing* and *Instability* for (a) HealthWatcher and (b) SmartHome systems.

- Considering the relation between *Concern Interlacing* and *Change Impact Set* metric, the correlations of these two metrics with respect to *Degree of crosscutting* one have been also presented. Observe that the latter is correlated with the formers and, thus, *Concern Interlacing* and *Change Impact Set* metrics are also linearly correlated (Fig. 21). Obviously, this conclusion was not surprising since the more dependencies between features, the more impact a change in the feature will have. However, as it has been mentioned in Section 6.2, this conclusion is interesting as previous works only propose the utilization of aspect-oriented techniques to reduce dependencies only between variable and mandatory features [34–36,51,52]. Observe that the results gathered for the other two product lines are also consistent with these assumptions.
- The results obtained in Figs. 21 and 24 also support the conclusions previously extracted in Ref. [17], where the authors analysed modularity in SPL at architectural level. Indeed, the results shown here provide insights of the importance of feature dependency analysis also at earlier development levels, i.e. requirements.
- According to Figs. 22 and 25, the more feature dependencies, the more unstable the software artifacts implementing the features (correlation between *Concern interlacing* and *Instability*). This conclusion is even more important in product line systems, where the products are built by combining different features. Note that products may be built by adding variable functionality to core architecture (in an additive or positive variability approach, according to Ref. [53]). Then, the addition of new features implies too many changes in those core features with a high interlacing (more dependencies). The need for identifying and assessing crosscutting properties is highlighted again since it may predict the more unstable features or those with more dependencies.
- Finally, the point *"Photo"* in Fig. 22 corresponds to the data obtained for the Photo feature. This feature is the one more digressed from the correlation and the reason here is the same as it was explained in Section 6.4: in release 6, part of the functionality implemented in the use cases related to this feature is moved to the use cases related to the Media feature.

The analysis of dependencies raises three final (summarized) conclusions that come to answer the research question RQ3 and, thus, to verify Hypothesises 3.1 and 3.2:

- **C5**: *The higher the Degree of crosscutting of a feature, the more dependencies between this feature and the rest ones.*
- **C6**: *The more dependencies between features, the bigger the impact set of a change is.*
- **C7**: *The more dependencies between features, the less stable the system is.*

7. Related work

In Ref. [2], the authors present an empirical analysis where they study the problem factors that lead to a poor software maintainability. The authors claim that about 66% of total cost of software life cycle is spent in software maintenance tasks. Moreover, they claim that most of the defects found in software system can be traced back to the requirement stages. In that sense, they argue that errors generated in the early phases are the most difficult to detect and more costly to correct [2]. This justifies the need for anticipating empirical maintainability analyses to the early phases of development, as shown in this paper.

There are several works [23,54–58] which have investigated the impact of a change in software artifacts at different abstraction levels. In Ref. [55], the authors propose a dependency taxonomy that identifies the relations existing between requirements-level concerns and their realization at architectural level. These relationships allow the distinction of situations where two requirements are tangled in a same operation of a component, they are tangled in an architectural component (but in different operations) or they are connected by an interface dependency. Based on these kinds of relations the authors identify the requirements with a wider impact set of a change as those more tangled or with more dependencies from/to other requirements. In order to take such decisions, the authors use *concentration* metrics [59]. However, unlike in our analysis of changeability, in this case the authors need at least two phases of the development completed to perform their analysis, i.e. requirements and architecture. Our work allows the anticipation of these results using only the requirements. In Ref. [23], the Crosscutting Pattern summarized in Section 2.1 was also used to analyse change impact in cases of tangling, scattering and crosscutting. This work has been used to establish our impact set metric and, thus, the analysis presented in this paper allows to empirically completing the study presented in Ref. [23]. In Ref. [58] the authors extend traditional change impact analysis techniques (e.g. Refs. [54,56,57] focused on procedural or object-oriented artifacts) to analyse change impact in AspectJ programs. This work is based on the utilization of atomic change representations to identify the semantic of changes due to software evolution in an AspectJ program. Based on these atomic changes, the authors build a call graph used to identify the set of impacted elements. However, this work is just focused on the programming level and it relegates the maintainability analysis to the latest phases of development.

Some works have also dealt with the analysis of stability in software systems [17,19,60]. In Ref. [17] the authors used some metrics to quantitatively and qualitatively assess the stability of SPLs using or not aspect-oriented techniques. The results showed that aspect-orientation improves stability of the features, especially of those variable or alternative. However, there were some cases where the use of aspect-orientation showed limitations and raised worse results, e.g. for modularizing optional features with shared code. The main contribution of this work is the empirical analysis of the improvements obtained by aspect-oriented techniques. However, the analysis is very tied to the programming level and, again, the benefits of assessing modularity and maintainability attributes are relegated to the latest phases of development. In that sense, the work presented here complements this work by anticipating the analysis to the early stages of life cycle. In Ref. [19] the authors show a similar analysis of stability based on the assessment of modularity in object and aspect-oriented versions of a system. This assessment is also carried out by using a set of software metrics. However, in this case, the example system is not a SPL and, again, the analysis is performed at the programming level. The work presented in Ref. [60] complements these previous analyses since evaluates to what extent the presence of crosscutting concerns are harmful for stability in software architectures. In that sense, the authors analyse the changes performed in a system using different architectural decomposition styles. Among other conclusions, the authors claim that aspect-oriented architectures lead, in general, to more stable designs when crosscutting concerns are involved in changes. However, non aspect-oriented architectures behave better for changes involving non crosscutting concerns. Our work could be used, together with the one focused on programming artifacts, for comparing the results presented in this paper and have a way to assess stability throughout a whole development process.

Regarding to the concern-oriented metrics, there are similar metrics suites as those introduced in Refs. [16,27,59,61]. In Ref. [27], Sant'Anna et al. introduce different metrics, namely *Concern Diffusion over Components, Concern Diffusion over Operations* and *Concern Diffusion over Lines of Code*. These metrics allow the

developer to assess the scattering of a concern using different levels of granularity. The authors also define the *Lack of Concern-based Cohesion* to assess the tangling in the system. In Ref. [61], Ducasse et al. introduce four concern measures: *Size*, *Touch*, *Spread* and *Focus*. Wong et al. introduce in Ref. [59] three concern metrics called *Disparity*, *Concentration* and *Dedication*. Eaddy et al. use an adaptation of *Concentration* and *Dedication* metrics for defining two new concern metrics [16]: *Degree of Scattering* and *Degree of Tangling*. Whilst *Degree of Scattering* is defined as the variance of the *Concentration* of a concern over all program elements with respect to the worst case, *Degree of Tangling* is defined as the variance of the *Dedication* of a component for all the concerns with respect to the worst case. However, all the aforementioned metrics are mainly defined to assess modularity using specific development artifacts so that they are focused on specific abstraction levels (design or programming). In Ref. [33], all the aforementioned metrics were empirically compared with our concern metrics. As a conclusion, we observed that our metrics were consistent with them and that they generalize these metrics since they are not tied to any specific development artifact. Moreover, the matrices used in our framework may assist in the visualization and application of the metrics presented in the above publications and the crosscutting product and crosscutting matrices provide specific measures for the degree of crosscutting, exclusive of our metrics suite. In Ref. [62] the authors define the metrics *Number of Features* and *Feature Crosscutting Degree*. This last metric is equivalent to the *Crosscutting Degree of an Aspect* metric, defined in Ref. [63]. However, these metrics may be only applied to aspect-oriented systems and not in legacy systems not modelled using aspect-orientation.

Finally, several works have introduced the benefits of using aspect-oriented techniques to deal with crosscutting features in SPL, reducing dependencies between them [20,34–36,51,52]. For instance, in Ref. [35] the authors propose aspect-oriented techniques to improve flexibility and configurability in product lines, where these characteristics are even more important. They introduce an aspect categorization into three different categories: orthogonal, weakly orthogonal and non-orthogonal aspects. The more orthogonal the aspects are, the fewer dependencies among aspects and components the system has. The work presented in Ref. [20] focuses on the application of aspects in product lines to model variability. The approach uses AspectJ as programming language to implement such features. In Ref. [52], the authors propose a framework for weaving models using aspect oriented techniques. The framework allows modelling variability so that these features may be added to the system by weaving different models. However, most of these aforementioned approaches rely only on the modelling of variable features in SPL using aspect-orientation. But, as it is stated in Ref. [20], common features could be also modelled using aspect-oriented techniques (e.g. aspectual components) if they crosscut to other features. Analogously, variable features need not to be defined always as crosscutting concerns. They may be effectively implemented in modular components if they do not crosscut to other features. Thus, a modularity analysis to identify crosscutting features in SPL is needed. However, all these approaches lack of a process to identify the crosscutting features, especially using an empirical analysis. In that sense, the crosscutting analysis presented here may complement these works providing a way to identify the features with a higher degree of crosscutting.

8. Conclusions

This paper has presented an empirical analysis where we studied the relation between crosscutting and maintainability attributes. This analysis is performed at early development stages so that maintainability is considered from the very beginning of the software life cycle. The empirical analysis is supported by a set of concern metrics that allows the assessment of modularity properties, namely tangling, scattering and crosscutting. These metrics are generically defined and, thus, they are not tied to any specific abstraction level or development artifact. This fact makes the application of the metrics to different domains possible.

Based on the application of the metrics we raised a main research question that was addressed in this paper: is maintainability negatively affected by crosscutting properties at early phases? This question was studied by relating crosscutting properties with changeability and stability ISO/IEC 9126 maintainability attributes. In that sense, the main research question was analysed by dividing it into three different minor questions: (i) Is changeability related with crosscutting properties? (ii) Are crosscutting measurements correlated with stability? (iii) Does crosscutting introduce feature dependencies and how do these dependencies affect to changeability and stability? The three research questions were addressed by empirically observing the different quality attributes in three different software product lines: MobileMedia, HealthWatcher and SmartHome systems.

For the first analysis, a new metric was introduced that assesses the impact set of a change in a feature. The correlation between crosscutting properties and changeability shows that the higher the degree of scattering, tangling or crosscutting for a feature is, the wider impact a change in such a feature has. Moreover, the new metric for change impact provides interesting information about traceability in the form of links affected by the change. This is even more useful with the transition to model-driven software engineering [64] gaining momentum, since this traceability information may be used to build model transformations. In a second analysis, we empirically observed that scattering, tangling and crosscutting negatively affect stability: the higher the degree of scattering, tangling or crosscutting of a feature, the less stable the software artifacts that implement the feature are.

The way of computing stability and changeability was slightly different. On one hand, the analysis of stability was driven by observing changes in target elements. This implies that the analysis was carried out after the changes have occurred (*"a posteriori"*). On the other hand, the analysis of changeability showed how the impact of a change could be anticipated before the change occurs (*"a priori"*). Both analyses complement each other. In fact the results obtained by the analyses have consistently shown that the features with a higher impact set were also those implemented by more unstable use cases (and, of course, with a higher degree of crosscutting). Moreover, the results obtained in both analyses allow the developer to anticipate maintainability decisions. Note that once we have observed that crosscutting metrics are correlated with changeability and stability measurements, the former may be used to anticipate the latter in future analyses, without the need of computing changeability or stability.

The third analysis has also demonstrated that crosscutting introduces dependencies between features which are also harmful for changeability and stability. In order to assess these dependencies, a concern interlacing metric was used and adapted to be used at the requirements level showing that concern interlacing is also correlated to change impact and instability. This result is particularly interesting as it shows how concern analysis reveals how certain instability or changeability indicators cannot rely only on conventional coupling metrics. The latter ones only capture dependencies between modules that are based on the syntactical description of the modules and their interactions.

By using SPLs as our target cases, the three analyses support the need for identifying crosscutting features independently of their nature (mandatory or variable). This conclusion is contrary to the works that propose modelling only variable features using aspect-oriented techniques. The results showed that mandatory

crosscutting features also negatively affect the quality of the system, being also candidate to be modularized using aspect-oriented or other advanced modularity techniques. On the contrary, variable features may be also implemented in modular components and they do not need to be always defined as crosscutting concerns. Our study revealed that the decomposition of use cases might have a considerable effect on the architectural and detailed decomposition in SPL-like systems, where modularity is a key driving force. In these systems, it is natural that early modules, such as feature models and use cases, influence the choice of modules (and their interfaces) in the design. Therefore, our study should be replicated in the context of our types of systems in order to confirm or refute our findings.

As future work, we plan to perform several empirical studies to compare the results obtained by our metrics at requirements level with those obtained at different levels of abstraction (e.g. architectural, design or implementation level). Also, the application of the metrics at source-code level would allow us to compare our results with the obtained by other studies where authors analyse instability at this level (e.g. Refs. [17,19]). By this analysis we could test different hypotheses, such as, whether similar properties of crosscutting concerns are found to be indicators of maintainability problems or what probabilities of early crosscutting measurements lead to false warnings (i.e. false positives or negatives) at source-code level. In these analyses we may also utilize an aspect-oriented version of the system assessed to check the improvements obtained by the utilization of different paradigms.

Acknowledgments

This work has been supported in part by the European CommissionGrant IST-2-004349: European Network of Excellence on AOSD (AOSD-Europe), by MEC and FEDER under Contracts TIN2008-02985 and TIN2011-27340, by University of Extremadura (research initiation program).

References

[1] D.D. Galorath, Software total ownership costs: development is only job one, Software Tech News 11 (3) (2008).
[2] J. Chen, S. Huang, An empirical analysis of the impact of software development problem factors on software maintainability, Journal of Systems and Software 82 (6) (2009) 981–992.
[3] L. Erlikh, Leveraging legacy system dollars for e-business, IEEE IT Professional 2 (3) (2000) 17–23.
[4] V. Hung, Software Maintenance. Connexions, July 2007. <http://cnx.org/content/m14719/latest/>.
[5] C.F. Kemerer, Software complexity and software maintenance: a survey of empirical research, Annals of Software Engineering (1995) 1–22.
[6] C.F. Kemerer, Progress, obstacles, and opportunities in software engineering economics, Communications of the ACM 41 (8) (1998) 63–66.
[7] J. Koskinen, Software Maintenance Costs. <http://www.cs.jyu.fi/~koskinen/smcosts.htm>.
[8] D. Notkin, Software, software engineering and software engineering research: some unconventional thoughts, Journal of Computer Science and Technology 24 (2) (2009) 189–197.
[9] S.W.L. Yip, T. Lam, A software maintenance survey, in: Proceedings of the 1st Asia–Pacific Software Engineering Conference, 1994, pp. 70–79.
[10] ISO, Software Engineering – Product quality – Part 1: Quality Model, ISO/IEC 9126-1, International Organization of Standardization, 2001.
[11] N.F. Schneidewind, Body of knowledge for software quality measurement, Computer 35 (2) (2002) 77–83.
[12] O. Balci, Verification, validation and certification of modeling and simulation applications, in: Proceedings of the 2003 Simulation Conference, vol. 1, 2003, pp. 150–158.
[13] A.B. Binkley, S.R. Schach, Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures, in: Proceedings of the 20th International Conference on Software Engineering, Kyoto, Japan, April 19–25, 1998.
[14] P. Tonella, Concept analysis for module restructuring, IEEE Transactions on Software Engineering 27 (4) (2001) 351–363.
[15] G. Kiczales, J. Lamping, A. Mendhekar, C. Meada, C. Lopes, J. Loingtier, J. Irwin, Aspect-Oriented Programming, in: Proceedings of the 11th European Conference on Object-Oriented Programming, ECOOP, Jyväskylä, Finland, 1997, pp. 220–242.
[16] M. Eaddy, T. Zimmermann, K. Sherwood, V. Garg, G. Murphy, N. Nagappan, A. Aho, Do crosscutting concerns cause defects?, IEEE Transactions on Software Engineering 34 (4) (2008) 497–515.
[17] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Filho, F. Dantas, Evolving software product lines with aspects: an empirical study on design stability, in: Proceedings of the 30th International Conference on Software Engineering, ICSE, Leipzig, Germany, 2008, pp. 261–270.
[18] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. Staa, Modularizing design patterns with aspects: a quantitative study, Transactions on Aspect-Oriented Software Development I, LNCS, vol. 3880, Springer, 2006.
[19] P. Greenwood, T. Bartolomei, E. Figueiredo, M. Dosea, A. Garcia, N. Cacho, C. Sant'Anna, S. Soares, P. Borba, U. Kulesza, A. Rashid, On the impact of aspectual decompositions on design stability: an empirical study, in: Proceedings of 21st European Conference on Object-Oriented Programming, Berlin, Germany, 2007, pp. 176–200.
[20] K. Lee, K. Kang, M. Kim, S. Park, Combining feature-oriented analysis and aspect-oriented programming for product line asset development, in: Proceedings of the 10th International Software Product Line Conference (SPLC), Baltimore, USA, 2006, pp. 103–112.
[21] F. Ferrari, R. Burrows, O. Lemos, A. Garcia, E. Figueiredo, N. Cacho, F. Lopes, N. Temudo, L. Silva, S. Soares, A. Rashid, P. Masiero, T. Batista, J. Maldonado, An exploratory study of fault-proneness in evolving aspect-oriented programs, in: Proceedings of the 32th International Conference on Software Engineering (ICSE'10), Cape Town, South Africa, May 2010.
[22] R. Burrows, F. Ferrari, O. Lemos, A. Garcia, F. Taanï, The impact of coupling on the fault-proneness of aspect-oriented programs: an empirical study, in: Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE), San Jose, USA, November 2010.
[23] K. van den Berg, Change impact analysis of crosscutting in software architectural design, in: Proceedings of the Workshop on Architecture-Centric Evolution at 20th ECOOP, Nantes, France, 2006.
[24] F. Filho, N. Cacho, R. Ferreira, E. Figueiredo, A. Garcia, C. Rubira, Exceptions and aspects: the devil is in the details, in: Proceedings of FSE-14, International Conference on Foundations on Software Engineering, November 2006.
[25] K. van den Berg, J. Conejero, J. Hernández, Analysis of Crosscutting in Early Software Development Phases based on Traceability, Transactions on Aspect-Oriented Software Development III, LNCS, vol. 4620, Springer, 2007.
[26] A. Garcia, C. Lucena, Taming heterogeneous agent architectures, Communications of the ACM 51 (5) (2008) 75–81.
[27] C. Sant'Anna, E. Figueiredo, A. Garcia, C. Lucena, On the modularity of software architectures: a concern-driven measurement framework, in: Proceedings of the 1st European Conference on Software Architecture, ECSA, Madrid, Spain, 2007, pp. 207–224.
[28] E. Baniassad, P. Clements, J. Araújo, A. Moreira, A. Rashid, B. Tekinerdogan, Discovering early aspects, IEEE Software 23 (1) (2006) 61–70.
[29] E. Figueiredo, B. Silva, C. Sant'Anna, A. Garcia, J. Whittle, D. Nunes, Crosscutting patterns and design stability: an exploratory analysis, in: Proceedings of the 17th International Conference on Program Comprehension (ICPC), Vancouver, Canada, 2009.
[30] A. Hovsepyan, R. Scandariato, S. van Baelen, Y. Berbers, W. Joosen, From aspect-oriented models to aspect-oriented code? The maintenance perspective, in: Proceedings of the 9th Aspect-Oriented Software Development Conference (AOSD), Rennes and St. Malo, France, 2010, pp. 85–96.
[31] E. Figueiredo, C. Sant'Anna, A. Garcia, T. Bartolomei, W. Cazzola and A. Marchetto, On the maintainability of aspect-oriented software: a concern measurement framework, in: Proceedings of 12th European Conference on Software Maintenance and Reengineering, Athens, Greece, 2008, pp. 183–192.
[32] C. Elsner, L. Fiege, I. Groher, M. Jäger, C. Schwanninger, M. Völter, Ample Project, Deliverable d5.3 – Implementation of First Case Study: Smart Home, Technical Report, 2008.
[33] J. Conejero, E. Figueiredo, A. Garcia, J. Hernández, E. Jurado, Early crosscutting metrics as predictors of software instability, in: Proceedings of the 47th International Conference Objects, Models, Components, Patterns, TOOLS Europe, LNBIP 33, Zurich, Switzerland, 2009, pp. 136–156.
[34] M. Voelter, I. Groher, Product line implementation using aspect-oriented and model-driven software development, in: Proceedings of the 11th International Software Product Line Conference, Kyoto, Japan, 2007, pp. 233–242.
[35] A. Colyer, A. Rashid, G. Blair, On the Separation of Concerns in Program Families, Lancaster University Technical Report Number: COMP-001-2004, 2004.
[36] M. Griss, Implementing product-line features by composing aspects, in: Proceedings of the 1st International Software Product Line Conference (SPLC), Denver, USA, 2000, pp. 271–288.
[37] E. Figueiredo, I. Galvao, S. Khan, A. Garcia, C. Sant'Anna, A. Pimentel, A. Medeiros, L. Fernandes, T. Batista, R. Ribeiro, P. van den Broek, M. Aksit, S. Zschaler, A. Moreira, Detecting architecture instabilities with concern traces: an exploratory study, in: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture, Cambridge, UK, 2009.
[38] K. Kang, S. Cohen, J. Hess, W. Novak, A. Spencer, Feature Oriented Domain Analysis (FODA). Feasibility Study, Carnegie Mellon University Technical Report CMU/SEI-90-TR-21, 1990.
[39] J. Conejero, J. Hernandez, E. Jurado, P.J. Clemente, R. Rodríguez, The crosscutting pattern: analyzing the modularity of software product lines, in: Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston, USA, 2009.

[40] M. Alférez, J. Santos, A. Moreira, A. Garcia, U. Kulesza, J. Araújo, V. Amaral, Multi-view composition language for software product line requirements, in: Proceedings of 2nd International Conference on Software Language Engineering, 2009, pp. 103–122.
[41] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, A. Rummler, An exploratory study of information retrieval techniques in domain analysis, in: Proceedings of 12th International Software Product Line Conference (SPLC), 2008, pp. 67–76.
[42] AMPLE (Aspect-Oriented, Model Driven Product Line Engineering) European Project. <http://ample.holos.pt/>.
[43] T. Young, Using AspectJ to Build a Software Product Line for Mobile Devices, MSc dissertation, Univ. of British Columbia, 2005.
[44] L.C. Briand, S. Morasca, V.R. Basili, Defining and validating measures for object-based high-level design, IEEE Transactions on Software Engineering (1999) 722–743.
[45] E. Figueiredo, A. Garcia, M. Maia, G. Ferreira, C. Nunes, J. Whittle, On the impact of crosscutting concern projection on code measurement, in: Proceedings of the 10th International Conference on Aspect-Oriented Software Development (AOSD'11), Porto de Galinhas, Brazil, March 2011.
[46] C. Nunes, A. Garcia, E. Figueiredo, C. Lucena, Revealing mistakes on concern mapping tasks: an experimental evaluation, in: Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR), Oldenburg, Germany, 2011.
[47] M. Revelle, T. Broadbent, D. Coppit, Understanding concerns in software: insights gained from two case studies, in: Proceedings of the IWPC 2005, pp. 23–32.
[48] Mining Early Aspects based on Syntactical and Dependency Analyses, 2010. <http://www.unex.es/eweb/earlyaspectmining/>.
[49] J. Taylor, An Introduction to Error Analysis, second ed., The Study of Uncertainties in Physical Measurements, University Science Books, 1997.
[50] D. Kelly, A study of design characteristics in evolving software using stability as a criterion, IEEE Transactions on Software Engineering 32 (2006) 315–329.
[51] N. Loughran, A. Sampaio, A. Rashid, From requirements documents to feature models for aspect oriented product line implementation, in: Proceedings of Workshop on MDD for Product Lines at MODELS, Montego Bay, Jamaica, 2005, pp. 262–271.
[52] B. Morin, O. Barais, J.M. Jézéquel, Weaving aspect configurations for managing system variability, in: Proceedings of Second International Workshop on Variability Modelling of Software-intensive Systems, Essen, Germany, 2008, pp. 53–62.
[53] I. Groher, M. Völter, Aspect-oriented model-driven software product line engineering, Transactions on Aspect-Oriented Software Development VI 5560 (2009) 111–152.
[54] R.S. Arnold, Software Change Impact Analysis, IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.
[55] S.S. Khan, S. Lock, Concern tracing and change impact analysis: an exploratory study, in: Proceedings of the Early Aspects Workshop at ICSE, Vancouver, Canada, 2009, pp. 44–48.
[56] X. Ren, F. Shah, F. Tip, B. Ryder, O. Chesley, Chianti: a tool for change impact analysis of Java programs, in: Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Vancouver BC, Canada, 2004, pp. 432–448.
[57] M. Stoerzer, B.G. Ryder, X. Ren, F. Tip, Finding failure-inducing changes in java programs using change classification, in: Proceedings of 14th International Symposium on Foundations of Software Engineering, ACM, Portland, USA, 2006, pp. 57–68.
[58] S. Zhang, J. Zhao, Change Impact Analysis for AspectJ Programs, Technical Report SJTU-CSE-TR-07-01, Center for Software Engineering, SJTU, 2007.
[59] W. Wong, S. Gokhale, J. Horgan, Quantifying the closeness between program components and features, Journal of Systems and Software (2000) 87–98.
[60] A. Molesini, A. Garcia, C. von Flach Garcia Chavez, T.V. Batista, Stability assessment of aspect-oriented software architectures: A quantitative study, Journal of Systems and Software 83 (5) (2010) 711–722.
[61] S. Ducasse, T. Girba, A. Kuhn, Distribution map, in: Proceedings of the International Conference on Software Maintenance, ICSM, Philadelphia, USA, 2006.
[62] R. Lopez-Herrejon, S. Apel, Measuring and characterizing crosscutting in aspect-based programs: basic metrics and case studies, in: Proceedings of the International Conference on Fundamental Approaches to Software Engineering, Braga, Portugal, 2007, pp. 422–437.
[63] M. Ceccato, P. Tonella, Measuring the effects of software aspectization, in: Proceedings of the 1st Workshop on Aspect Reverse Engineering, Delft University of Technology, the Netherlands, 2004.
[64] M. Alférez, U. Kulesza, A. Sousa, J. Santos, A. Moreira, J. Araújo, V. Amaral, A model-driven approach for software product lines requirements engineering, in: Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, 2006.