Early Crosscutting Metrics as Predictors of Software Instability

José M. Conejero[1], Eduardo Figueiredo[2], Alessandro Garcia[3], Juan Hernández[1], and Elena Jurado[1]

[1] Quercus Software Engineering Group, University of Extremadura, Spain
[2] Computing Department, Lancaster University, United Kingdom
[3] Informatics Department,Pontifical Catholic University of Rio de Janeiro, Brazil
{chemacm,juanher,elenajur}@unex.es,
e.figueiredo@lancaster.ac.uk, afgarcia@inf.puc-rio.br

Abstract. Many researchers claim that crosscutting concerns, which emerge in early software development stages, are harmful to software stability. On the other hand, there is a lack of effective metrics that allow software developers to understand and predict the characteristics of "early" crosscutting concerns that lead to software instabilities. In general, existing crosscutting metrics are defined for specific programming languages and have been evaluated only against source-code analysis, when major design decisions have already been made. This paper presents a generic suite of metrics to objectively quantify key crosscutting properties, such as scattering and tangling. The definition of the metrics is agnostic to particular language intricacies and can be applied to all early software development artifacts, such as usecases and scenarios. We have performed a first stability study of crosscutting on requirements documents. The results pointed out that early scattering and crosscutting have, in general, a strong correlation with major software instabilities and, therefore, can help developers to anticipate important decisions regarding stability at early stages of development.

Keywords: Concern Metrics, Modularity, Stability, Requirements Engineering.

1 Introduction

There is growing empirical evidence that software stability is often inversely proportional to the presence of crosscutting concerns [8, 9, 12, 13]. A software system is stable if, when observed over two or more versions of the software, the differences between its quality measures are insignificant [14]. It is claimed that crosscutting concerns often lead to harmful software instabilities, such as increased modularity anomalies [9, 13] and higher number of introduced faults [8]. The problem of crosscutting concerns is usually described in terms of scattering and tangling [3]. Scattering occurs when the realization of a concern is spread over the software modules whilst tangling occurs when the concern realization is mixed with other concerns in a module.

Most of the crosscutting concerns manifest in early development artifacts, such as requirements descriptions [3] and architectural models [18, 11], due to their

M. Oriol and B. Meyer (Eds.): TOOLS EUROPE 2009, LNBIP 33, pp. 136–156, 2009.

widely-scoped influence in software decompositions. They can be observed in every kind of requirements and design representations, such as usecases and component models [3, 18, 11, 2]. Over the last years, aspect-oriented software development (AOSD) [15] has emerged with the goal of supporting improved modularity and stability of crosscutting concerns throughout the software lifecycle. However, the use of aspect-oriented decompositions cannot be straightforwardly applied without proper assessment mechanisms for early software development stages. This became more evident according to recent empirical studies of AOSD based on source-code analysis (e.g. [9, 12, 13]). First, not all types of crosscutting concerns were found to be harmful to design stability. Second, there are certain measurable characteristics of crosscutting concerns that seem to recurrently lead to design instabilities [9, 13].

However, there is little or no knowledge about how characteristics of crosscutting concerns, observable in early artefacts, are correlated with design instabilities. Most of the systematic studies of crosscutting concerns (e.g. [8, 9, 12, 13]) concentrate on the analysis of source code, when architectural decisions have already been made. Even worse, a survey of existing crosscutting metrics has pointed out that they are defined in terms of specific OO and aspect-oriented (AO) programming languages [10]. However, inferring design stability after investing in OO or AO implementations can be expensive and impractical. In addition, crosscutting metrics defined for early design representation are very specific to certain models, such as component-and-connector models [18]. These metrics are overly limited as many crosscutting concerns are visible in certain system representations, but not in others [10].

In this context, the major contributions of this paper are threefold. First, it presents a language-agnostic metrics suite for early quantification of crosscutting (Section 3). This is particular useful with the transition to model-driven software engineering gaining momentum, where analysis of crosscutting concerns should also be undertaken in early system representations. The definition of the metrics is based on a conceptual framework (Section 2) that is independent of specific requirements and architectural models. Second, canonical instantiations of the crosscutting metrics are given for usecases. Third, we also present a first exploratory study investigating the correlation of early crosscutting measures and design instabilities (Section 4). The results obtained help developers to anticipate important decisions regarding stability at early stages of development. Finally, Sections 5 and 6 discuss related work and conclude this paper.

2 Characterizing and Identifying Crosscutting Concerns

The operational definitions of concern-oriented metrics need to be conceived in an unambiguous manner. However, concern properties, such as crosscutting and scattering, are often not formally defined. Our proposed concern-oriented metrics (Section 3) are based on a previously-defined conceptual framework [3] that supports the characterization and identification of *crosscutting*. Section 2.1 describes the key definitions of this conceptual framework. Section 2.2 illustrates its instantiation to requirements-level artefacts of a software system.

2.1 A Conceptual Framework for Crosscutting

Our previous work [3] presented a conceptual framework where a formal definition of crosscutting was provided. This framework is based on the study of matrices that represent particular features of a traceability relationship between two different domains. These domains, generically called Source and Target, could be, for example, concerns and usecases respectively or, in a different situation, design modules and programming artefacts. We used the term *Crosscutting Pattern* to denote this situation (see Fig. 1).

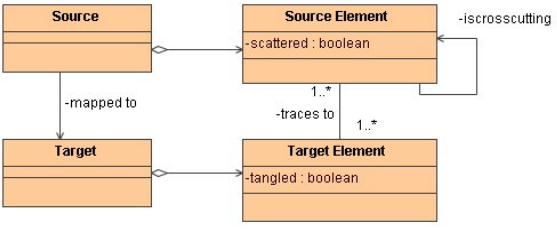


Fig. 1. Abstract meta-model of the crosscutting pattern

The relationship between Source and Target can be formalized by two functions *f* and *g*, where *g* can be considered as a *special* inverse function of *f*.

Let f: *Source* $\longrightarrow \mathscr{P}$ *(Target)* and g: *Target* $\longrightarrow \mathscr{P}$ *(Source)* be these functions defined by:

$\forall s \in Source, f(s) = \{t \in Target$ *:there exists a trace relation between s and t* $\}$

$\forall t \in Target, g(t) = \{s \in Source$ *: there exists a trace relation between s and t*$\}$.

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

Definition 1. [Scattering] We say that an element s ∈ Source is scattered if $card(f(s)) > 1$, where card 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 ∈ Target is tangled if $card(g(t))>1$. *Tangling occurs when, in a mapping between source and target, a target element is related to multiple source elements.*

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

Definition 3. [Crosscutting] Let s1, s2 ∈ Source, s1 ≠ s2, we say that s1 crosscuts s2 if $card(f(s1)) > 1$ and $\exists t \in f(s1)$: $s2 \in g(t)$. *Crosscutting occurs when, in a mapping between source and target, a source element is scattered over target elements and where in at least one of these target elements, some other source element is tangled.* In [6] we formally compared our definition with others existing in the literature, such as [17].

2.2 Identification of Crosscutting

In [3], we defined the dependency matrix to represent function *f*. An example of dependency matrix with five source and six target elements is shown in Table 1. A 1 in a cell means that the target element of the corresponding column contributes to or addresses the source element of the corresponding row. Based on this matrix, two different matrices called scattering matrix and tangling matrix are derived.

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

The crosscutting product matrix is obtained through the multiplication of scattering matrix and tangling matrix. The crosscutting product matrix shows the quantity of crosscutting relations (Table 2) and is used to derive the final crosscutting matrix (Table 3). A cell in the final crosscutting matrix denotes the occurrence of crosscutting, but abstracts the quantity of crosscutting. More details about the conceptual framework and the matrix operations can be found in [3].

Table 2. 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 3. 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

3 Concern-Oriented Metrics for Early Development Assessment

In this section, we propose a concern-oriented metric suite based on the framework presented in Section 2. These metrics allow developers to quantify the degree of scattering, tangling, and crosscutting at earlier development stages of a software system, such as requirements and architecture modeling. The metrics defined are based on the relation between source and target domains represented by the crosscutting pattern. In order to illustrate the metrics, we rely on requirements descriptions of a running example (MobileMedia).

3.1 The MobileMedia System

The MobileMedia [9] is a product line system built to allow the user of a mobile device to perform different options, such as visualizing photos, playing music or videos,

and sending photos by SMS (among other concerns). It has about 3 KLOC. The system has been built as a product line in 8 different releases. In this section we show a simple usecase diagram (Fig. 2) which corresponds to a part of the usecase diagram used for release 0 in the MobileMedia system. In this part of the diagram, the actions for adding albums and photos to the system are implemented. These actions include the option for providing a label. Some actions for recording the data into a persistent storage are also included. Then, we consider that four main concerns are involved in this part of the system: album, photo, label, and persistence.

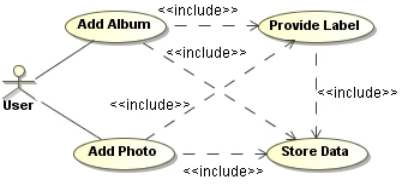


Fig. 2. Simplification of the usecase diagram for release 0 in MobileMedia

In Table 4, we show a simplified description of the usecases shown in Fig. 2. We have shadowed in light and dark grey colors the flows and relations in these usecases corresponding to Label and Persistence concerns, respectively. Although there are other two concerns involved in the example (album and photo), we have not shadowed any flow related to them to keep the example clear.

Table 4. Usecase descriptions for Add Album, Add Photo and Provide Label usecases

Usecase: Add Album	**Usecase:** Add Photo	**Usecase:** Provide Label	**Usecase:** Store Data
Actor: Mobile Phone (system) and User **Description:** The user can store (add) an album to the mobile phone **Pre/Posconditions: (…)** **Basic flows:** 1. (add)The user selects the option to add an album. 2. (label) User provides label to the new created album 3. (saved) A new album is available 4. (listing) The list of photos is displayed **Includes:** 1.Provide Label usecase 2. Store Data usecase	**Actor:** Mobile Phone (system) and User **Description:** User can store (add) a photo in an album available **Pre/Posconditions: (…)** **Basic flows:** 1. (select) The user selects an album to store the photo. 2. (add)The user selects the option to add photo. 3. (path) User provides the path for uploading the photo. 4. (label) User assigns a label to the photo. 5. (saved) The new photo is stored in the album. **Includes:** 1. Provide Label usecase 2. Store Data usecase	**Actor:** Mobile Phone (system) and User **Description:** The user provides label for the photo and album **Pre/Posconditions: (…)** **Basic flows:** 1. (label) The users provides a name for a photo or an album 2. (save) The edited label is saved **Includes:** 1. Store Data usecase	**Actor:** Mobile Phone (system) **Description:** The data of a photo or an album must be stored into the device storage **Pre/Posconditions: (…)** **Basic flow:** 1. (space) The device select a space in the storage 2. (save) The data are saved in the storage

In order to clarify the metrics presented in next sections, each metric is illustrated using the previously described example. In particular, we show the values of each metric for the partial usecase diagram (Fig. 2) and usecase descriptions (Table 4).

3.2 Metrics for Scattering

According to Definition 1 in Section 2.1, *Nscattering* of a source element s_k as the number of 1's in the corresponding row (k) of the dependency matrix:

$$NScattering\ (s_k) = \sum_{j=1}^{|T|} dm_{kj} \tag{1}$$

where |T| is the number of target elements and dm_{kj} dm_{kj} is the value of the cell [k,j] of the dependency matrix. We may also express this metric according to the functions defined in Section 2.1 as *NScattering* (s_k) = *card* $\{t \in Target : f'(s_k)=t\}$, i.e. *card*$(f(s_k))$. This metric measures how scattered a concern is. In the example shown in Section 3.1, the *NScattering* for Label and Persistence concerns is 3 and 4, respectively. As we can see in Table 4, all the usecases descriptions have some flows or relations shadowed with dark grey (related to Persistence), however there are only 3 usecases with light grey (related to Label). Then we may assure that the Persistence concern is more scattered than Label.

This *NScattering* metric can be normalized in order to obtain a value between 0 and 1. Then, we define *Degree of scattering* of the source element s_k as:

$$Degree\ of\ scattering\ (s_k) = \begin{cases} \dfrac{\sum_{j=1}^{|T|} dm_{kj}}{|T|} & if \ \sum_{j=1}^{|T|} dm_{kj} > 1 \\ 0 & if \ \sum_{j=1}^{|T|} dm_{kj} = 1 \end{cases} \tag{2}$$

The closer to zero this metric for a source element (i.e., a concern), the better encapsulated the source element. Conversely, when the metric has a value closer to 1, the source element is highly spread over the target elements and it is worse encapsulated. This metric could have been also defined in terms of the scattering matrix (instead of dependency matrix).

In order to have a global metric for how much scattering the system's concerns are, we define the concept of *Global scattering (Gscattering)* which is obtained just by calculating the average of the *Degree of scattering* values for each source elements:

$$Gscattering = \frac{\sum_{i=1}^{|S|} Degree\ of\ scattering(s_i)}{|S|} \tag{3}$$

where |S| is the number of analyzed source elements. In our particular case, it represents the number of concerns of interest in the system.

3.3 Metrics for Tangling

Similarly to Nscattering for scattering, we also defined the *Ntangling* metric for the target element t_k, where |S| is the number of source elements and dm_{jk} is the value of the cell [i,k] of the dependency matrix:

$$Ntangling\ (t_k) = \sum_{i=1}^{|S|} dm_{ik} \tag{4}$$

Again, according to the functions introduced in Section 2.1, Ntangling (tk) = card {s ϵ Source : f'(s)=tk}, i.e., card(f(tk)). Then, this metric measures the number of source elements addressed by a particular target element. In the MobileMedia example (Section 3.1), the NTangling for the Add Album and Store Data usecases are 2 and 1, respectively. As we can see in Table 4, Store Data usecase is only shadowed in dark grey color so that it just addresses the Persistence concern (whilst Add Album addresses Persistence and Label).

We follow the same steps performed for the scattering metrics and to define two tangling metrics: *Degree of tangling* and *Gtangling*. These metrics represent the normalized tangling for the target element t_k and the global tangling, respectively:

$$Degree\ of\ tangling\ (t_k) = \begin{cases} \frac{\sum_{i=1}^{|S|} dm_{ik}}{|S|} & if\ \sum_{i=1}^{|S|} dm_{ik} > 1 \\ 0 & if\ \sum_{i=1}^{|S|} dm_{ik} = 1 \end{cases} \tag{5}$$

$$Gtangling = \frac{\sum_{j=1}^{|T|} Degree\ of\ tangling(t_j)}{|T|} \tag{6}$$

Like *Degree of scattering*, the *Degree of tangling* metric may take values between 0 and 1, where the value 0 represents a target element addressing only one source element. The number of source elements addressed by the target element increases as the metric is closer to 1.

3.4 Metrics for Crosscutting

Finally, this section defines three metrics for crosscutting: *Crosscutpoints, NCrosscut* and *Degree of crosscutting*. These metrics are extracted from the crosscutting product matrix and the crosscutting matrix of the framework presented in Section 2.1.

The *Crosscutpoints* metric is defined for a source element s_k as the number of target elements where s_k is crosscutting to other source elements. This metric is calculated from the crosscutting product matrix (remember that this matrix is calculated by the product of scattering and tangling matrices). The *Crosscutpoints* metric for s_k corresponds to the value of the cell in the diagonal of the row k (cell [k,k] or $ccpm_{kk}$).

$$Crosscutpoints(s_k) = ccpm_{kk} \tag{7}$$

According to our running example of Section 3.1, we can see that *Crosscutpoints* metric for Persistence has a value of 3. Note that there are three usecases descriptions

(Table 4) which are shadowed with both light and dark color (Add Album, Add Photo and Provide Label). Then, the Persistence and Label concerns cut across each other in these usecases.

The *NCrosscut* metric is defined for the source element s_k as the number of source elements crosscut by s_k. The *NCrosscut* metric for s_k is calculated by the addition of all the cells of the row k in the crosscutting matrix:

$$NCrosscut(s_k) = \sum_{i=1}^{|S|} ccm_{ki} \tag{8}$$

In our example, *NCrosscut* for Persistence is 1 since it is crosscutting just to the Label concern. Finally, the two crosscutting metrics above allow us to define the *Degree of crosscutting* metric of a source element s_k. Note that, *Degree of crosscutting* is normalized between 0 and 1, so that those source elements with lower values for this metric are the best modularized.

$$Degree\ of\ crosscutting(s_k) = \frac{Crosscutpoints(s_k) + Concerns\ crosscut(s_k)}{|S| + |T|} \tag{9}$$

We summarize all our metrics in Table 5. In this table we show the definition of each metric and the relation with the matrices used by the crosscutting pattern.

Table 5. Summary of the concern-oriented metrics based on the Crosscutting Pattern

Metric	Definition	Relation with matrices	Calculation
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)	$= \sum_{j=1}^{\|T\|} dm_{kj}$
Degree of scattering (s_k)	Normalization of NScattering (s_k) between 0 and 1		$= \begin{cases} \frac{\sum_{j=1}^{\|T\|} dm_{kj}}{\|T\|} & if \ \sum_{j=1}^{\|T\|} dm_{kj} > 1 \\ 0 & if \ \sum_{j=1}^{\|T\|} dm_{kj} = 1 \end{cases}$
Gscattering (s_k)	Average of Degree of scattering of the source elments		$= \frac{\sum_{i=1}^{\|S\|} Degree\ of\ scattering(s_i)}{\|S\|}$
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)	$= \sum_{i=1}^{\|S\|} dm_{ik}$
Degree of tangling (t_k)	Normalization of NTangling (t_k) between 0 and 1		$= \begin{cases} \frac{\sum_{i=1}^{\|S\|} dm_{ik}}{\|S\|} & if \ \sum_{i=1}^{\|S\|} dm_{ik} > 1 \\ 0 & if \ \sum_{i=1}^{\|S\|} dm_{ik} = 1 \end{cases}$
Gtangling (t_k)	Average of Degree of tangling of the target elments		$= \frac{\sum_{j=1}^{\|T\|} Degree\ of\ tangling(t_j)}{\|T\|}$
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)	$= ccpm_{kk}$
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)	$= \sum_{i=1}^{\|S\|} ccm_{ki}$
Degree of crosscutting (s_k)	Addition of the two last metrics normalized between 0 and 1		$= \frac{ccpm_{kk} + \sum_{i=1}^{\|S\|} ccm_{ki}}{\|S\| + \|T\|}$

4 Evaluation and Discussion

In this section we present a first empirical study using the metrics presented in Section 3. The main goal of the analysis presented is to observe how early crosscutting metrics may help in predicting instabilities. Then, these metrics provide indications that the crosscutting concerns identified should be modularized, e.g., using aspects [2]. This hypothesis is tested by using a double validation: (1) an internal validation of the crosscutting metrics with respect to their ability of accurately quantifying certain crosscutting properties, and (2) an external validation of the crosscutting metrics in terms of their predictability of software stability [14].

4.1 Survey of Related Metrics

In this section we briefly discuss several concern-oriented metrics [7, 8, 19, 22] which have been used in our internal and external validations. These metrics are summarized in Table 6. Unlike the metrics presented in this paper, the metrics summarized in Table 6 are mainly defined in terms of specific design or implementation artefacts. Accordingly, we have adapted these metrics to the requirements level in order to compare the results obtained by our metrics with those obtained by the rest of metrics (in Section 4.2). The adaptation has mainly consisted of a change in the target element

Table 6. Survey of metrics defined by other authors

Authors		Metric	Definition
Sant'Anna et al.	[19]	*Concern Diffusion over Components (CDC)*	It counts the number of components addressing a concern.
		Concern Diffusion over Operations (CDO)	It counts the number of methods and advices addressing a concern.
		Concern Diffusion over Lines of Code (CDLOC)	It counts the number of lines of code related to a particular concern.
	[18]	*Lack of Concern Cohesion (LOCC)*	It counts the number of concerns addressed by the assessed component.
		Component-level Interlacing Between Concerns (CIBC)	It counts the number of other concerns with which the assessed concerns share at least a component.
Ducasse et al. [7]		*Size*	It counts the number of internal members of classes (methods or attributes) associated to a concern.
		Spread	It counts the number of modules (classes or components) related to a particular concern
		Focus	It measures the closeness between a module and a property or concern
		Touch	It assesses the relative size of a concern or a property (Size of property divided into total size of system)
Wong et al. [22]		*Concentration (CONC)*	It measures how much a concern is concentrated in a component
		Dedication (DEDI)	It quantifies how much a component is dedicated to a concern.
		Disparity (DISP)	It measures how many blocks related to a particular property (or concern) are localised in a particular component
Eaddy et al. [8]		*Degree of scattering (DOS)*	It is defined as the variance of the Concentration of a concern over all program elements with respect to the worst case
		Degree of tangling (DOT)	It is defined as the variance of the Dedication of a component for all the concern with respect to the worst case

used for the different measures. For example, where a metric was defined for measuring concepts using components or classes, we have adapted the metric to the requirements domain by using usecases as the target entity. We have also taken into account the different granularity levels used by the metrics. For instance, there are some metrics which use operations or lines of code (instead of components or classes) as the target entity. In order to adapt these metrics, we changed operations by usecase flows or steps, since flows represent a finer granularity level (similar to operations or lines of code) than usecases. Then, the granularity level used at requirements keeps consistent with the used by the original metrics.

4.2 Internal Validation

In this section we show the results obtained by calculating our metrics to the MobileMedia system [9]. The application has been used for performing different analyses in software product lines mainly at architectural and programming level [9]. Our work complements those previous analyses since we focus on modularity at the requirements level. The reason for calculating the metrics at this level is to identify the concerns with a higher *Degree of crosscutting* (a poor modularity) as soon as possible. Then, the developer may anticipate important decisions regarding quality attributes at early stages of development.

Table 7. Different releases of MobileMedia

Release	Description
r0	MobilePhoto core
r1	Error handling added
r2	Sort Photos by frequency and Edit Label concerns added
r3	Set Favourites photos added
r4	Added a concern to copy photo to an album
r5	Added a concern for sending photos by SMS
r6	Added the concern for playing music
r7	Added the concern for playing videos and capture media

Table 8. 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		

As discussed in Section 3.1, MobileMedia has evolved to 8 successive releases by adding different concerns to the product line. For instance, release 0 implements the original system with just the functionality of viewing photos and organizing them by albums. In Table 7 we show the different releases with the concerns added in each release (see [9] for more details). The reasons for choosing this application for our first analysis are several. (1) The MobileMedia application is a product line, where software instability is of upmost importance; instabilities affect negatively not only the Software Product Line (SPL) architecture, but also all the instantiated products. (2) The software architecture and the requirements had all-encompassing documentation; e.g., the description of all the usecases were made available as well as a complete specification of all the component interfaces. (3) The architectural components were independently defined and provided by the real developers, rather than ourselves. The architectural part could be used by a second study to analyze traceability of crosscutting concerns (observed using our metrics). (4) The developers had implemented an

aspect-oriented version of the system, which may be also used in a different analysis for comparing the metrics applied in different paradigms.

We have calculated the metrics presented in Section 3 for the requirements of each release. We have considered the different concerns of each release and the usecases implementing the system as the source and target domains, respectively. Table 8 shows the concerns used for the analysis and the releases in which these concerns were included. We do not show the usecases diagrams for each release due to space constraints, the whole analysis of the experiment may be found in [1].

4.2.1 Calculating the Metrics

Based on the two domains (concerns and usecases as source and target, respectively) we build the dependency matrix for each release showing the usecases contributing to the different concerns. Our metrics and those summarized in Table 6 are automatically calculated using as input the dependency matrix. Based on this dependency matrix, we derive the rest of matrices presented in Section 2.2 (Scattering, Tangling, Crosscutting Product, and Crosscutting Matrices). Due to space reasons, we just show the dependency matrix for the MobileMedia system in release 7, which includes all the concerns of the system (Table 9).

Table 9. Dependency matrix for the MobileMedia system in release 7

		Usecases																						
		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

Although our original dependency matrix is a binary matrix, in this case we have used a not-binary matrix in order to allow the calculation of metrics which utilize a granularity level different from usecase. That means that a cell represents the number of control flows or steps of the usecase addressing a particular concern. For instance, in Table 9 we can see how the View Album usecase has 3 and 1 control flows addressing the Album and Label concerns, respectively. In order to relate concerns and

usecases (i.e. fill in the dependency matrix), we have used a shadowing technique (see an example in Section 3.1) which was used at source code level in [9].

Using the dependency matrix for each release, we automatically calculate all metrics for these releases. In Table 10 we show the average of the metrics for all releases. In this table we have shown only the metrics calculated for concerns (source elements in the Crosscutting Pattern). In [1] we show the calculation for all the metrics presented in Table 6. We have performed a pairwise systematic comparison of the metrics and an in-depth discussion is presented at the website [1]. In next section we focus on the key results.

Table 10. Average of the metrics for all the releases

Releases	Average of all releases											
Authors	Ours					Sant'Anna			Ducasse			Eaddy
Metrics / Concerns	Nscattering	Degree of scattering	Crosscutpoints	NCrosscut	Degree of crosscut-ting	Concern Difussuion over Usecases	Concern Difussion over Flows	Usecase level Interlacing between Concerns	Size	Spread	Focus	Degree of Scatte-ring
Album	3,63	0,26	3,63	5,25	0,39	3,63	8	5,25	8	3,63	0,25	0,77
Photo	4,13	0,3	3,88	4,13	0,38	4,13	7,38	5,38	7,38	4,13	0,24	0,62
Label	5,38	0,34	5,38	6	0,46	5,38	7,88	6,13	7,88	5,38	0,25	0,82
Persistence	12,8	0,85	12,4	6,38	0,77	12,8	25,1	6,38	25,1	12,8	0,39	0,98
Error Handling	15,9	0,98	15,9	7	0,89	15,9	27,6	7	27,6	15,9	0,36	0,99
Sorting	4,33	0,25	4,33	7,33	0,43	4,33	5,33	7,33	5,33	4,33	0,34	0,78
Favourites	3	0,17	3	6	0,32	3	4	6	4	3	0,26	0,66
Copy	2,5	0,13	2,5	6,25	0,29	2,5	2,5	6,25	2,5	2,5	0,09	0,61
SMS	3,67	0,18	3,67	6,67	0,32	3,67	3,67	6,67	3,67	3,67	0,16	0,76
Music	1	0	0	0	0	1	3	0	3	1	1	0
Media	6,5	0,31	6,5	8,5	0,44	6,5	12,5	8,5	12,5	6,5	0,25	0,85
Video	1	0	0	0	0	1	1	0	1	1	1	0
Capture	1	0	0	0	0	1	1	2	1	1	0,33	0
Globals/Avg		0,27			0,34							

4.2.2 Discussion on Internal Validation

The main goal of the measures shown in previous section is to analyze the accuracy of the crosscutting metrics. However, by means of this validation we have also extracted important conclusions about the used metrics. First of all, we have observed through an analytical comparison (which was confirmed by an analysis of the MobileMedia data) that some of our proposed metrics are generic enough to embrace existing code-level metrics currently used in studies based on source-code analysis [8, 9, 12, 13]. Examples of these metrics are Sant`Anna's *Concern Diffusion over Components* or Eaddy's *Degree of Scattering*. A full pairwise comparison and discussion about the metrics is presented in [1].

In Table 10 we have shown the metrics which are more interesting for extracting conclusions on source elements (concerns). Fig. 3 shows three charts where *Degree of scattering* and *Degree of crosscutting* metrics are represented. Using these charts we observed that the metrics tend towards the same values for the same concerns.

One important conclusion that we have extracted from the analysis is the need for using the *Degree of crosscutting* metric (exclusive of our metrics suite). This metric is calculated using *Crosscutpoints* and *NCrosscut* metrics (see Section 3.4), and it is a special combination of scattering and tangling metrics.

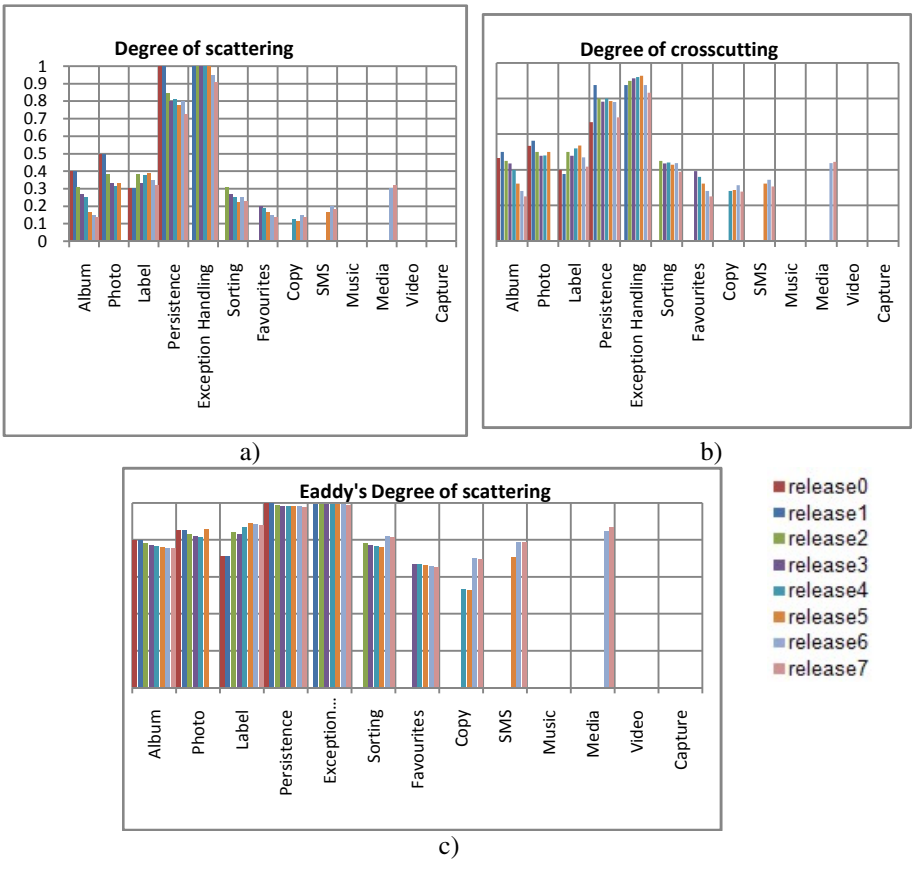
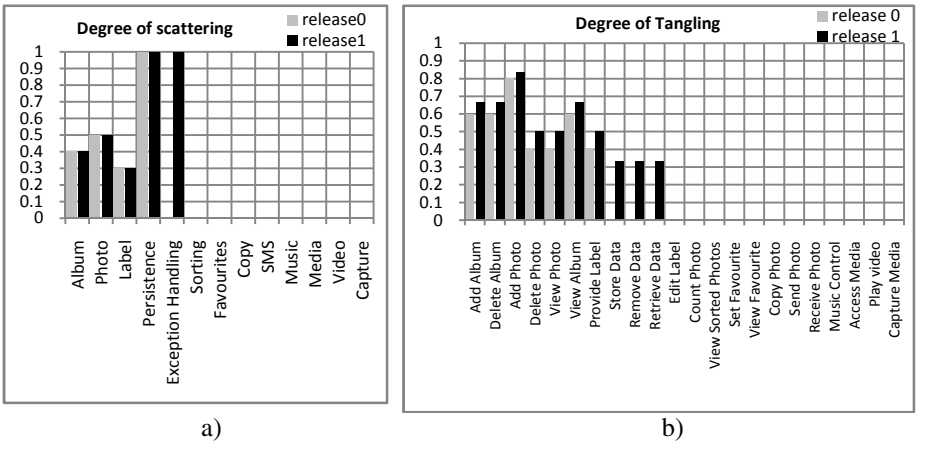


Fig. 3. Charts showing *Degree of scattering* (ours and Eaddy's) and *Degree of crosscutting*

Fig. 4 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. 5). This is due to the tangling of the usecases where the Album concern is addressed (see in Fig. 4b). 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 disclose 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 problem just observing the values for this metric. The same analysis could be done for Eaddy's metrics since they do not have a specific metric for crosscutting (see the Album concern in releases 0 and 1, in Fig. 3c).



a) b)

Fig. 4. *Degree of scattering* and *Degree of tangling* for releases 0 and 1

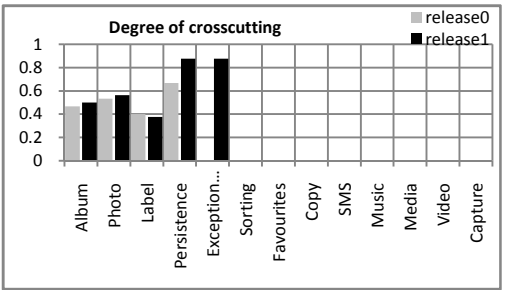


Fig. 5. *Degree of crosscutting* for releases 0 and 1

4.3 External Validation

To date, there is no empirical study that investigates whether scattering and crosscutting negatively affect to software stability. In this section, our analysis shows that the concerns with a higher degree of scattering and crosscutting are addressed by more unstable usecases than concerns with lower degree of scattering and crosscutting. Stability is highly related to change management so that the more unstable a system is, the more complicated the change management becomes (decreasing quality of the system) [4]. Then, we can infer that crosscutting has also a negative effect on software quality. In this analysis we mainly focus on changes in the functionality of the system. We do not focus on changes performed to correct bugs or in maintainability tasks (we do not rule out this kind of changes in future analyses).

4.3.1 Relating Crosscutting Metrics with Stability

In order to perform our empirical study, we have shown in Table 11 the usecases (rows) which change in the different releases (columns). A change in a usecase is due mainly to either the concerns which it addresses have evolved or it has been affected by the addition of a new concern to the system. In this table a 1 in a cell represents that in that release, the corresponding usecase has changed. 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 to all the usecases. An "a" in a cell represents that the usecase is added in that release. There are also some usecases which change their names in a release. These usecases are marked in the "Renaming" column, where the release which introduces the change in the name is shown (e.g. Add Photo usecase changes its name to Add Media in release 6). Finally, usecases with a number of changes higher than a threshold value (e.g. 2 in our analysis) are marked as unstable.

Table 11. Changes in usecases 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

Once the changes affecting each usecase are known, the number of unstable usecases which realise each concern is calculated. Table 12 shows the unstable usecases (those with two changes or more) in the columns and the concerns in the rows. A cell with 1 represents that the usecase addresses the corresponding concern. The last column of the table shows the total number of unstable usecases contributing to each concern.

We relate the number of unstable usecases for each concern with the degree of scattering and crosscutting for such concerns. In particular, Fig. 6 shows the linear

regression between the number of unstable usecases and the *Degree of scattering* and *Degree of crosscutting* metrics, respectively. We have used the least squares criteria to estimate the linear regression between the variables assessed so that the higher the degree of scattering or crosscutting for a concern, the more unstable usecases addressing such a concern. We can anticipate that usecases addressing scattered or crosscutting concerns are more prone to be unstable.

Table 12. Number of unstable usecases addressing each concern

		Usecases				
		Add Media	Delete Media	View Photo	View Album	Unstable usecases
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

We have also related Eaddy's *Degree of Scattering* metric with stability (Fig. 7). This figure complements the internal validation previously presented by showing consistency in the correlations of Fig. 6a) and Fig. 7 (they follow the same tendency).

4.3.2 Discussion on External Validation

In this section we present some conclusions extracted from the analysis performed in previous sections. As we can see in Fig. 6 and Fig. 7 correlations follow a linear tendency so that the higher the degree of scattering or crosscutting for a concern, the more unstable usecases addressing this concern. This analysis allows the developer to decide which parts of the system are more unstable just observing the degree of scattering or crosscutting. Also, since the analysis is performed in 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.

Fig. 6 and Fig. 7 also show the value for Pearson's *r* (a common measure of the linear dependence between two variables) [21]. The values of *r* shown in Fig. 6a) and Fig. 6b) are 0.844 and 0.879 respectively. These values indicate that *Degree of scattering* and *Degree of crosscutting* are highly correlated with the number of unstable components. Using the critical values table for *r* [21], we calculated the probability after N measurements (in our case 13) that the two variables are not correlated. For Fig. 6a) and Fig. 6b), the value obtained for this probability is 0.1%. Accordingly, the probability that these variables are correlated is 99.9%. For Fig. 7, we obtained that *r* is 0.788. Analogously, Eaddy's *Degree of Scattering* is also linearly correlated with

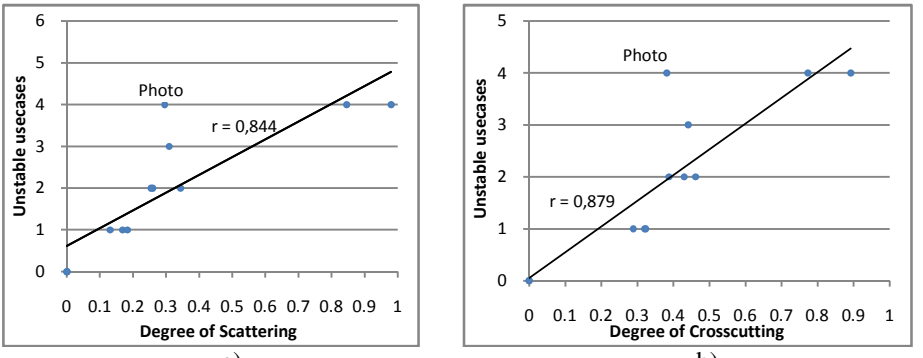


Fig. 6. Correlation between *Degree of scattering* and *Degree of crosscutting* and stability

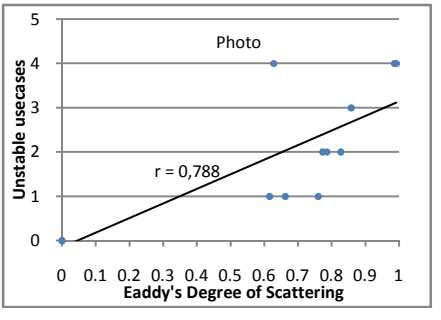


Fig. 7. Correlation between Eaddy's Degree of Scattering and stability

the number of unstable components. In particular, the probability that the variables assessed in Fig. 7 are not correlated is only 0.8%.

We observed that, in general, we obtained a better correlation for the *Degree of crosscutting* with stability than for *Degree of scattering* with stability. After analyzing the data, we observed that the correlations between *Degree of scattering* metrics (both ours and Eaddy's) and stability were much influenced by those concerns either without scattering or completely scattered. As an example, we can see in Fig. 6a) 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 situation is even more evident in Fig. 7 where the correlation coefficient obtained is lower than for the other correlations. The reason is the aforementioned commented: the difference between the values obtained for this metric in cases without scattering and the rest of cases. This metric obtained high values (greater than 0.5) for almost all the concerns assessed. However, when a concern does not present scattering the result of the metric is 0, highly influencing the correlation. Finally, we concluded that *Degree of crosscutting* presents a better correlation with stability since this metric somehow takes into account not only scattering but also tangling. This conclusion supports the need for having a specific metric for assessing crosscutting.

We have also annotated in all the correlations a point called *Photo*. These points are the most digressed from the linear regression in all the figures. We observed that although this concern (Photo) presents values for the scattering and crosscutting metrics not very high, the number of unstable usecases was high. After analyzing this situation, we observed 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 *crosscutting* for Photo drastically decrease in releases 6 and 7. It highly influences to the average of the metrics and this is the reason why although having non-relatively high values for the metrics; the number of unstable usecases remains high.

5 Related Works

In [19], Sant'Anna et al. introduce different metrics (summarized in Table 6), namely *Concern Diffusion over Components (CDC), 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. However, these metrics are mainly defined to assess modularity using specific deployment artefacts so that they are focused on specific abstraction levels (design or programming). In [18], the same authors adapted the metrics to the architectural level and introduced new metrics. However, the metrics still keep tied to specific deployment artefacts and they are not generic enough to be used at any abstraction level. In [9], the metrics are used for analyzing stability in product lines. However, the work is very tied to the programming level, relegating the benefits of the metrics to the latest phases of development. In [7], Ducasse et al. introduce four concern measures: *Size*, *Touch, Spread and Focus* (see Table 6). Again, these metrics are tied to the implementation.

Wong et al. introduce in [22] three concern metrics called *Disparity, Concentration* and *Dedication* (Table 6). Eaddy et al., use an adaptation of *Concentration* and *Dedication* metrics for defining two new concern metrics [8]: *Degree of Scattering (DOS)* and *Degree of Tangling (DOT)*. Whilst *DOS* is defined as the variance of the *Concentration* of a concern over all program elements with respect to the worst case, *DOT* is defined as the variance of the *Dedication* of a component for all the concern with respect to the worst case. Both works are focused on assessing modularity at programming level as well.

In [16] Lopez-Herrejon and Apel define two concern metrics: *Number of Features (NOF)* and *Feature Crosscutting Degree (FCD)*, measuring number of features and number of classes or components crosscut by a particular concern respectively. Ceccato and Tonella also introduce a metric called *Crosscutting Degree of an Aspect (CDA)* [5] which counts the number of modules affected by an aspect. These metrics are defined to assess attributes of an aspect-oriented implementation. They could not be used to anticipate decisions in not aspect-oriented systems.

In [20], the authors use a tool to analyze change impact in Java applications. This tool allows the classification of changes so that they detect the more failure inducing changes. However, like most of the aforementioned approaches, this work is focused on programming level when the system is already designed.

6 Conclusions and Future Work

In this paper, we proposed a concern-oriented metrics suite to complement traditional software metrics. The metrics proposed are based on the crosscutting pattern presented in our previous work which establishes a dependency between two generic domains, source and target, based on traceability relations. This metric suite allows the developer to perform a modularity analysis, identifying the crosscutting concerns in a system but also quantifying the degree of crosscutting of each concern. The metrics are generic and they are not tied to a specific deployment artifact. Then, they may be used at different abstraction levels, allowing developers to assess modularity and infer quality properties at early stages of development. Even, with the transition to model-driven software engineering gaining momentum, the assessment of abstract models (usecases or components models) becomes more important.

Through the internal validation, we observed not only that our metrics are consistent with other metrics but also that they complement other metrics since they are not defined in terms of any deployment artefact. Moreover, we showed the need for introducing a specific metric for crosscutting. The external validation was focused on demonstrating the utility of the metrics for other software quality attributes. In particular, we show how the *Degree of scattering* and *Degree of crosscutting* metrics present a linear correlation with stability and that the *Degree of crosscutting* metric is better correlated with stability than *Degree of scattering*.

As future work, we plan to perform several empirical studies. In a first study we expect to compare the results obtained by our metrics at requirements level with those obtained at different abstraction levels (e.g., architectural level, design or implementation). Also, the application of the metrics at source-code level would allow us to compare our results with the obtained by other studies were authors analyze instability at this level (e.g., [9, 13]). By this analysis we could test different hypotheses, such as, whether similar properties of crosscutting concerns are found to be indicators of instabilities 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. We also plan to apply the metrics in several case studies (projects) demonstrating that the results obtained are not just coincidental.

Acknowledgements

This work has been supported in part by the European Commission grant IST-2-004349: European Network of Excellence on AOSD (AOSD-Europe) and by MEC under contract: TIN2008-02985. Eduardo is supported by CAPES, Brazil.

References

1. Analysis of modularity in the MobileMedia system (2008), http://quercusseg.unex.es/chemacm/research/analysisofcrosscutting
2. Baniassad, E., Clements, P., Araújo, J., Moreira, A., Rashid, A., Tekinerdogan, B.: Discovering Early Aspects. IEEE Software 23(1), 61–70 (2006)
3. van den Berg, K., Conejero, J., Hernández, J.: Analysis of Crosscutting in Early Software Development Phases based on Traceability. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 73–104. Springer, Heidelberg (2007)
4. van den Berg, K.: Change Impact Analysis of Crosscutting in Software Architectural Design. In: Workshop on Architecture-Centric Evolution at 20th ECOOP, Nantes (2006)
5. Ceccato, M., Tonella, P.: Measuring the Effects of Software Aspectization. In: Proceedings of the 1st Workshop on Aspect Reverse Engineering, Delft University of Technology, the Netherlands (2004)
6. Conejero, J., Hernandez, J., Jurado, E., van den Berg, K.: Crosscutting, what is and what is not? A Formal definition based on a Crosscutting Pattern. Technical Report TR28_07, University of Extremadura (2007)
7. Ducasse, S., Girba, T., Kuhn, A.: Distribution Map. In: Proc. of the Int'l Conference on Software Maintenance (ICSM), Philadelphia, USA (2006)
8. Eaddy, M., Zimmermann, T., Sherwood, K., Garg, V., Murphy, G., Nagappan, N., Aho, A.: Do Crosscutting Concerns Cause Defects? IEEE Transactions on Software Engineering 34(4), 497–515 (2008)
9. Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F., Dantas, F.: 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)
10. Figueiredo, E., Sant'Anna, C., Garcia, A., Bartolomei, T., Cazzola, W., Marchetto, A.: On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework. In: Proceedings of CSMR 2008, pp. 183–192 (2008)
11. Garcia, A., Lucena, C.: Taming Heterogeneous Agent Architectures. Commun. ACM 51(5), 75–81 (2008)
12. Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., Staa, A.: Modularizing design patterns with aspects: A quantitative study. In: Rashid, A., Aksit, M. (eds.) Transactions on Aspect-Oriented Software Development I. LNCS, vol. 3880, pp. 36–74. Springer, Heidelberg (2006)
13. Greenwood, P., Bartolomei, T., Figueiredo, E., Dosea, M., Garcia, A., Cacho, N., Sant'Anna, C., Soares, S., Borba, P., Kulesza, U., Rashid, A.: On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. In: Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 176–200. Springer, Heidelberg (2007)
14. Kelly, D.: A Study of Design Characteristics in Evolving Software Using Stability as a Criterion. IEEE Trans. Software Eng. 32(5), 315–329 (2006)
15. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
16. Lopez-Herrejon, R., Apel, S.: Measuring and Characterizing Crosscutting in Aspect-Based Programs: Basic Metrics and Case Studies. In: Proc. of the Int'l Conference on Fundamental Approaches to Software Engineering (2007)

17. Masuhara, H., Kiczales, G.: Modeling Crosscutting in Aspect-Oriented Mechanisms. In: Cardelli, L. (ed.) ECOOP 2003. LNCS, vol. 2743, Springer, Heidelberg (2003)
18. Sant'Anna, C., Figueiredo, E., Garcia, A., Lucena, C.J.P.: On the modularity of software architectures: A concern-driven measurement framework. In: Oquendo, F. (ed.) ECSA 2007. LNCS, vol. 4758, pp. 207–224. Springer, Heidelberg (2007)
19. Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., von Staa, A.: On the Reuse and Maintenance of Aspect-Oriented Software: an Assessment Framework. In: Proc. of the Brazilian Symposium on Software Engineering (SBES), Manaus, Brazil (2003)
20. Stoerzer, M., Ryder, B.G., Ren, X., Tip, F.: Finding Failure-Inducing Changes in Java Programs using Change Classification. In: Proc. of 14th International Symposium on Foundations of Software Engineering, Portland, USA, pp. 57–68. ACM, New York (2006)
21. Taylor, J.: An Introduction to Error Analysis. The Study of Uncertainties in Physical Measurements, 2nd edn. University Science Books (1997); ISBN: 0-935702-75-X
22. Wong, W., Gokhale, S., Horgan, J.: Quantifying the Closeness between Program Components and Features. Journal of Systems and Software (2000)