The AOSD Research Community in Brazil and its Crosscutting Impact

Christina Chavez‡, Uirá Kulesza¶, Sérgio Soares†, Paulo Borba‡, Carlos Lucena‡, Paulo Masiero*, Claudio Sant’Anna‡, Eduardo Piveta‡, Fabiano Ferrari**, Fernando Castor‡, Roberta Coelho†, Lyrene Silva†, Vander Alves‡, Nabor Mendonça††, Eduardo Figueiredo, Valter Camargo**, Carla Silva‡, Paulo Pires‡, Thais Batista†, Nélio Cacho‡, Arndt von Staa‡, Julio Leite†, Fabio Silveira††, Otávio Lemos‡, Rosangela Penteado**, Flávia Delicato‡, Rosana Braga†, Marco Túlio Valente†, Ricardo Ramos†, Rodrigo Bonifácio‡, Fernanda Alencar‡, Jaelson Castro‡, *UFBA, †UFPE, §PUC-Rio, ¶USP, ⌂UFSCAR, ⌃UnB, UNIFOR, UNIFESP, UNIVASF

Abstract—In this paper, we present the birth, growth, and maturation of Aspect-Oriented Software Development (AOSD) research over the last years, with emphasis on the Brazilian AOSD community and its research contributions. These research contributions are illustrated from different perspectives: (i) an overview of the research work developed by our community in several prominent software engineering areas; (ii) a historical chronology of the community; and (iii) the growth, impact and quality of research outcomes.

I. INTRODUCTION

Aspect-Oriented Software Development (AOSD) [4] promotes advanced separation of concerns and software modularity throughout the software development lifecycle, by investigating, developing, and evaluating new modularization paradigms and techniques for extensibility, flexibility, and adaptability.

Separation of concerns is a fundamental principle that addresses the limitations of human cognition for dealing with complexity. It advocates that to master complexity, one has to deal with one important issue (or concern) at a time [1]. In software engineering (SE), this principle is usually related to system decomposition and modularization: complex software systems should be decomposed into smaller, clearly separated modular units, each dealing with a single concern. The expected benefits are improved comprehensibility and increased potential for evolution and reuse.

The abstraction and composition mechanisms supported by successive generations of programming languages have evolved to foster the expression of solutions to real world problems in a more natural way as well as to advance one’s ability to achieve separation of concerns at the source code level. Object-Oriented Programming (OOP) has been the dominant programming technology for several years and its benefits are broadly recognized. However, OOP has some limitations for dealing with concerns that address requirements involving global constraints and systemic properties, such as synchronization, persistence, error handling, logging mechanisms, among many others. These concerns have been called crosscutting concerns (CC) since they naturally crosscut the boundaries of modular units that implement other concerns. Without proper means for separation and modularization, CC tend to be scattered over a number of modular units and tangled up with other concerns. The natural consequences are lower cohesion and stronger coupling between modular units, reduced comprehensibility, evolvability, and reusability of code artifacts.

Aspect-Oriented Programming (AOP) [3] is an evolutionary technology that supports a new flavor of separation of concerns at the source code level. AOP uses aspects as a new modular unit for separating CC, also providing a new mechanism for composing (or weaving) aspects with other modular units at well-defined join points. Aspect-Oriented Software Development (AOSD) has the goal of promoting separation of CC throughout the software development lifecycle [4].

Since the seminal work of Kiczales et al. [3] was presented at ECOOP 1997, the research work on AOSD has evolved and matured to be applied along the different software development stages and activities with the promise of improving the modularity of crosscutting requirements, concerns, and features. The AOSD Brazilian (AOSD-BR) research community also emerged and matured during this period, and has contributed to different research topics. The involvement of this community with the Brazilian Symposium on Software Engineering (SBES) since 1999 has been very important to promote a long established interaction between the AOSD-BR research groups and the development of high-quality research work, thus allowing the consolidation of this community along the last years.

In this paper, we present the birth, growth, and maturation of AOSD research over the last years, with emphasis on the Brazilian AOSD community (AOSD-BR) and its research contributions, illustrated from different perspectives. Section II presents an overview of the research work developed by AOSD-BR in several prominent SE areas. Section III presents a historical chronology of the community from its official birth – the First Brazilian Workshop on AOSD (WASP) in 2004 – until the organization, in Brazil, of the 10th International

---

1These limitations are not specific to OOP, but rather part of a general problem that has been coined “The Tyranny of Dominant Decomposition” [2].
Conference on AOSD in 2011, and a discussion on the growth, impact, and quality of the AOSD-BR research outcomes in terms of several indicators, such as the number of publications in journals and conferences over the last decade, the number of papers published in international top SE conferences, and the number of citations to papers from our community. Section IV raises some questions for discussion by the Brazilian AOSD community.

II. OVERVIEW OF RESEARCH CONTRIBUTIONS

In this section, we provide an overview of the main contributions of the AOSD-BR community for different AOSD research topics and challenges.

A. Exception Handling

Exception handling (EH) [5] is a promising candidate for the use of AOP techniques since, in most programming languages, it is an inherently scattered and tangled concern [3]. The first study to investigate the use of AO constructs to modularize EH was performed by Lippert and Lopes [6]. In this work they observed a considerable reduction in the amount of EH code in the AO version of a reusable object-oriented framework. Since then, a considerable body of work has studied the effects of aspects on EH, from a number of different perspectives. AOSD-BR researchers have conducted the majority of this research. We can organize the work on exceptions and aspects in three categories: (i) investigation on the effects of using aspects to modularize EH code; (ii) new EH mechanisms leveraging AOP constructs; (iii) tools to support the understanding and verification of EH code in AO systems.

Castor Filho et al. [7] conducted a study similar to the one of Lippert and Lopes but targeting four different, non-trivial applications. This work revealed that the aspectization and reuse of exception handlers is not as straightforward as advocated beforehand by Lippert and Lopes. Instead, it depends on a set of factors, such as, the type of exceptions being handled, what the handler does and the amount of contextual information needed. Later on, Castor Filho et al. [8] elaborated a catalog of best and worst practices related to the aspectization of EH. Taveira et al. [9] continued the investigation and found out that a considerable amount of error handling code could be reused inside a single application by using aspects. Nonetheless, the programming overhead associated to it, somewhat, overshadowed that reuse.

Coelho et al. [10] performed an empirical study which took into account the error-proneness of AspectJ constructs for handling exceptions. The most important finding of the study was that EH aspects, even when used with great care, can introduce a number of bugs that materialize as uncaught exceptions. This led to the creation of a catalog of bug patterns and an approach for preventing and automatically detecting such bug patterns [11].

In the second category, Cacho et al. [12] devised a new EH model that uses the join point and advice mechanisms of AspectJ to generalize the concepts of EH context and exception handler. Later, Cacho et al. [13] performed an exploratory study to evaluate this new model, and observed that it was useful to foster the development of readable and reliable software systems.

Last but not least, Brazilian researchers have recently devised a number of testing and static analysis approaches targeting the EH code of AO systems [14], [15]. Bernardo et al. [14] proposed an extension of the JUnit framework to check whether the paths that exceptions take in an OO or AO system are the intended ones. This tool relies on aspects to monitor the exception paths of a system. Coelho et al. [15] presented a static-analysis-based approach which allows AO developers to describe the EH behavior of a system and automatically check it. Both tools target Java and AspectJ based systems.

B. Refactoring

The AOSD-BR research community has been working in several research areas related to Refactoring [16]. For instance, the community has results on the identification of refactoring opportunities, including the definition of catalogs of bad smells [17], their detection [18], and guidelines to reduce their occurrence in AO software [19]. Additional research focused on the representation of refactoring opportunities [20] and on refactoring processes [21]. Moreover, recent studies have pointed out that not all kinds of CC are harmful to the system quality attributes, such as robustness [22] and stability [23]. Therefore, only harmful CC should be factored out to aspects. Brazilian researchers on AOSD have documented [24] and proposed means to modularize [25] several patterns of CC recurrently observed in software systems. The patterns of harmful CC can be detected by visualization environments [26], metrics and heuristic strategies [27], [28].

Another research area focuses on assessing the effects of refactoring on quality. In this context, AOSD-BR researchers have been developing tools and concepts to improve the quality of AO software. One of these tools is ConcernMetrics [29], an Eclipse plugin that computes metrics commonly employed to evaluate the benefits of AOP, without requiring physical extraction of CC to aspects. This tool was used to evaluate the benefits of using aspects in three small-to-medium Java systems.

The AOSD-BR community has also played a pioneering role in research focusing on guaranteeing that AO refactoring preserves behavior. The first work on this area uses AO programming laws for deriving refactorings for AspectJ [30], aiming to verify if the transformations they define preserve behavior. These laws have been used to derive a catalog of AspectJ refactorings, which can be employed to restructure applications and to extract variations in software product lines. In both cases, the aspects were used as a modularization mechanism, either for features [31] or for common CC, such as distribution and persistence. Most refactorings on this catalog were implemented in an industrial strength tool [32] and used in the mobile game domain [33].

Refactoring is also used to provide mechanisms for aspect-mining, in which code fragments related to a CC are located
and extracted to aspects. However, the base program often has to be modified to fit these aspects properly. In this context, the AOSD-BR community has proposed a catalog of OO transformations [34] to associate crosscutting code with points of the base program that can be captured using the pointcut model of current AO languages. This catalog was evaluated on a case study with medium-sized Java-based systems that have been aspectized using AspectJ.

Current and future work on this area focus on refactoring to introduce aspects in large-scale software applications and product lines, including refactorings for extracting features to aspects, for improving the internals of AO constructs, and for moving features from aspects to objects.

C. Testing

The understanding and subsequent testing of AO programs is one of the main ways to make the approach less costly and more feasible in practice [35]. With this in mind, the AOSD-BR research community has contributed to establish testing approaches that address particularities of AO software. The main achievements are following summarized.

1) Structural-based testing: Zhao [36] published the seminal piece of work related to structural testing of AO programs. Since then, the AOSD-BR contributions have been very relevant, resulting in a series of papers published in scientific events [37], [38] and in high quality journals [39], [40], [41]. The contribution addresses structural-based testing at the unit level [37], [39] and the integration level [38], [40], [41]. Control flow and data flow models have been formally defined, from which a range of test selection criteria has been devised. The control flow-based criteria extended the concepts of the all-nodes and all-edges criteria to incorporate AO-specific properties like join points and pointcuts. Similarly, the widely studied data flow-based criteria, such as all-defs and all-uses, were adapted to take into consideration data that flows through aspectual elements like pieces of advice and introductions. The applications of all these criteria is supported by a tool named JaBUTi/AJ [39], [40], [41], which has been continuously evolved through the last years.

2) Fault-based testing: Fault-based testing [42] requires a well-characterized set of faults types – generally organized as a taxonomy – to be applied effectively. In this context, Alexander et al. [43] had the first initiative to define a fault taxonomy for AO software. Some years later, Brazilian researchers performed a systematic survey of candidate fault taxonomies and devised a comprehensive taxonomy for AO programs [44]. Their subsequent studies investigated the fault-proneness of evolving AO programs and revealed that the taxonomy has the ability to classify the observed variety of AOP-specific faults. These studies addressed, respectively, coarse-grained [22] and fine-grained fault classifications [45]. The proposed taxonomy guided the definition of a mutation-based testing approach for AspectJ-like programs [44]. The approach encompasses a set of mutation operators that model several instances of fault types and is supported by a tool named Proteum/AJ [46]. Note that such set of operators has inspired other research groups to propose AO-specific testing approaches and automated support [47]. The feasibility of applying the mutation operators supported by Proteum/AJ has been recently reported [48].

3) Functional-based testing: Functional-based testing has been addressed by Silveira [51], with emphasis on the importance of developing a testing model for AO programs, which can reveal a potential fault caused by aspects composition project both in static and dynamic weaving processes. METEORA [52] is a new state-based testing method for AO software, which includes a model to represent the dynamic behavior of aspects interactions, an extended strategy to derive testing sequences, and a testing tool prototype to support the method.

Experimentation and other types of evaluation comprise an essential part of research. With this in mind, a major limitation to be addressed by the community is the robust evaluation of the varied testing approaches proposed in the last decade. Another prominent research branch comprises hybrid approaches that combine different testing techniques (e.g. structural and mutation testing, or static and dynamic evaluation). Preliminary efforts have been reported [49], [50] and shall be further revisited.

D. Metrics

As a new software development technique emerges, it is essential that empirical studies are carried out to investigate whether and in which cases the advocated benefits are real. Software metrics provide modularity indicators of software design. The AOSD-BR community has been developing significant work on quantitative assessment of AO software.

Sant’Anna et al. proposed one of the first works on AO metrics: a suite of metrics for quantifying modularity-related attributes, published in a SBES paper [53]. The suite included coupling, cohesion and size metrics adapted from existing OO metrics to deal with AO abstractions and mechanisms. The suite also encompassed innovative concern-driven metrics, aimed at quantifying different facets of separation of concerns. Concern-driven metrics promote the notion of concern as a measurement abstraction. This kind of metric relies on the identification and documentation of the pieces of source code that implement each concern of the system. A first empirical study relied on the suite of metrics to compare Java and AspectJ implementations of a multi-agent system [53], [54]. These metrics were later formalized by means of a conceptual framework [55].

Several other empirical studies used the suite of AO metrics proposed in [53] to assess AO systems of different domains and nature. Empirical studies [56], [57], [58] compared the modularity of Java and AspectJ solutions of the 23 design patterns from the Gang of Four. Another study systematically investigated how AOP scaled up to deal with modularization of design patterns in the presence of pattern interactions [59]. The authors made both qualitative and quantitative assessments of 62 pair-wise compositions of patterns taken from 3 medium-sized systems implemented in Java and AspectJ programming languages. Cacho et al. [60] assessed the use of
AOP for improving the modularity of a reflective middleware. Kulesza et al. [61] presented an empirical study in which they quantified the effects of AOP in the maintainability of a web-based information system. Figueiredo et al. [23] undertook an empirical study for evaluating whether AOP promotes better modularity and changeability of product lines than conventional variability mechanisms, such as conditional compilation. Silva et al. [62] performed an empirical study to assess whether AO solutions for agent-oriented design patterns improve the separation of pattern-related concerns.

The aforementioned studies showed that concern-driven metrics could be a very useful mechanism for evaluating software modularity. Then, some works focused on further investigating this kind of measurement. Sant’Anna et al. [63], [64] extended the metrics suite defined in [53] with new concern-driven metrics. Besides concern scattering and tangling, these new metrics also quantify concern interaction, concern-based cohesion, and concern-sensitive coupling. This work also adapted concern-driven metrics to be applied on architectural design and the architectural metrics were used in some empirical studies [63], [65], [64], [66]. In addition, Figueiredo et al. [28] defined heuristic rules based on concern-driven metrics aiming at detecting code smells.

There are several opportunities for future work on concern-driven measurement. First, there is still a need for empirically validating concern-driven metrics in terms of their correlation with external quality attributes, such as change-proneness [67]. The use of visualization techniques to support concern-driven measurement has already been exploited [26], but there is still much to be done in this area.

E. Early Aspects

The term “Early Aspects” (EA) is used to refer to crosscutting properties at the requirements and software architecture levels. Preliminary work on AO Requirements Engineering (AORE) [68], [69], [70] pointed out that some existing techniques failed to explicitly handle the crosscutting nature of some requirements. These crosscutting requirements produced scattered and tangled representations that were harder to understand, maintain and reuse. In this context, the Requirements Engineering community addressed CC and their modularization in earlier phases of the software development cycle.

Sousa et al.’s work was the first one to address AORE in Brazil [71]. The authors used the NFR-Framework [72] along with use cases to explicitly represent crosscutting requirements separately from non-crosscutting ones and to compose them in a non-invasive way. Other researchers have addressed the modeling [73], [74] and evaluation of AO requirements [75], the mapping of CC between different modeling languages, and the elicitation of crosscutting requirements [76], [77].

In the context of requirements modeling, an extension of the i* (iStar) modeling language [78], called Aspectual i*, was proposed to support separation of CC [74]. i* models capture social and intentional characteristics, while Aspectual i* defines rules to systematically identify, modularize and compose CC, reducing the complexity and making maintenance and evolution of i* models easier [74], [79]. Aspectual i* has also been used as the basis for the identification of aspectual use cases, as described in [80].

In the modeling context, a meta-model for AO requirements languages [73] was defined. Two modeling languages have been defined upon this meta-model: (i) AOPML [81] – an extension of the BPM notation to describe processes, improve modularity and make processes easier to understand and reuse [82]; and (ii) AOV-graph [83] – an extension of V-graph (a goal-oriented language used to model early aspects [84]) to represent the relationships among requirements. The AOV-graph has been used in empirical studies with the Health Watcher and Mobile Media benchmarks [85], [86], [87].

More recently, AOV-graph has been extended to PL-AOVgraph, in a strategy to automatically map feature models in the context of Software Product Lines (SPL) – see also Section II-G – into PL-AOVgraph and vice-versa [88]. Aspectual i* has also been used in the SPL context to capture common and variable features and thus facilitate the selection among product configurations [89]. Finally, use case scenarios [90] and business processes [91] were defined to modularize features in SPL specifications. Case studies and controlled experiments were conducted showing that in software evolution scenarios, the gains in modularization depend on several factors such as the team experience and the nature of the change requests.

There are many challenges ahead of AORE research, especially in the context of modeling tools, evaluation mechanisms and controlled experiments for assessing the benefits of the separation of crosscutting requirements, from earlier to later stages in the development lifecycle. Another issue to be addressed is how to mainstream these approaches into industry.

In the context of software architecture, research on AO architecture aims to improve and broaden the understanding of the identification, representation and management of architectural concerns, specially crosscutting architectural concerns. The work from Batista and colleagues [92] pioneered the discussions on some fundamental issues about the integration of AOSD and architecture description languages (ADLs). They proposed a composition model based on existing architectural abstractions – connectors and configuration – to support the definition of crosscutting composition facilities and quantification mechanisms. The AspectualACME [93], [94] AO ADL was developed following this approach. AspectualACME supported improved composability by means of aspectual connectors. In contrast to the heavyweight solutions provided by some AO-ADLs, AspectualACME is a lightweight AO-ADL that extends traditional architectural connection abstractions to support the definition of composition facilities.

Chavez et al. [95] propose style-based composition, a new flavor of aspect composition at the architectural level based on architectural styles that addresses the pointcut fragility problem. The authors proposed style-based join point

2Pointcut definitions are fragile because when the system evolves, pointcuts may unintentionally capture or miss a given join point, as a result of apparently harmless modifications of base elements.
models and provided a pointcut language that supports the selection of join points based on style-constrained architectural models. Stability and reusability assessments of the proposed style-based composition model were carried out through three case studies involving different styles [95]. The interplay of style-based pointcuts and some style composition techniques was also discussed [97] using four different architectural styles applied to an evolving multi-agent architecture.

Current and future directions include the integration of ADLs, AOSD and SPL. The idea is to support the modular representation of architectural commonalities, variabilities, and their composition, and also providing means to express the specific product configuration. Another promising venue is to provide mechanisms to improve the traceability to other activities and related models (e.g., analysis and implementation).

F. Modeling and Model-Based Techniques

Aspect-Oriented Modeling (AOM) is a critical part of AOSD that focuses on notations and techniques for specifying, representing, visualizing and communicating AO solutions. Approaches range from lightweight profiles for AspectJ programs [98] to heavyweight and generic UML extensions [99] for most kinds of AO programs. The AOSD-BR research community has also developed solutions for AOM, from modeling languages, for instance, the aSideML language [100], [101] to profiles such as UML-AOF [102]. The aSideML language[100] is a modeling language for specifying and communicating AO designs. It provides notation, semantics and rules with the main purpose of addressing the conceptual modeling and detailed design of a system in terms of aspects and crosscutting within the UML Object Model – for structural and behavioral models. UML-AOF is a profile that makes the design characteristics of AO frameworks (AOFs) more evident in models than standard profiles for AOP [102]. UML-AOF gathers several stereotypes and tagged values which represent design and architectural details commonly found in AOFs, such as idioms, patterns and extension mechanisms. Furthermore, since most profiles lack important concepts or have deficiencies when used for code generation, a process to detect inconsistencies between what is provided by the profile and what is required by the paradigm [103] has been proposed.

Historically, AOSD has focused on modeling and managing CC (horizontal separation) whereas Model Driven Development (MDD) [104] has focused on the explicit separation of platform independent from platform specific concerns (vertical separation). As such, AOSD and MDD are complementary and can benefit from each other to tackle the challenges of developing large and complex software systems.

Some work combines AOSD with MDD, integrating CC in models by using model-driven transformations [101], [105], [106], [107], [108]. The AOSD-BR community has exploited the synergy between AOSD and MDD by developing an infrastructure named CrossMDA [109], which encompasses a transformation process, a set of services and associated support tools. CrossMDA is based on the Model-Driven Architecture (MDA) [110] and aims at: (i) raising the abstraction level of aspect modeling through the use of platform independent models (PIM) representing CC independent on the business models; (ii) promoting the reuse of CC modeled as PIM elements; (iii) automating the process of mapping the relationship of CC and business models through automatic transformation.

The AOSD-BR community has developed different solutions to tackle the pointcut fragility problem [96] using model-based techniques. One of the proposals is an extension of the CrossMDA framework [109], named CrossMDA2 [111], built as a new instantiation of the model-driven pointcuts approach [96]. Another approach [112] also relies on the use of model-based pointcuts [96] but has the distinctive feature of defining model-based pointcuts at the architectural level, instead of focusing at the design or implementation levels as other existing proposals. An MDD process drives the definition of conceptual and aspect models, and their instantiation and composition to generate the architectural base model. The solution is implemented using an AO-ADL [113].

The research in Model-Based Techniques in AOSD can be further explored in different directions. One possible direction is to investigate the types of join points that can be suitably captured by architecture-oriented approaches and those that cannot be captured (or even do not make sense to be). Another one is to investigate what happens with the portions of the system that do not adhere to identifiable architectural patterns.

Finally, the area still lacks on experiments that evaluate practical aspects, such as the overhead introduced in the software development process by the proposed approaches.

G. Product-Lines and Frameworks

A Software Product Line (SPL) is a set of software intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission, and that are developed from a common set of core assets in a prescribed way [114]. Since the implementation of features tends to have crosscutting and tangled implementation [115], AOSD has been explored as a possible implementation strategy for SPLs in order to tackle these issues and ensure modular feature implementation.

The synergy between AOP and SPL research has been reported in the SPL community by exploring variability management to address portability issues in industrial-strength mobile game development [116], [31]. Such works propose the use of aspects to modularize the implementation of device-specific features, by applying a set of refactorings to bootstrap a product line and to evolve it. Furthermore, an SPL-based extension of XPIs [117] named Extension Join Points (EJPs) [118], [119] has been proposed to enforce design rules between object-oriented core and aspect-based device-specific assets. Empirical work has assessed benefits and drawbacks of AOP in evolving SPLs [23].

In the context of AOSD, many researchers have investigated how aspects impact both framework development and instantiation [120], [121], [122], [123] and how to turn the reuse of CC into a more systematic and controlled task. A Crosscutting Framework (CF) [123], [124] is an AO framework that
encapsulates a single CC, e.g., persistence or business rules. Patterns [125], idioms [126] and UML profiles [127] support the use of CFs.

The interaction between AOP and SPL is also investigated in terms of development processes and tools. The work of Pacios and others [128] proposes guidelines to use AOP to implement the functional features of a SPL using an incremental strategy. It was further evolved to an approach named AIPLE-IS [129], in which the main idea is to incrementally build a SPL for information systems using AOP. The SPL core is developed first, followed by the addition of optional features through aspects. Application generators have been proposed to ease SPL instantiation. Captor-AO is a Configurable Application Generator that implements AOP concepts and is used to support the development of domain-specific applications [130]. Captor-AO can also be used in an approach in which aspects are used to implement SPL crosscutting features to improve their reuse not only within the SPL domain, but also across different domains [131]. The idea is to isolate these crosscutting features into “crosscutting domains”, so that features scattered across different SPLs are easily located and reused.

Prospective development on the interaction of AOSD and SPL Engineering includes providing sound and evidence-based guidelines and tools on how to combine different implementation strategies that provide feature modularity and stability of SPL assets as well as usability and low usage cost. Another promising research thread is exploring the use of formal methods techniques in order to ensure in a scalable way the checking of properties such as type safety and non-functional requirements of SPL artifacts at different levels of abstraction.

III. EVOLUTION AND RESEARCH IMPACT

A. AOSD-BR Timeline and Growth

The creation of the Brazilian Workshop on AOSD (WASP), in 2004, is probably the starting milestone of the AOSD Brazilian community. That was definitely the first effort to bring together the AOSD researchers in a single event. WASP was organized as a one-day workshop at the Brazilian Symposium on Software Engineering (SBES). After the first edition in Brasilia - DF, two more WASP workshops were held at SBES 2005 (Uberlândia - MG) and SBES 2006 (Florianópolis - SC). At this point, the community was better organized and there were well-established international collaborations.

In 2006, the Latin American Network for Research in AOSD (LatinAOSD) emerged, involving several institutions from Brazil, Chile, Argentina and Colombia. The LatinAOSD network pushed WASP to a broader scope, and in 2007, WASP became LA-WASP, the Latin-American Workshop on AOSD, organized as two-day workshops at SBES.

In 2011, the Brazilian AOSD community had another important milestone, the 10th International Conference on AOSD was held in Brazil (Porto de Galinhas - PE). The conference had 138 attendees, where 65 (47%) were Brazilian researchers. This recognizes the role played by the AOSD Brazilian community in the international context. Table I presents a timeline that summarizes the Brazilian community milestones also showing some crucial international AOSD milestones.

Table II depicts raw/absolute numbers related to AOSD research in Brazil, including publications, theses, and international cooperation projects. The selected papers: (1) had at least one author working in a Brazilian institution when the paper was written; (2) brought contributions to AOSD or related topics; and (3) were indexed by Google Scholar. The growth of the AOSD community can be noted by the number of international research projects developed in cooperation among Brazilian and foreign groups, as well as an increase in the number of research papers resulting from such cooperative research. This is more evident when looking at the number of conference papers (Table II) – more than 50% were accepted by international conferences.

Figures 1 and 2 depict publications and theses growing from the AOSD-BR community over the years. It is interesting to notice: (i) the increase in the number of publications and theses since the formation of AOSD-BR community in 2004; and (ii) the high number of international publications that, since 2006, has been greater than the number of publications at Brazilian
venues, showing once more the internationalization process of this community. Figure 1 also shows the number of SBES papers from the AOSD-BR community. The growth of the community and its research can be clearly seen by the increase of the AOSD related papers at SBES over the years. In fact, most of the national AOSD-BR papers (76) were published on SBES (49), as shown in Table II.

B. Assessment of AOSD-BR Research Impact

The research impact of the AOSD-BR community can be quantified in terms of the increasing number of accepted papers in top SE conferences around the world, the number of citations, and the number of best paper awards and nominations received in international and national conferences. In addition, other relevant results are reported, such as the adoption of AO applications as benchmarks by the AOSD research community, and the participation of Brazilian researchers in program and organizing committees of several international conferences and workshops.

A quantitative indicator of the quality and relevance of the AOSD-BR research community is the visible increase of accepted papers in top SE conferences over the last years. Table III shows the number of papers published by the Brazilian research community from 1999 to 2011 in top software engineering (SE) conferences3 and the number and percentage of the total number of these accepted papers that are a contribution from the AOSD-BR community. In some cases, the contribution of our community is equal or superior to 50% of the total number of papers published by Brazilian researchers in these conferences: International Conference on SE (ICSE), International Conference on Software Maintenance (ICSM), Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), European Conference on Object-Oriented Programming (ECOOP), and Foundations on SE (FSE). We can also notice that the AOSD-BR community has gradually increased the number of accepted papers in the premier AOSD conference from 2002 to 2011. It is important to emphasize that all these conferences have a strict acceptance rate, and the papers published there represent the state-of-the-art of outstanding and high-quality research developed around the world. These publications also demonstrate the maturity and quality of research work developed by the AOSD-BR community.

Another important quality indicator of the AOSD-BR research is the citation number of papers published by this community in SE international and national conferences. Two research contributions [134], [58] from the community have received more than 200 citations each. The first one [134] is a contribution from the Software Productivity Group (SPG) from UFPE that illustrates the modularization of the persistence and distribution CC using AspectJ. The second one was the first relevant international publication of the SE Laboratory (LES) from PUC-Rio that quantifies and compares OO and AO implementations of classical design patterns [58] using software internal metrics. There are also several papers [23], [65], [53], from our community that have received more than 100 citations, even though many of them have been published only recently. One of these papers, “On the reuse and maintenance of AO software: An assessment framework” [53], presents a metrics suite to compare AO and OO implementations, including the new separation of concerns (SoC) metrics. To the best of our knowledge, this is the most cited SBES paper considering all SBES editions. It illustrates that a paper published in SBES can be read and recognized by the international SE research community. Many of the empirical assessments developed by the AOSD-BR community [23], [65] adopted this metric suite and helped to disseminate the development of several quantitative comparative studies between AO and OO implementations. Moreover, several other papers written by our community and published in international SE conferences have received between 40 to 100 citations. Finally, many papers published in SBES have been cited by other international and national papers. A complete list of citations to papers from the AOSD-BR research community can be found elsewhere [135]. Also, as a result of this high number of citations to their publications, six researchers from AOSD-BR community are listed in the Top 100 SE research ranking considering the citations over the last five years in the Microsoft Academic Search engine.

### Table III

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ICSE</td>
<td>4</td>
<td>2 (50%)</td>
</tr>
<tr>
<td>ICSM</td>
<td>7</td>
<td>4 (57%)</td>
</tr>
<tr>
<td>ECOOP</td>
<td>3</td>
<td>2 (67%)</td>
</tr>
<tr>
<td>OOPSLA</td>
<td>2</td>
<td>1 (50%)</td>
</tr>
<tr>
<td>FSE</td>
<td>2</td>
<td>1 (50%)</td>
</tr>
<tr>
<td>CSMR</td>
<td>15</td>
<td>6 (40%)</td>
</tr>
<tr>
<td>AOSD</td>
<td>9</td>
<td>9 (100%)</td>
</tr>
</tbody>
</table>

Several papers have been nominated to the best paper award in the AOSD – the premier international conference on AOSD, and SBES – the main Brazilian SE symposium. Three

---

3Only papers with at least one author from a Brazilian institution.

4Source: Google Scholar.
papers have been nominated at the AOSD conference, one in 2010 [136], another in 2009 [95], and the other in 2005 [57]. Besides, our community has received 11 nominations for the SBES best paper award along the last decade (2001-2010), and received the best paper award on three occasions [56], [101], [26]. This reflects the high quality and recognition of the SE international and national communities on the research work developed by AOSD-BR groups. A list of all awards received by AOSD-BR research work is available elsewhere [135].

Another contribution of this community is the development and evolution of two application benchmarks that were used by several researchers around the world: HealthWatcher [134] and MobileMedia [23]. HealthWatcher [134] is a real web information system that stores information about the public health system and registers complaints from citizens. HealthWatcher was first implemented in Java programming language and the Software Productivity Research Group from UFPE derived an AspectJ implementation of it. From this point, both Java and AspectJ versions were evolved, creating several versions that were used in several studies at Brazilian and international institutions. For example, Health Watcher was the recommended case study for the papers submitted to the Early Aspects at ICSE 2007 (Workshop in AO Requirements Engineering and Architecture Design). At this moment, HealthWatcher encompasses nine subsequent releases that provide support to the execution of AO maintainability studies. Our main partner on the evolution of HealthWatcher is the Lancaster University, where the versions of HealthWatcher are publicly available.

MobileMedia [23] is a Software Product Line (SPL) exemplar for media management on mobile devices. MobileMedia was first implemented in Java and AspectJ by Brazilian students at Lancaster University. After a few years, the current version of MobileMedia gathered contributions from a set of 16 institutions. Each of these contributed in many different ways. For example, some institutions’ members directly provided initial MobileMedia artifacts that followed AO design guidance cooperatively developed by them. Other partners contributed by reviewing the artifacts according to their area of expertise. Some contributions were offered by experts in specific techniques used to produce the MobileMedia artifacts, such as goal models and architecture descriptions. At this moment, MobileMedia encompasses eight subsequent releases that support the execution of studies of AO SPL maintainability. In addition, MobileMedia has already been successfully used in a number of assessments of AOSD [23], [86].

Last but not least, AOSD-BR researchers now participate in program committees of important international conferences (AOSD and ICSE) and have organized and participated in the technical program committee of several workshops (Early Aspects, Assessment of Contemporary Modularization Techniques - ACOM, Workshop on Empirical Evaluation of Software Composition Techniques - ESCOT) in conjunction with top SE conferences.

Acknowledgements.

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), CNPq grant 573964/2008-4.

References


http://www.springsource.org/
[67] B. Silva et al., “Concern-based cohesion as change proneness indicator: An initial empirical study”, in ICSE WETSoM 2011, pp. 52−58.